GNU Emacs Manual　(Japanese Translation)

GNU Emacs Manual(Japanese Translation)

Updated for Emacs Version 29.1

Richard Stallman et al.

This is the *GNU Emacs Manual*, updated for Emacs version 29.1.

Copyright © 1985–1987, 1993–2023 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 978-0-9831592-8-5

Cover art by Etienne Suvasa; cover design by FSF staff.

Short Contents

Table of Contents

前書き

このマニュアルは、Emacs エディターの使い方と、簡単なカスタマイズについて記述します。あなたがプログラマーではなくても、Emacs を簡単にカスタマイズできますが、カスタマイズに興味がない場合、カスタマイズのヒントを無視できます。

これは主にリファレンスマニュアルですが、入門書としても使用できます。Emacs に馴染みがない場合は、このマニュアルを読む前に、総合的で、learn-by-doing 形式のチュートリアルから始めることを推奨します。チュートリアルを実行するには、Emacs を開始して `C-h t` ("control h の後に t" のこと) とタイプします。そのチュートリアルでは、コマンドについて、それを実行するときと、その結果について説明されています。このチュートリアルは複数の言語で利用できます。

最初に読むときは、チャプター 1 と 2 はざっと目を通してください。これらのチャプターでは、このマニュアルの表記規約と、Emacs 表示スクリーンの一般的な外観を説明しています。このチャプターで説明されている疑問については注記してあるので、あとで参照することができます。チャプター 4 を読み終えたら、そこで示されたコマンドを練習してみましょう。続くいくつかのチャプターでは、頻繁に使用される基本的なテクニックと概念について説明します。これらは充分に理解する必要があるので、納得するまで試してください。

チャプター 14 から 19 では、多くの種類の編集に置いて便利な、中間レベルの機能について説明します。チャプター 20 以降ではオプションですが便利な機能について説明します。必要になったらこれらのチャプターを読んでください。

Emacs が正しく動作していないように見えるときは、一般的な問題に関するチャプターを読んでください。いくつかの一般的な問題に対処する方法 (Section 34.2 [Dealing with Emacs Trouble], page 537 を参照)、同様に Emacs のバグを報告する方法 (xrefBugs を参照) について説明されています。

特定のコマンドについてのドキュメントを探すには、インデックスを参照してください。キー (文字によるコマンド) と、コマンド名は、別のインデックスをもっています。用語ごとにクロスリファレンスがついた glossary(用語集) もあります。

このマニュアルは印刷された書籍、Info ファイルとしても利用できます。Info ファイルは Emacs 自身から読むか、Info プラグラムで読むためのものです。Info ファイルは GNU システムの中では主要なドキュメント形式です。Info ファイルと印刷された書籍は、ほぼ同じテキストを含んでおり、GNU Emacs と共にが畏怖される同じソースファイルから生成されます。

GNU Emacs は Emacs エディターファミリーの 1 つです。多くの Emacs エディターがあり、それらは一般的な原理と構成を共有します。Emacs の背後にある主義、および Emacs の開発から得られる教訓についての情報については *Emacs, the Extensible, Customizable Self-Documenting Display Editor* を参照してください。これは `https://dspace.mit.edu/handle/1721.1/5736` から入手できます。

このバージョンのマニュアルは、主に GNU および Unix システムにインストールされた GNU Emacs の使用を意図しています。GNU Emacs は MS-DOS、Microsoft Windows、Macintosh システムでも使用できます。このマニュアルの Info ファイル版には、それらのシステムでの Emacs の使用についての、追加情報が含まれています。これらのシステムは異なるファイル名の構文を使用します。それに加えて MS-DOS ではすべての GNU Emacs 機能をサポートしません。Windows での Emacs の使用については、See Appendix H [Microsoft Windows], page 608.Macintosh(および GNUstep) での Emacs の使用については、See Appendix F [Mac OS / GNUstep], page 602.

ディストリビューション

GNU Emacs は *free software(*フリーソフトウェアー、自由なソフトウェアー*)* です。これはすべての人が自由に使用でき、特定の条件の元に自由に再配布できることを意味します。GNU Emacs はパブリックドメイン (public domain: 特許権の消滅状態) ではありません。copyright(版権) されており、配布については制限があります。しかし、それらの制限は良き共同的な市民 (good cooperating citizen) が行ないたいと欲するであろう、すべてのことを許すようデザインされています。なにが許されていないか、それはあなたから取得するかもしれない GNU Emacs の任意のバージョンの更なる共有を妨げる試みです。これの正確な条件は Emacs の GNU General Public License で見ることができ、このマニュアルにも記載されています [1]。 Appendix A [Copying], page 555 を参照してください。

GNU Emacs のコピーを入手する 1 つの方法は、それを所有する他の誰かから入手する方法です。これを行なうための許可を求めたり、他の誰かに告げる必要はありません。ただコピーするだけです。インターネットへアクセスできるなら、匿名 FTP から GNU NEmacs の最新のディストリビューションバージョンを入手できます。わたしたちのウェブサイトについての詳細は、`https://www.gnu.org/software/emacs`を参照してください。

コンピューターを購入したときに、GNU Emacs を入手するかもしれません。コンピューター業者は、他のすべての人に適用されるのと同じ条件で、コピーを自由に配布できます。これらの条件は、コンピューター業者がソースにたいして行なった変更を含む完全なソースをあなたに与えることと、General Public License の通常の条件の下に、入手した GNU Emacs をあなたが再配布できることを要求します。言い換えると、そのプログラムはあなたが入手したときはフリーでなければならず、業者にとっては単にフリーという訳ではありません。

GNU Emacs が有用だと思ったら、わたしたちの作業をサポートするために、どうか Free Software Foundation に寄付を送ってください。合衆国では Free Software Foundation への寄付は、税金が控除されます。職場で GNU Emacs を使用している場合は、どうかその企業に寄付を行なうよう提案してください。寄付をするには、`https://my.fsf.org/donate/`を参照してください。あなたが手助けできる他の方法については、`https://www.gnu.org/help/help.html`を参照してください。

わたしたちは、このマニュアルと、Robert J. Chassell による *An Introduction to Programming in Emacs Lisp* のハードコピーも販売しています。あなたは、わたしたちのオンラインショップ `https://shop.fsf.org/`を訪れることができます。販売による収益は、Free Software Foundation の目的 — すなわち新しいソフトウェアーの開発、GNU Emacs を含む既存のプログラムの改良をサポートします。

Free Software Foundation に連絡する必要がある場合は、`https://www.fsf.org/about/contact/`を参照するか、下記に手紙を送ってください

Free Software Foundation
51 Franklin Street, Fifth Floor
Boston, MA 02110-1301
USA

[1] このマニュアル自身は GNU Free Documentation License により保護されています。このライセンスの精神は General Public License と同様ですが、よりドキュメントに適したものです。Appendix B [GNU Free Documentation License], page 566 を参照してください。

Acknowledgments

Contributors to GNU Emacs include Jari Aalto, Eric Abrahamsen, Per Abrahamsen, Tomas Abrahamsson, Jay K. Adams, Alon Albert, Michael Albinus, Nagy Andras, Benjamin Andresen, Ralf Angeli, Dmitry Antipov, Joe Arceneaux, Emil ^^c3^^85str^^c3^^b6m, 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, Stephen Berman, Jonas Bernoulli, Karl Berry, Anna M. Bigatti, Ray Blaak, Martin Blais, Jim Blandy, Johan Bockg^^c3^^a5rd, Jan B^^c3^^b6cker, Joel Boehland, Lennart Borgman, Per Bothner, Terrence Brannon, Frank Bresz, Peter Breton, Emmanuel Briot, Kevin Broadey, Vincent Broman, Michael Brouwer, David M. Brown, Ken Brown, Stefan Bruda, Damien Cassou, Daniel Colascione, Georges Brun-Cottan, Joe Buehler, Scott Byer, W^^c5^^82odek Bzyl, Tino Calancha, Bill Carpenter, Per Cederqvist, Hans Chalupsky, Chris Chase, Bob Chassell, Andrew Choi, Chong Yidong, Sacha Chua, Stewart Clamen, James Clark, Mike Clarkson, Glynn Clements, Andrea Corallo, Andrew Cohen, Daniel Colascione, Christoph Conrad, Ludovic Court^^c3^^a8s, Andrew Csillag, Toby Cubitt, Baoqiu Cui, Doug Cutting, Mathias Dahl, Yue Daian, 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^^c3^^a4rv, 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^^c3^^b6rn Einarsson, Tsugutomo Enami, David Engster, Hans Henrik Eriksen, Michael Ernst, Ata Etemadi, Frederick Farnbach, Oscar Figueiredo, Fred Fish, Steve Fisk, Thomas Fitzsimmons, Karl Fogel, Gary Foster, Eric S. Fraga, Romain Francoise, Noah Friedman, Andreas Fuchs, Shigeru Fukaya, Xue Fuqiao, Hallvard Furuseth, Keith Gabryelski, Peter S. Galbraith, Kevin Gallagher, Fabi^^c3^^a1n E. Gallina, Kevin Gallo, Juan Le^^c3^^b3n Lahoz Garc^^c3^^ada, Howard Gayle, Daniel German, Stephen Gildea, Julien Gilles, David Gillespie, Bob Glickstein, Nicolas Goaziou, Deepak Goel, David De La Harpe Golden, Boris Goldowsky, David Goodger, Chris Gray, Kevin Greiner, Michelangelo Grigni, Odd Gripenstam, Kai Gro^^c3^^9fjohann, Michael Gschwind, Bastien Guerry, Henry Guillaume, Dmitry Gutov, 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, Torsten Hilbrich, Anders Holst, Jeffrey C. Honig, J^^c3^^bcrgen H^^c3^^b6tzel, Tassilo Horn, Kurt Hornik, Khaled Hosny, 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 J^^c3^^b8rgensen, Tomoji Kagatani, Brewster Kahle, Tokuya Kameshima, Lute Kamstra, Stefan Kangas, Ivan Kanis, David Kastrup, David Kaufman, Henry Kautz, Taichi Kawabata, Taro Kawagishi, Howard Kaye, Michael Kifer, Richard King, Peter Kleiweg, Karel Kl^^c3^^ad^^c4^^8d, Shuhei Kobayashi, Pavel Kobyakov, Larry K. Kolodney, David M. Koppelman, Koseki Yoshinori, Robert Krawitz, Sebastian Kremer, Ryszard Kubiak, Tak Kunihiro, Igor Kuzmin, David K^^c3^^a5gedal, Daniel LaLiberte, Karl Landstrom, Mario Lang, Aaron Larson, James R. Larus, Gemini Lasswell, 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, Leo Liu, Emilio C. Lopes, Martin Lorentzson, Dave Love, Eric Ludlam, K^^c3^^a1roly L^^c5^^91rentey, Sascha L^^c3^^bcdecke, Greg McGary, Roland McGrath, Michael McNamara, Alan Mackenzie, Christopher J. Madsen, Neil M. Mager, Arni Magnusson, Artur Malabarba, Ken Manheimer, Bill Mann, Brian Marick, Simon Marshall, Bengt Martensson, Charlie Martin, Yukihiro Matsumoto, Tomohiro Matsuyama, David Maus, Thomas May, Will Mengarini, David Megginson, Jimmy Aguilar Mena, Stefan Merten, Ben A. Mesander, Wayne Mesard, Brad Miller, Lawrence Mitchell, Richard Mlynarik, Gerd M^^c3^^b6llmann, Dani Moncayo, 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 Nik^^c5^^a1i^^c4^^87, Jeff Norden, Andrew Norman, Theresa O'Connor, Kentaro Ohkouchi, Christian Ohler, Kenichi Okada, Alexandre Oliva, Bob Olson, Michael Olson, Takaaki Ota, Mark Oteiza, Pieter E. J. Pareit, Ross Patterson, David Pearson, Juan Pechiar, Jeff Peck, Damon Anton Permezel, Tom Perrine, William M. Perry, Per Persson, Jens Petersen, Nicolas Petton, Daniel Pfeiffer, Justus Piater, Richard L. Pieri, Fred Pierresteguy, Fran^^c3^^a7ois Pinard, Daniel Pittman, Christian Plaunt, Alexander Pohoyda, David Ponce, Noam Postavsky, Francesco A. Potort^^c3^^ac, Michael D. Prange, Mukesh Prasad, Steve Purcell, 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, Phil Sainty, James B. Salem, Masahiko Sato, Timo Savola, Jorgen Sch^^c3^^a4fer, Holger Schauer, William Schelter, Ralph Schleicher, Gregor Schmid, Michael Schmidt, Ronald S. Schnell, Philippe Schnoebelen, Jan Schormann, Alex Schroeder, Stefan Schoef, Rainer Sch^^c3^^b6pf, 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 ^^c5^^a0imko, Espen Skoglund, Rick Sladkey, Lynn Slater, Chris Smith, David Smith, JD Smith, Paul D. Smith, Wilson Snyder, William Sommerfeld, Simon South, Andre Spiegel, Michael Staats, Thomas Steffen, Ulf Stegemann, Reiner Steib, Sam Steingold, Ake Stenhoff, Philipp Stephani, Peter Stephenson, Ken Stevens, Andy Stewart, Jonathan Stigelman, Martin Stjernholm, Kim F. Storm, Steve Strassmann, Christopher Suckling, Olaf Sylvester, Naoto Takahashi, Steven Tamm, Jan Tatarik, Jo^^c3^^a3o T^^c3^^a1vora, Luc Teirlinck, Jean-Philippe Theberge, Jens T. Berger Thielemann, Spencer Thomas, Jim Thompson, Toru Tomabechi, David O'Toole, Markus Triska, Tom Tromey, 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

Zieli^^c5^^84ski, Ian T. Zimmermann, Reto Zimmermann, Neal Ziring, Teodor Zlatanov, and Detlev Zundel.

あなたは Emacs のマニュアル読んでいるところです。Emacs は、セルフドキュメント方式で、カスタマイズ可能で、拡張可能エディターであり、GNU の先進性を具現化したものです (GNU (GNU's Not Unix) の 'G'は発音します)。

Emacs は先進的 *(advanced)* であるというのは、単純な挿入と削除だけでなくプロセスの制御、プログラムのインデントの自動化、複数ファイルの同時表示、ローカルファイルと同様なリモートファイルの編集、整形済みテキストの編集、文字、単語、行、文、段落、ページを扱うのと同様に、異なるプログラミム言語の式やコメントを扱う機能なども提供するからです。

セルフドキュメント方式 *(Self-documenting)* とは、いつでもヘルプコマンドとして知られる、特別なコマンドを使うことができることです。これはどのようなオプションがあるのか、コマンドが何をするのかを見つけたり、与えられたトピックと関連するすべてのコマンドを見つけることができるコマンドです。Chapter 7 [Help], page 42 を参照してください。

カスタマイズ可能 *(Customizable)* とは、シンプルな方法で Emacs コマンドの動作を簡単に変更できるということです。たとえば、'<**'で始まり '**>'で終わるようなコメントのプログラム言語を使っている場合は、Emacs のコメント操作コマンドに、これらの文字列を使うように指示できます (Section 23.5 [Comments], page 295 を参照してください)。別の例としては、カーソル動作 (上下左右) を再編成して使いやすくカスタマイズできます。Chapter 33 [Customization], page 499 を参照してください。

拡張可能 *(Extensible)* とは、単純なカスタマイズではなく、まったく新しいコマンドを作成できるということです。新しいコマンドは、Emacs 自身の Lisp 処理系で動作する、Lisp 言語でプログラムを記述されます。既存のコマンドは、編集を行なっている最中でさえ、Emacs を再起動することなく再定義できます。Emacs のほとんどの編集コマンドは Lisp で記述されています。Lisp で記述することも可能ですが、効率のために C で記述されているものもあります。拡張機能の記述はプログラミングですが、プログラマーでない人も後でそれを使うことができます。もしあなたが Emacs Lisp を学びたければ、Section "Preface" in *An Introduction to Programming in Emacs Lisp* を参照してください。

1 画面の構成

X ウィンドウシステムを使う GNU/Linux のようなグラフィカルなディスプレーの場合、Emacs はグラフィカルなウィンドウに表示されます。テキスト端末の場合、Emacs は端末スクリーン全体を表示領域として使います。Emacs が占有する画面スクリーンや、グラフィカルなウィンドウを指して、フレーム *(frame)* という用語を使用します。Emacs の振る舞いは、どちらのフレームでも同じです。通常は 1 個のフレームだけで始まりますが、必要ならば新たにフレームを作れます (Chapter 18 [Frames], page 194 を参照)。

それぞれのフレームにはいくつかの領域が含まれています。いちばん上のフレームはメニューバー *(menu bar)* で、メニューにある一連のコマンドにアクセスできます。グラフィカルなディスプレーでは、メニューバーのすぐ下にツールバー *(tool bar)* があり、アイコンをクリックすることにより編集コマンドを実行できます。いちばん下のフレームはエコーエリア *(echo area)* で、メッセージが表示されたり、Emacs が入力を求める際に使用されます。

(もしあれば) ツールバーの下とエコーエリアの間の、フレームの主要な領域の部分を、ウィンドウ *(the window)* といいます。このマニュアルでは “ウィンドウ” という言葉を、上記のような場合に使います。グラフィカルなディスプレーのシステムでは、“ウィンドウ” という言葉を違う意味で用いますが、上述したとおり、そのようなグラフィカルなウィンドウのことは、“フレーム” と呼ぶことにします。

Emacs のウィンドウには、バッファー *(buffer)* — 編集、または閲覧しているテキストやグラフィック — が表示されます。グラフィカルなディスプレーでは、ウィンドウの片側にスクロールバー *(scroll bar)* あり、これを使ってバッファー内をスクロールできます。ウィンドウのいちばん下の行は、モードライン *(mode line)* です。これには保存されてない変更や、使用されている編集モード、現在のライン番号など、バッファーについての様々な情報が表示されます。

Emacs を起動すると、通常フレームには 1 つのウィンドウが表示されます。しかしこのウィンドウを水平方向、または垂直方向に分割して複数のウィンドウを作成し、それぞれ異なるバッファーを表示することもできます (Chapter 17 [Windows], page 186 を参照)。

どんな時でも、1 つのウィンドウが選択されたウィンドウ *(selected window)* となります。グラフィカルなディスプレーでは、選択されたウィンドウには目立つカーソル (通常は塗りつぶされて点滅している) が表示され、他のウィンドウには目立たないカーソル (通常はぬりつぶされていない四角) が表示されます。テキスト端末では、選択されたウィンドウのカーソルだけが表示されます。選択されたウィンドウ上に表示されているバッファーを、カレントバッファー *(current buffer)* と呼び、それが編集が行われているバッファーとなります。多くの Emacs コマンドはカレントバッファーに暗黙に適用され、選択されてないウィンドウに表示されているテキストは参照用に使用します。もしグラフィカルなディスプレーで複数のフレームを使っている場合、特定のフレームを選択すると、そのフレームのウィンドウが選択されます。

1.1 ポイント

カーソルは、選択されたウィンドウで多くの編集コマンドが作用する場所を示し、その場所をポイント *(point)*[1] と呼びます。多くの Emacs コマンドはポイントをテキスト中で移動し、テキスト中のさまざまな箇所で編集できるようにします。マウスのボタン 1(通常は左ボタン) をクリックしても、ポイントを移動できます。

デフォルトでは、選択されたウィンドウ上にある、塗りつぶされたボックスのカーソルは文字の上に表示されますが、ポイントは 2 つの文字の間にあると考える必要があります。つまりポイントは、

[1] “point” という用語は、文字 ‘.’ に由来します。この文字は、現在ポイントと呼んでいる値を参照するための TECO(オリジナルの Emacs を記述していた言語) のコマンドです。

カーソルが重なっている文字の前にあります。たとえば、‘frob’というテキストで、‘b’にカーソルがある場合、ポイントは‘o’と‘b’の間にあります。その位置に‘!’という文字を挿入すると、‘fro!b’という結果になり、ポイントは‘!’と‘b’の間にあります。つまりカーソルは‘b’の上にあり、実行前と同じです。

Emacs でいくつかのファイルを編集して、各ファイルがそれぞれ専用のバッファーにある場合、各バッファには独自のポイント位置があります。バッファーが表示されていなくても、後で表示されるときに備えて、ポイント位置を記録しています。1 つのフレームに複数のウィンドウがある場合、各ウィンドウには独自のポイント位置があります。

Emacs がカーソルをどのように表示するか制御する方法については、Section 11.21 [Cursor Display], page 99 を参照してください。

1.2 エコーエリア

フレームの 1 番下の行は、エコーエリア *(echo area)* です。ここは、いろいろな目的向けの短いテキスト表示に使われます。

エコーエリアという名前は、あなたが打った文字がエコーされることが由来で、これは複数の文字からなるコマンドが表示されることを意味します。1 文字のコマンドをエコーすることはありません。複数の文字からなるコマンド (Section 2.2 [Keys], page 12 を参照) の途中で、入力中に 1 秒以上間を置くとエコーされます。Emacs はそれまでに入力されたコマンドの文字を表示し、ユーザーに残りの文字の入力を促します。いったんエコーが始まると、コマンドの残りは、打つと同時にただちにエコーされます。これは、タイプに自信のあるユーザーには速い応答を提供する一方で、自信のないユーザーには最大限のフィードバックを与えるための機能です。

エコーエリアは、コマンドを実行できなかったときに、エラーメッセージ *(error message)* を表示するためにも使用されます。エラーメッセージと共に、ビープ音が鳴ったり、画面が点滅する場合もあります。

エコーエリアに有用なメッセージを表示するコマンドもあります。これらの有用なメッセージは、エラーメッセージによく似ていますが、ビープ音を伴わず点滅もしません。たとえば、コマンド C-x =(Ctrlを押したまま xをタイプし、Ctrlを離してから=をタイプします) は、テキスト中のポイントの文字位置と、ウィンドウの現在の列位置を示すメッセージを表示します。処理に時間のかかるコマンドでは、実行中に‘...’(どの程度進行したかをパーセント表示で示す場合もあります) で終わるメッセージをエコーエリアに表示し、完了時には‘done’を最後に付け加えることがよくあります。

エコーエリアに表示される有益なメッセージは、*Messages*と呼ばれる特別なバッファーに保存されます (まだバッファーについては説明していませんが、詳細は Chapter 16 [Buffers], page 176 を参照)。画面上に短時間しか表示されないメッセージを見逃してしまった場合には、*Messages* バッファーに切り替えて、もう一度そのメッセージを見ることができます。*Messages*のサイズは、ある行数に制限されています。変数 message-log-maxは、その行数を指定します (まだ変数については説明していませんが、詳細は Section 33.2 [Variables], page 508 を参照)。いったんバッファーがこの行数を超えると、最後に 1 行加わるごとに先頭の 1 行を削除します。

Emacs がエコーエリアをどのように使用するか制御するオプションについては、Section 11.24 [Display Custom], page 102 を参照してください。

エコーエリアはミニバッファー *(minibuffer)* の表示にも使われます。これは、編集しようとするファイル名のような、コマンドへの引数を読むのに使われるウィンドウです。ミニバッファーが使用されているとき、エコーエリアにはプロンプト文字列 *(prompt string)* で始まるテキストが表示され、エコーエリアが一時的に選択されたウィンドウとなり、カーソルもその行に表示されます。C-gを打つと、いつでもミニバッファーから抜けられます。Chapter 5 [Minibuffer], page 28 を参照してください。

1.3 モードライン

ウィンドウの最後の行はモードライン *(mode line)* であり、そのウィンドウで何が進行しているか表示します。ウィンドウが 1 つしかなければモードラインはエコーエリアのすぐ上に表示されます。フレームでは最後から 2 番目の行になります。グラフィカルなディスプレーではモードラインは立体的に描画されます。Emacs は選択されたウィンドウのモードラインが目立つように、通常は選択されていないウィンドウとは異なるカラーで描画します。

モードラインに表示されるテキストは以下の書式です:

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

テキスト端末では、上記テキストの後ろからウィンドウの右端まで一連のダッシュ表示されます。これらのダッシュはグラフィカルなディスプレーでは省略されます。

*cs*とその後ろのコロンは、カレントバッファーのキャラクターセットと改行の規則を説明しています。通常 Emacs はこれらの設定を自動的に処理しますが、このメッセージが便利なときもあります。

*cs*はバッファーのキャラクターセットを説明します (Section 19.5 [Coding Systems], page 223 を参照)。もしこれがダッシュ(‘`-`’) の場合、特定のキャラクターセットの処理が行われていないことを意味します (例外として、以降で説明する行末規則があります)。‘`=`’の場合は、変換が行われていないことを意味し、通常はテキストに非テキストデータが含まれているときに使用されます。他の文字はさまざまなコーディングシステム *(coding systems)* — たとえば ‘`1`’は ISO Latin-1 を表します。

テキスト端末では *cs*の前に追加で 2 つの文字が表示され、それによりキーボード入力と端末出力のコーディングシステムが示されます。さらに何らかの入力メソッドを使用している場合は、*cs*の前に入力メソッドを識別する文字列が表示されます (Section 19.3 [Input Methods], page 220 を参照)。

*cs*の後ろの文字は、通常コロンです。もし違う文字が表示されている場合、それはファイルのエンコーディングに特別な行末規則が使われていることを意味します。通常ファイル内のテキストの各行は改行文字 *(newline characters)* で区切られていますが、他の 2 つの規則が使われる場合もあります。MS-DOS のファイルを編集する場合にはキャリッジリターン (carriage-return) とラインフィード (linefeed) が使われ、コロンではなくバックスラッシュ(‘`\`’) または ‘`(DOS)`’(オペレーティングシステムに依存する) が表示されます。古いマッキントッシュシステムのファイルでは、改行文字の代わりにキャリッジリターン (carriage-return) が使われ、そのような場合コロンではなくスラッシュ(‘`/`’) または ‘`(Mac)`’が表示されます。いくつかのシステムでは行の区切りとして改行文字を使う場合、コロンではなく ‘`(Unix)`’と表示されます。

`emacsclient` (Section 31.6.2 [Invoking emacsclient], page 471 を参照) で作成されたフレームでは、次の文字に ‘`@`’が表示されます。これは通常、デーモン (Section 31.6 [Emacs Server], page 469 を参照) として実行中の Emacs プロセスのフレームにたいして表示されます。

モードラインの次の要素は *ch*で示される文字列です。2 つのダッシュ(‘`--`’) が表示されている場合、ウィンドウに表示されているバッファーとディスク上のファイルの内容が同じことを意味し、たとえばバッファーが未変更 (unmodified) の場合です。バッファーが変更されている場合には 2 つの星印 (‘`**`’) が表示されます。読み出し専用のバッファーの場合には、バッファーが編集されている場合には ‘`%*`’になり、バッファーが編集されていない場合には ‘`%%`’となります。

通常、*ch*の後ろの文字はダッシュ(‘`-`’) です。しかしカレントバッファーの `default-directory` (Section 15.1 [File Names], page 145 を参照) がリモートマシン上 (Section 15.1 [File Names], page 145 を参照) にある場合、かわりに ‘`@`’が表示されます。

*fr*は選択されているフレームの名前です (Chapter 18 [Frames], page 194 を参照)。これはテキスト端末でだけ表示されます。フレーム名の初期値は ‘`F1`’です。

*buf*は、ウィンドウに表示されているバッファーの名前です。通常は編集中のファイル名と同じです。Chapter 16 [Buffers], page 176 を参照してください。

*pos*はウィンドウの上またはウィンドウの下に、まだテキストがあるかを知らせます。もしバッファーが小さくてウィンドウに全体が表示されている場合、*pos*には‘`All`’が表示されます。そうではなくバッファーの一部が表示されているときには、バッファーの先頭が表示されている場合には‘`Top`’、バッファーの最後が表示されている場合には‘`Bot`’、‘*nn*`%`’と表示されている場合、*nn*はウィンドウの上部がバッファーのどこかをパーセント表示したものです。Size Indication(サイズ表示) モードでは、バッファー全体のサイズを表示できます。

*line*は、‘`L`’のあとに現在ポイントがある行の番号が続いたものです (Column Number(列番号) モードをオンにすると、現在の列番号も表示できます。Section 11.19 [Optional Mode Line], page 97 を参照)。

*major*は、そのバッファのメジャーモード *(major mode)* の名前です。メジャーモードはバッファーを編集する際の主要なモードで、Text モード、Lisp モード、C モードなどがあります。Section 20.1 [Major Modes], page 240 を参照してください。メジャーモード名の後ろに追加の情報を表示するメジャーモードもあります。たとえばCompilation buffer(コンパイルバッファー) やShell buffer(Shell バッファー) などは、サブプロセスの状態を表示します。

*minor*は有効になっているマイナーモード *(minor modes)* の一覧で、上位のメジャーモードに追加の機能を提供するための編集モードです。Section 20.2 [Minor Modes], page 241 を参照してください。

いくつかの機能は、それらが本当はマイナーモードではなくても、有効になっていればマイナーモードの一覧とともに表示されます。‘`Narrow`’は、表示中のバッファーが、そのテキストの一部のみを編集するように制限されていることを示します (Section 11.5 [Narrowing], page 81 を参照)。‘`Def`’は、キーボードマクロを定義中であることを示します (Chapter 14 [Keyboard Macros], page 137 を参照)。

さらにEmacsが再帰編集 (recursive edit) にあるときには、モードを囲んでいるカッコの周りに角カッコ (‘`[...]`’) が現れます。再帰編集中も別の再帰編集に入ると、角カッコは2重になります。再帰編集は、特定のバッファにだけ関係するものではなく、Emacs全体に影響するので、角カッコはすべてのウィンドウのモード行に表示されるか、まったく表示されないのどちらかです。Section 31.11 [Recursive Edit], page 484 を参照してください。

モードラインの外観は、その内容の書式と同様、変更できます。Section 11.19 [Optional Mode Line], page 97 を参照してください。さらにモードラインはマウスに反応します。モードラインの違う部分をクリックすることでさまざまなコマンドを実行できます。Section 18.5 [Mode Line Mouse], page 198 を参照してください。また、モードラインのマウス感応範囲上でマウスポインターをホバリングすると、モードライン上でクリックして呼び出すことができるコマンドに関する情報を表示するツールチップ (Section 18.19 [Tooltips], page 213 を参照) が表示されます。また、モードラインのマウスセンシティブ (mouse-sensitive: マウス感応) な部分の上にマウスをホバリングすると、モードライン上をクリックして呼び出せるコマンドに関する情報がツールチップ (Section 18.19 [Tooltips], page 213 を参照) が表示されます。

1.4 メニューバー

各Emacsフレームには通常、最上部にメニューバー *(menu bar)* があり、よく使われる操作を実行するために使用できます。これはあなた自身で簡単に確かめられますから、ここではそれらを列挙する必要はないでしょう。

マウスをサポートするディスプレー上では、マウスを使ってメニューバーからコマンドを選ぶことができます。メニューアイテムの後にある右矢印は、そのアイテムにサブメニュー *(submenu)* があることを示します。アイテムの最後に‘`...`’がある場合は、コマンドを実際に実行する前に、そのコマンドがキーボードから引数を読み取ることを意味します。

メニューのコマンドの中には、他のコマンドと同様、キーが割り当てられているものもあります。そのような場合、メニューのアイテムの後にキーバインディングが表示されます。メニューアイテムの完全なコマンド名や説明文を見るには、C-h kとタイプしてから、通常どおりにマウスでメニューバーを選択してください (Section 7.1 [Key Help], page 45 を参照)。

マウスを使う代わりにF10(コマンドmenu-bar-openを実行します) を押せば、メニューバーの最初のアイテムを呼び出すことができます。その後は矢印キー、またはC-bとC-f(左右)、C-pとC-n(上下) でメニュー操作できます。選択されたメニューアイテムを起動するときはRET、メニュー操作をキャンセルするときはC-gかESC ESC ESCを押します。(ただし、GUI ツールキットとともにビルドされたEmacsでは、メニューはそのツールキットにより描画および制御され、メニュー操作をキャンセルするキーシーケンスは上記の説明とは異なるかもしれません。)

テキスト端末では、エコーエリアからメニューバーのメニューにアクセスすることもできます。これを行うには、変数tty-menu-open-use-tmmをカスタマイズして、非nil値にします。その後はF10をタイプするとメニューをドロップダウンせずに、コマンドtmm-menubarが実行されます (M-`は常にtmm-menubarを呼び出します)。tmm-menubarでは、キーボードでメニューアイテムを選択できます。暫定的な候補がエコーエリアに表示されるので、上矢印か下矢印でメニューの異なるアイテムを表示し、RETを押せばアイテムを選択できます。各メニューアイテムを文字か数字で指定することもできます (通常メニューアイテム名のイニシャル)。この文字または数字とアイテム名は '==>' で区切られています。アイテムの文字または数字を押せばアイテムを選択することができます。

2 Characters, Keys and Commands

この章では、Emacs がコマンド入力に使う文字セット、およびキー *(keys)* とコマンド *(commands)* の基本的な概念と、それによって Emacs がどのようにキーボードやマウス入力を解釈するかを説明します。

2.1 ユーザー入力の種類

GNU Emacs は、主にキーボードを使うようにデザインされています。マウスを使ってメニューバーやツールバーの編集コマンドを実行することはできますが、キーボードを使う場合に比べて効率的ではありません。

Emacs にたいするキーボード入力は、ASCIIを大きく拡張したバージョンが基本となっています。'a'、'B'、'3'、'='や空白文字 (SPC) と表記します) などの単純な文字は、それぞれに対応するキーをタイプして入力します。RET、TAB、DEL、ESC、F1、Home、LEFTなどの制御文字なども、この方法で入力できますし、非英語キーボードの特定の文字も同様です (Chapter 19 [International], page 216 を参照)。

Emacs は修飾キー *(modifier keys)* を用いて入力された制御文字も認識します。よく使用される修飾キーは、Control(通常 Ctrl) というラベル) と、Meta(通常 Alt[1] というラベル) の 2 つです。たとえば、Control-aは Ctrlを押したままで aを押して入力しますが、これを短く C-aと記します。同様に、Meta-aまたは短く M-aは、Altを押したまま aを押すことです。修飾キーは英数文字以外のキーにも適用できます。例: C-F1、M-LEFT

ESCで始まる 2 文字キーシーケンスを使って、Meta 文字を入力することもできます。したがって M-aを ESC aと入力することができます。C-M-a(Ctrlと Altを両方押しながら aを押下) は ESC C-aと入力できます。Metaと違い、ESCは切り離された文字です。次の文字を押すとき ESCを押しっぱなしにするのではなく、ESCを押して離してから次の文字を入力します。この機能は Metaキーをあてにできない、一部のテキスト端末で有用です。

Emacs は追加の修飾キーをサポートします。Section 33.3.7 [Modifier Keys], page 523 を参照してください。

Emacs にはマウスボタンやマウスホイール、それにタッチパッドやタッチスクリーンのようなその他のポインティングデバイスを幅広くサポートしています。詳細については Section 2.3 [Mouse Input], page 13 を参照してください。

グラフィカルなディスプレーでは、ウィンドウマネージャーが M-TAB、M-SPC、C-M-d、C-M-lなどのキーボード入力をブロックするかもしれません。このような問題がある場合、ウィンドウマネージャーがこれらのキーをブロックしないようにカスタマイズしたり、影響を受ける Emacs のコマンドをリバインド (rebind) したりできます (Chapter 33 [Customization], page 499 を参照)。

単純な文字や制御文字、同様にマウスのクリックなどの非キーボード入力は、総じて入力イベント *(input events)* と呼ばれます。Emacs が内部で入力イベントをどのように処理するかについての詳細は、Section “Input Events” in *The Emacs Lisp Reference Manual* を参照してください。

2.2 キー

Emacs コマンドには、ただ 1 つの入力イベントで呼び出されるものが、いくつかあります。たとえば C-fはバッファーを 1 文字前方に移動します。他のコマンドは、C-x C-fや C-x 4 C-fのように、2 つ以上の入力イベントにより呼び出されます。

[1] 歴史的な理由により、Altのことを Metaという名前で参照します。

キーシーケンス *(key sequence)*、短く書くとキー *(key)* は、1 つの単位として考えることのできる、1 つまたはそれ以上の一連の入力イベントの集まりのことです。もし、あるキーシーケンスがコマンドを呼び出すような場合、それをコンプリートキー *(complete key)* と呼ぶことにします。たとえば `C-f`、`C-x C-f`、`C-x 4 C-f`などはコンプリートキーです。もし、あるキーシーケンスがコマンドを呼び出すほど十分長くないとき、それをプレフィクスキー *(prefix key)* と呼ぶことにします。たとえば前の例でいうと、`C-x`や `C-x 4`はプレフィクスキーです。すべてのキーシーケンスは、コンプリートキーかプレフィクスキーのどちらかになります。

プレフィクスキーは、その後の入力イベントと組み合わせて、もっと長いキーシーケンスを作るためのものです。たとえば `C-x` はプレフィクスキーなので、`C-x`と入力しただけではコマンドは呼び出されません。かわりに Emacs は更なる入力を待ちます (もし 1 秒以上入力がない場合、入力を促すために `C-x`がエコーされます。Section 1.2 [Echo Area], page 8 を参照)。`C-x`は、それに続く次の入力イベントと組み合わされる、2 イベントのキーシーケンスで、それはプレフィクスキー (`C-x 4`) など) のときもあれば、コンプリートキー (`C-x C-f`など) のときもあります。キーシーケンスの長さに制限はありませんが、実際に 3 つ、4 つ以上の入力イベントの場合は、ほとんどありません。

コンプリートキーに入力イベントを付け加えることはできません。たとえば、`C-f`はコンプリートキーなので、2 イベントのシーケンス `C-f C-k`は、1 つではなく 2 つのキーシーケンスです。

デフォルトでは Emacs のプレフィクスキーは `C-c`、`C-h`、`C-x`、`C-x RET`、`C-x @`、`C-x a`、`C-x n`、`C-x r`、`C-x t`、`C-x v`、`C-x 4`、`C-x 5`、`C-x 6`、`ESC`、`M-g`です (`F1`と `F2`は `C-h`と `C-x 6`のエイリアス)。このリストは不変のものではありません。Emacs をカスタマイズすれば、新しいプレフィクスキーを作ることができます。標準のプレフィクスキーを無効にすることさえできますが、これはほとんどのユーザーにたいして推奨はできません。たとえばプレフィクス定義 `C-x 4`を削除すると、`C-x 4 C-f`は無効なキーシーケンスになります。Section 33.3 [Key Bindings], page 519 を参照してください。

プレフィックスキーのあとにヘルプ文字 (`C-h`や `F1`) を押すと、そのプレフィックスで始まるコマンド一覧を表示できます。唯一の例外は `ESC`です。`ESC C-h`は `C-M-h`と同じで、これは何かまったく別のことを行うコマンドです。しかし `F1`ならば、`ESC`で始まるコマンドの一覧を表示できます。

2.3 マウス入力

Emacs はデフォルトではマウスの左ボタンクリックによるカーソルのセット、マウスポインターのドラッグによる領域選択のような通常のマウスアクションをすべてサポートしています。キーボードイベントをバインドするのと同じ方法によって、すべてのマウスアクションはコマンドへのバインドに使用することができます (Section 2.2 [Keys], page 12 を参照)。このセクションでは Emacs におけるマウスの使用についての概観を説明します。Emacs のマウスコマンドについての詳細は Section 18.1 [Mouse Commands], page 194 以降のセクションを参照してください。

マウスの左ボタンがクリックされると、Emacs は `mouse-1`イベントを受信します。`C-h c`をタイプした後にマウスの左ボタンを押せば、このイベントにバインドされているコマンドを確認できます。同様にマウスの中ボタンは `mouse-2`、右ボタンは `mouse-2`です。ホイール付きマウスのホイールイベントは通常は `wheel-down`と `wheel-up`、あるいはオペレーションシステムの設定によっては `mouse-4`と `mouse-5`にバインドされています。

一般的に X などのレガシーシステムや端末 (Section 18.22 [Text-Only Mouse], page 215 を参照) では `mouse-4`と `mouse-5`、それ以外のシステムではすべて `wheel-down`と `wheel-up`がレポートされるでしょう。

水平スクロール用ホイールがある一部のマウス、それにタッチパッドも同じように水平スクロールをサポートしています。これらのイベントは X などのレガシーシステムや端末では `mouse-6`と `mouse-7`、それ以外のすべてのシステムでは `wheel-left`と `wheel-right`がレポートされます。

たとえばMetaキーを押したままマウスの中ボタンをクリックのように、修飾キーとマウスイベントを組み合わせて特別なコマンドをトリガーさせることができます。この例だとイベント名は`M-mouse-2`になるでしょう。

タッチスクリーンのイベント処理にたいしてコマンドをバインドできるシステムもあります。この場合のイベント名は`touchscreen-update`と`touchscreen-end`です。

2.4 キーとコマンド

このマニュアルは、特定のキーが何を行うかを説明するページばかりです。しかし、Emacsは直接キーに意味を与えてはいません。そのかわりに、Emacsは名前を付けたコマンド *(commands)* に意味を持たせ、キーとコマンドをバインディング *(binding)* することによって、キーに意味を与えています。

すべてのコマンドには、プログラマーが選んだ名前が付いています。名前は、たとえば`next-line`、`forward-word`のように、いくつかの英単語をダッシュで区切って作られます。内部的には、それぞれのコマンドはLispの関数 *(function)* の特別な型で、コマンドに関連付けられたアクションは、関数を実行することによって機能します。Section “What Is a Function” in *The Emacs Lisp Reference Manual* を参照してください。

キーとコマンドの間のバインディングは、*keymaps* というテーブルに記憶されます。Section 33.3.1 [Keymaps], page 519 を参照してください。

“`C-n`は下に1行動きます” という言い方は、通常の使用では関係ないものの、Emacsをカスタマイズする上では重要になる点を隠蔽しています。1行下に移動するコマンドは`next-line`です。`C-n`が`next-line`にバインドされているから効果があるのです。もし`C-n`をコマンド`forward-word`にリバインドしたら、`C-n`で1語前方に動くことになります。

厳密に言えばキーはコマンドにバインドされているだけですが、このマニュアルでは`C-n`をコマンドであるかのような言い回しをするときがあります。そのようなときは、処理を実行させるキーの後ろに、本当に処理を行うコマンドの名前をカッコ内に記します。たとえば、“コマンド`C-n` (`next-line`) は、ポイントを垂直下方に移動します” というときは、コマンド`next-line`がポイントを垂直下方に移動し、それは通常`C-n`にバインドされている、ということを意味します。

カスタマイズについて議論したので、変数 *(variables)* にもふれておくべきでしょう。コマンドの説明で、“これを変更する場合、変数`mumble-foo`をセットしてください” というときがあります。変数とは、値を保存するときに使用する名前のことです。このマニュアルに記載されている変数は、ほとんどがカスタマイズに関するものです。いくつかのコマンド、およびEmacsのある部分は、変数を調べてその変数にセットされた値により、動作がかわります。カスタマイズに興味がでるまでは、、変数に関する情報は無視してかまいません。その後で変数 (Section 33.2 [Variables], page 508 を参照) の基本を読めば、特定の変数についての情報に合点がいくでしょう。

3 Entering and Exiting Emacs

この章では、Emacs を起動する方法、および終了する方法を説明します。

3.1 Emacs の起動

Emacs を呼び出す通常の方法は、シェルコマンド `emacs`です。GUI 端末上で実行される Unix シェルからは、`emacs &`により Emacs をバックグラウンドで実行できます。この方法だと Emacs が端末ウィンドウに結びつけられないため、他のシェルコマンドを実行できます (MS-Windows で Emacs を開始する方法については、Section H.1 [Windows Startup], page 608 を参照)。

Emacs を起動すると、初期フレームは‘`*GNU Emacs*`’という名前の特別なバッファーを表示します。このスタートアップ画面 *(startup screen)* には、Emacs についての情報と、初心者にとって便利な一般的タスクへのリンクが含まれています。たとえば‘`Emacs Tutorial`’というリンクは Emacs のチュートリアルを開きます。これはコマンド `C-h t` (`help-with-tutorial`) と同じです。リンクをアクティブにするには、ポイントをそこに動かして `RET`をタイプするか、`mouse-1`(マウスの左ボタン) をクリックしてください。

コマンドライン引数を使うと、Emacs が起動直後に 1 つ以上のファイルをアクセスするよう指示できます。たとえば `emacs foo.txt`は、‘`foo.txt`’の内容を表示するバッファーとともに Emacs を起動します。これは他のエディターとの互換性により存在する機能で、シェルから短い編集セッションを始めるときのためにデザインされています。Emacs をこの方法で呼び出すと、初期フレームは 2 つのウィンドウに分割されます。1 つは指定されたファイルで、もう 1 つはスタートアップ画面です。Chapter 17 [Windows], page 186 を参照してください。

一般的に、ファイルを編集するたびに新たに Emacs を起動するのは不必要で無駄です。Emacs を使うときの推奨方法は、Emacs を 1 度だけ起動する方法で、ログインしたら起動して、同じ Emacs セッションですべての編集作業を行うのです。1 つ以上のファイルをアクセスする方法は、Chapter 15 [Files], page 145 を参照してください。この方法で Emacs を使うと、Emacs のセッションはキルリング (kill ring)、レジスター (registers)、アンドゥヒストリー (undo history)、マークリング (mark ring) などの、値をもつコンテキストを蓄積するので、これを共有すれば編集がより快適になります。これらの機能については、このマニュアルの後で説明します。

Emacs を実行中に、他のプログラムからファイルを編集する場合、既存の Emacs セッションのファイルを開くために、`emacsclient`というヘルパープログラムを使うことができます。Section 31.6 [Emacs Server], page 469 を参照してください。

コマンドライン引数を使って、Emacs に Lisp ファイルをロードして初期フレームに適用させたりできます。Appendix C [Emacs Invocation], page 574 を参照してください。

変数 `inhibit-startup-screen`が非 `nil`の場合、Emacs はスタートアップ画面を表示しません。この場合、コマンドラインに 1 つ以上のファイルが指定されていれば、Emacs は単にそれらのファイルを表示し、指定されていないときは Lisp の式を対話的に評価できる、`*scratch*`という名前のバッファーを表示します。Section 24.10 [Lisp Interaction], page 330 を参照してください。変数 `inhibit-startup-screen`のセットは、Emacs のカスタマイズ機能 (Section 33.1 [Easy Customization], page 499 を参照してください) を使うか、初期設定ファイル (Section 33.4 [Init File], page 528 を参照してください) を編集して行うことができます。[1]

変数 `initial-buffer-choice`にファイルやディレクトリーの名前をセットすることにより、Emacs のスタートアップ時にファイルやディレクトリーを表示させることもできます。

[1] `site-start.el`の中で `inhibit-startup-screen`をセットしても機能しません。なぜならスタートアップ画面は `site-start.el`が読み込まれる前にセットアップされるからです。`site-start.el`についての情報は、Section 33.4 [Init File], page 528 を参照してください。

initial-buffer-choiceの値に、その後に表示するバッファーを戻す関数 (引数なし) をセットすることもできます。initial-buffer-choiceが非 nilの場合、コマンドラインにファイルを指定しても、それらのファイルは表示されますが、初期画面としては表示されません。

3.2 Emacs からの exit

C-x C-c　Emacs を kill(終了) します。(save-buffers-kill-terminal)。

C-z　テキスト端末では Emacs をサスペンドします。グラフィカルなディスプレーでは選択されたフレームをアイコン化 (または “最小化”) します (suspend-frame)。

Emacs を *Kill* するというのは、Emacs プログラムを終了するという意味です。これを行うには、C-x C-c (save-buffers-kill-terminal) とタイプします。2 文字キーシーケンスが使われているのは、アクシデントにより間違ってタイプしづらくするためです。もし変更されたファイルがある場合、C-x C-cをタイプすると、Emacs とそれらのバッファーを巡回して、バッファーを保存するか問い合わせます。それらすべてを保存しない場合、未保存の変更が失われてしまう前に、もう一度問い合わせます。サブプロセスがまだ実行中の場合にも、Emacs を kill するとサブプロセスも kill されるので、問い合わせを行います (Section 31.5 [Shell], page 457 を参照してください)。

もし Emacs をサーバーとして使っている場合、C-x C-cは特別に振る舞います。もしクライアントフレームからタイプした場合は、クライアントのコネクションをクローズします。Section 31.6 [Emacs Server], page 469 を参照してください。

Emacs はオプションで、kill したときに表示していたファイルなどの、セッション情報を記録することができます。この情報は次回 Emacs を起動するとき利用可能です。Section 31.10 [Saving Emacs Sessions], page 482 を参照してください。

変数 confirm-kill-emacsの値が非 nilの場合、C-x C-cはその値が関数だとみなして、その関数を呼び出します。その関数呼び出しの結果が非 nilの場合、セッションは kill され、そうでない場合、Emacs は実行を続けます。confirm-kill-emacsの値として使うのに適した関数の 1 つが、yes-or-no-pです。confirm-kill-emacsのデフォルト値は nilです。

変数 confirm-kill-processesの値が nilの場合、C-x C-cは Emacs により開始されたサブプロセスを kill する前に確認を求めません。デフォルトでは、この値は tです。

Emacs を exit するとき何が起こるかさらにカスタマイズするには、Section “Killing Emacs” in *The GNU Emacs Lisp Reference Manual* を参照してください。

保存の問い合わせを行わずに Emacs を kill するときは、M-x kill-emacsとタイプします。

C-zは、コマンド suspend-frameを実行します。グラフィカルなディスプレーでは、このコマンドは選択された Emacs のフレームを後で戻れるように、最小化 (またはアイコン化) して隠します (どのように隠されるかはウィンドウシステムに依存します)。テキスト端末では、C-zは Emacs をサスペンド (休止) します。プログラムは一時的に停止し、制御は親プロセス (通常はシェル) に戻ります。ほとんどのシェルではシェルのコマンド%emacsで、サスペンド中の Emacs を再開できます。

テキスト端末は、通常、実行中のプログラムを kill したりサスペンドする、特定の特殊文字を監視しています。この端末の機能は、**Emacs** ではオフになっています。Emacs での C-zや C-x C-cのキーの意味は、いくつかのオペレーティングシステムでプログラムを休止させたり終了させたりするために用いる文字、C-zと C-cにヒントを得たものですが、オペレーティングシステムとの関係はそれだけです。これらのキーは、他のコマンドを実行するようにカスタマイズできます (Section 33.3.1 [Keymaps], page 519 を参照してください)。

4 基本的な編集コマンド

ここではテキストの入力、修正、ファイルへの保存といった基本操作について説明します。これらに接するのが初めてなら、learn-by-doing(行ってみることで学ぶ) 形式のチュートリアルをやってみることを提案します。チュートリアルを行うには C-h t (help-with-tutorial) とタイプしてください。

4.1 テキストの挿入

普通のグラフィック文字 *(graphic character)*(例 'a'、'B'、'3'、'=') は、対応するキーをタイプして挿入することができます。これによりバッファーのポイント位置に文字が追加されます。挿入によりポイントは前方に移動するので、ポイントは挿入された文字の直後になります。Section 1.1 [Point], page 7 を参照してください。

行を終了して新しい行を開始するには RET(newline) を入力します (キーボードで RETキーは、Returnや Enter、もしくは^^e2^^86^^b2 のような奇妙な左矢印のラベルがついているかもしれませんが、このマニュアルでは RETと呼ぶことにします)。このコマンドは改行文字をバッファーに挿入してから、メジャーモードに基づきインデント (Chapter 21 [Indentation], page 247 を参照してください) を行います。ポイントが行末にある場合には、新しく空行を作成してから新しい行をインデントします。もしポイントが行の途中にある場合、行はその位置で分割されます。自動インデントをオフにするには、Electric Indent モード (Section 21.4 [Indent Convenience], page 249 を参照してください) を無効にするか、自動インデントを行わず改行だけを挿入する C-jを入力します。

マニュアルの後ろで説明しますが、マイナーモード *(minor modes)* を利用することにより、Emacs が挿入を処理する方法を変更できます。たとえば Auto Fill モードというマイナーモードは行が長くなりすぎたとき自動的に行を分割します (Section 22.6 [Filling], page 256 を参照)。Overwrite mode というマイナーモードは、既存の文字を右方に押しやるかわりに、既存の文字を置き換え (上書き) ます。Section 20.2 [Minor Modes], page 241 を参照してください。

対応するキーを押して挿入できるのはグラフィック文字だけです。他のキーは編集コマンドとして動作し、文字自体の挿入はしません。たとえば、デフォルトでは DELは、コマンド delete-backward-charを実行します (違うコマンドにバインドされているモードもあります)。このキーはリテラルの 'DEL'(ASCIIの文字コード 127) を入力する訳ではありません。

非グラフィック文字や、キーボードがサポートしていない文字を挿入するには、最初に C-q (quoted-insert) で文字をクォート *(quote)* します。C-qの使い方は 2 つあります:

- C-qに続けて非グラフィック文字 (C-gでさえも) をタイプすると、その文字が挿入されます。たとえば C-q DELは、リテラルの 'DEL'文字を挿入します。
- C-qに続けて 8 進文字のシーケンスを入力すると、8 進の文字コードに対応する文字が挿入されます。任意の 8 進数字を使うことができます。非 8 進数字により入力は終了します。もし終了文字が RETの場合、RETは入力の終了だけに用いられます。他の非 8 進文字は入力を終了させてから、通常の入力として扱われます。つまり C-q 1 0 1 Bは 'AB'を挿入します。

  8 進数字での入力は、通常の非バイナリーの Overwrite モードでは無効になっています。それにより上書きすることなく数字を挿入する便利な方法が提供されます。

8 進のかわりに 10 進や 16 進を使うには、変数 read-quoted-char-radixに、10 や 16 をセットします。もし基数が 16 の場合、aから fは文字コードの一部として扱われます。大文字小文字は区別されません。

数は多くありませんが、一般的な Unicode 文字は C-x 8 で始まるコマンドを通じて挿入できます。たとえば C-x 8 [は『‘ 』を挿入します。これは Unicode コードポイント U+2018 LEFT SINGLE QUOTATION MARK (単独の左 "curved quote"、または "curly quote" と呼ばれることもあり) を

挿入します。同様に C-x 8] は『 ’』、C-x 8 {は『“ 』、C-x 8 }は『 ”』を挿入します。また Alt キーも (後に RETが続かない) C-x 8と同じように機能します。たとえば A-[は C-x 8 [と同様に『‘ 』を挿入します。どの文字が C-x 8による短縮入力をもつかを確認するには C-x 8 C-hとタイプしてください。

かわりにコマンド C-x 8 RET (insert-char) を使うこともできます。これはミニバッファーを使って、Unicode 名かコードポイント (code-point) の入力を求めます。もし名前を入力する時、コマンドが補完機能を提供します (Section 5.4 [Completion], page 31 を参照してください)。コードポイントを入力する場合、それは 16 進 (Unicode の規約による)、または指定した基数の数字 (例 #o23072 (octal); Section “Integer Basics” in *The Emacs Lisp Reference Manual* を参照してください) であるべきです。このコマンドは対応する文字をバッファーに挿入します。

たとえば以下はすべて同じ文字を挿入します:

```
C-x 8 RET left single quotation mark RET
C-x 8 RET left sin TAB RET
C-x 8 RET 2018 RET
C-x 8 [
A-[   (Alt キーが機能する場合)
`     (Electric Quote モードの場合)
```

C-qまたは C-x 8 ...への数引数は、文字のコピーを何個挿入するかを指定します (Section 4.10 [Arguments], page 25 を参照してください)。

C-x 8のかわりに、C-u C-x \ iso-transl RETとタイプして対応する一時入力メソッド (transient input method) を選択することを選択できます。その場合には一時入力メソッドが一時的にアクティブになり、C-x \ [とタイプすることによって同じ文字‘ が入力されます ([transient input method], page 223 を参照)。

さらに加えて、あるコンテキストにおいては、`like this'のようにクォートに grave accent と apostrophe を使用した場合は、たとえ C-x 8コマンドを使用していなくても、これは 1 つのクォーテーションマークを使用した形式``like this''のように変換されます。同様に、‘‘like this’’ のように 2 重の grave accent と apostrophe を使用した場合、これはダブルクォーテーションマークを使用した形式“ like this ”のように変換されます。Section 22.5 [Quotation Marks], page 255 を参照してください。

4.2 ポイント位置の変更

文字の挿入以上のことを行うには、ポイントを移動する方法について知る必要があります (Section 1.1 [Point], page 7 を参照してください)。キーボードのコマンド C-f、C-b、C-n、C-pは、それぞれ右・左・下・上にポイントを移動します。ほとんどのキーボードにある矢印キー— RIGHT、LEFT、DOWN、UPでもポイントを移動できます。しかし多くの Emacs ユーザーは矢印キーより、コントロールキーのほうが速いと考えています。なぜなら矢印キーを押すためにそれらが配置されている領域に手を動かす必要があるからです。

ポイントを移動したい場所でマウスの左ボタンをクリックしてもポイントを移動できます。Emacs は、さらに洗練された方法でポイントを移動する、さまざまなキーボードコマンドを提供します。

C-f　1 文字前方 (forward) に移動します (forward-char)。

RIGHT　このコマンド (right-char) は C-fと同様に振る舞います。例外はポイントのあるパラグラフが right-to-left の場合です。Section 19.20 [Bidirectional Editing], page 238 を参照してください。

C-b 1 文字後方 (backward) に移動します (backward-char)。

LEFT このコマンド (left-char) は C-bと同様に振る舞います。例外は現在のパラグラフが right-to-left の場合です。Section 19.20 [Bidirectional Editing], page 238 を参照してください。

C-n
DOWN スクリーンに表示された行で 1 行下に移動します (next-line)。このコマンドは横方向の位置を変更しないよう試みます。そのため行の途中でコマンドを開始すると、次の行の途中に移動することになります。

C-p
UP スクリーンに表示された行で 1 行上に移動します (previous-line)。このコマンドは C-nと同様、行内の位置を保ちます。

C-a
Home 行の先頭に移動します (move-beginning-of-line)。

C-e
End 行の最後に移動します (move-end-of-line)。

M-f 1 単語前方に移動します (forward-word)。Section 22.1 [Words], page 251 を最終してください。

C-RIGHT
M-RIGHT このコマンド (right-word) は M-fと同様に振る舞います。例外は現在のパラグラフが right-to-left の場合、1 語後方に移動することになります。Section 19.20 [Bidirectional Editing], page 238 を参照してください。

M-b 1 単語後方に移動します (backward-word)。Section 22.1 [Words], page 251 を参照してください。

C-LEFT
M-LEFT このコマンド (left-word) は M-bと同様に振る舞います。例外は現在のパラグラフが right-to-left の場合、1 語前方に移動することになります。Section 19.20 [Bidirectional Editing], page 238 を参照してください。

M-r スクリーン上のテキストを移動させることなく、ポイントの位置をウィンドウ上で中央にもっとも近いテキスト行の左端に移動します。連続して呼び出すと、最上行の左端、最下行の左端へと循環的にポイントを移動します (move-to-window-line-top-bottom)。

数引数はスクリーンの行の何行目にポイントを移動するか指定します。数値はウィンドウの最上行から数えた行数です (0 は最上行を意味します)。負の引数は最下行から数えた行数です (-1 は最下行を意味します。数引数については詳細は、Section 4.10 [Arguments], page 25 を参照してください。

M-< バッファーの先頭に移動します (beginning-of-buffer)。数引数 nが与えられた場合、最上行から $n/10$ に移動します。グラフィカルなディスプレイでは、C-HOMEで同じことを行うことができます。

M-> バッファーの最後に移動します (end-of-buffer)。グラフィカルなディスプレイでは、C-ENDで同じことを行うことができます。

C-v
PageDown
next　画面を 1 画面前方にスクロールします。もし必要ならポイントをスクリーン上の位置に移動します (scroll-up-command)。Section 11.1 [Scrolling], page 77 を参照してください。

M-v
PageUp
prior　画面を 1 画面後方にスクロールします。もし必要ならポイントをスクリーン上の位置に移動します (scroll-down-command)。Section 11.1 [Scrolling], page 77 を参照してください。

M-g c　数値 *n*を読み、ポイントをバッファー位置 *n*に移動します。位置に 1 を指定するとバッファーの先頭に移動します。もしポイントがバッファーの数字の上または直後にある場合には、その数が *n*のデフォルトになります。そこで単にミニバッファーで RETを押すと、その数が使われます。数プレフィックス引数を M-g cに与えて *n*を指定することもできます。

M-g M-g
M-g g　数値 *n*を読み、ポイントをバッファーの先頭から *n*行目に移動します。行に 1 を指定するとバッファーの先頭に移動します。もしポイントがバッファーの数字の上または直後にある場合、その数が *n*のデフォルトになります。ミニバッファーで単に RETを押すと、その数が使われます。数値のプレフィクス引数で *n*を指定して M-g M-gに与えることもできます。単にプレフィクス引数を与えた場合の M-g M-gの動作については、Section 16.1 [Select Buffer], page 176 を参照してください。ナローされたバッファーのアクセス可能範囲にたいして相対的な行にポイントを移動するためには、かわりにコマンド goto-line-relativeを使用できます。

goto-lineは独自のヒストリーリストをもちます (Section 5.5 [Minibuffer History], page 36 を参照)。ユーザーオプション goto-line-history-localをカスタマイズすれば、すべてのバッファー間で共有される単一リスト (デフォルト)、あるいはバッファーそれぞれにたいして個別にリストをもつことができます。

M-g TAB　数値 *n*を読み取り、現在行の *n*列目に移動します。列 0 は最左列です。プレフィクス引数とともに呼び出された場合、引数で指定された数の列に移動します。

C-x C-n　カレントバッファー内の現在ポイントがある列を、半恒久的な目標列 *(semipermanent goal column)* として使用します。目標列が有効な場合にコマンド C-n、C-p、<prior>や<next>で垂直に移動すると、その列もしくはできる限り近い列に移動しようと試みます。目標列はキャンセルされるまで有効です。

C-u C-x C-n
目標列をキャンセルします。それ以降の C-nや C-pは通常どおり水平位置を保とうと試みます。

バッファーのテキストがウィンドウの幅より長い場合、通常 Emacs は 2 行以上のスクリーン行 *(screen lines)* で表示します。便宜上、C-nと C-pそして downと upも、同様にスクリーン行にしたがってポイントを移動します。これらのコマンドを論理行 *(logical lines)*(たとえばバッファーのテキスト行) にしたがって移動させるには、line-move-visualに nilをセットします。そうすると論理行が複数のスクリーン行となるような場合、カーソルは追加されたスクリーン行をスキップします。詳細は Section 4.8 [Continuation Lines], page 23 を参照してください。line-move-visualなどの変数をセットする方法については、Section 33.2 [Variables], page 508 を参照してください。

C-nや C-pと異なり、ほとんどの Emacs コマンドは論理的な行に作用します。たとえば C-a (move-beginning-of-line) や C-e (move-end-of-line) は、論理行の先頭もしくは最後に移動します。C-nや C-pのようにスクリーン行に作用するコマンドの場合、わたしたちはそれを示すようにします。

line-move-visualが nilの場合、変数 track-eolにも非 nil値をセットできます。そうすると論理行の行末で C-nや C-pを開始すると、次の論理行の行末に移動します。通常 track-eolは nilです。

通常 C-nをバッファーの最後の行で使用した場合、バッファーの最後でストップします。しかし変数 next-line-add-newlinesに非 nil値をセットした場合、バッファーの最後の行で C-nを押すと、行を追加してその行に移動します。

4.3 テキストの消去

DEL
BACKSPACE
: ポイントの前の文字、またはリージョンがアクティブのときはリージョンを削除します (delete-backward-char)。

Delete
: ポイントの後の文字、またはリージョンがアクティブのときはリージョンを削除します (delete-forward-char)。

C-d
: ポイントの後ろの文字を削除します (delete-char)。

C-k
: 行末まで kill します (kill-line)。

M-d
: 次の単語 (word) の末尾までを前方に kill します (kill-word)。

M-DEL
M-BACKSPACE
: 前の単語の先頭までを後方に kill します (backward-kill-word)。

コマンド DEL (delete-backward-char) は、ポイントの前の文字を削除して、カーソルと後ろの文字を後方に移動します。ポイントが行の先頭にある場合、前の改行を削除して、その行を前の行と連結します。

しかしリージョンがアクティブのとき、DELはリージョンのテキストを削除します。リージョンの説明は、Chapter 8 [Mark], page 52 を参照してください。

ほとんどのキーボードでは、DELには BACKSPACEというラベルがついていますが、このマニュアルでは DELと呼ぶことにします (DELを Deleteと混同しないでください。Deleteについてはこの後で議論します)。いくつかのテキスト端末では、Emacs は DELを正しく認識しません。もしこの問題に遭遇したときには、Section 34.2.8 [DEL Does Not Delete], page 541 を参照してください。

コマンド Delete (delete-forward-char) は、反対方向に削除します。これはポイントの後ろの文字、たとえばカーソルの下の文字を削除します。ポイントが行末にある場合は、その行を次の行と連結します。DELと同様、リージョンがアクティブのときはリージョンのテキストを削除します (Chapter 8 [Mark], page 52 を参照してください)。

C-d (delete-char) は、Deleteと同じようにポイントの後ろの文字を削除しますが、リージョンがアクティブかどうかは関係ありません。

上述した削除コマンドについての詳細な情報は、Section 9.1.1 [Deletion], page 59 を参照してください。

C-k (kill-line) は行を一度に消去 (kill) します。もし行頭または行の途中で C-kとタイプすると、行末までのすべてのテキストを kill します。行末で C-kとタイプすると、その行を次の行と連結します。

C-kと関連するコマンドについては、Chapter 9 [Killing], page 59 を参照してください。

4.4 変更のアンドゥ

C-/
C-x u
C-_ undo レコードにあるエントリーを undo します。通常 1 つのコマンドを元に戻す (undo) ことに相当します (1 つ目のキーはテキストモードのディスプレイでは利用できないかもしれない)。

Emacs はバッファー内のテキストに行われた変更のリストを記録しているので、最近の変更は undo できます。これは C-/(および C-x uと C-_) にバインドされているコマンド undoを使って行われます。通常このコマンドは最後の変更を undo して、ポイントを変更前の位置に移動します。undo コマンドはバッファーへの変更のみに適用されるので、カーソルの動きを undo することはできません。

Control以外のすべてのキーにたいする Control修飾をサポートする端末では、Shift修飾が不要なので C-/が undoを呼び出すもっとも簡単な方法です。ASCII コントロール文字だけを許容する端末には C-/が存在しませんが、それらの多くでは実際には C-_を Emacs に送信するので、C-/が依然として機能します。それ以外の場合も多くは C-_をタイプ (実際には C--を押下) する際に Shift 修飾を省略できるので、これが undoを呼び出すもっとも簡単な方法になります。

個々の編集コマンドは、通常 undo レコードの個別のエントリーとなりますが、とても単純なコマンドはグループ化される場合があります。1 つのエントリーが、実は複雑なコマンドのほんの一部の場合もあります。

もし C-/(またはそれの別名コマンド) を繰り返すと、undo された箇所はさらに undo され、初期の変更も undo され、ついには利用可能な undo 情報の限界に達します。もし記録された変更がすべて undo されている場合、undo コマンドはエラーメッセージを表示して、何も行いません。

undoコマンドについてさらに学ぶには、Section 13.1 [Undo], page 131 を参照してください。

4.5 ファイル

Emacs のバッファーに挿入したテキストは、Emacs のセッションの間だけ存在します。テキストを永続化させるためには、それをファイル *(file)* に保存しなければなりません。

ホームディレクトリーに、test.emacsという名前のファイルがあるとしましょう。このファイルを Emacs で編集するには、以下を入力します

```
C-x C-f test.emacs RET
```

ここでファイル名は、コマンド C-x C-f (find-file) に与えられる、引数 *(argument)* です。このコマンドは引数を読み取るためにミニバッファー *(minibuffer)* を使い、RETは引数を終端させます (Chapter 5 [Minibuffer], page 28 を参照してください)。

このコマンドに従うために、Emacs はそのファイルを *visit(*訪問*)* します: すなわちバッファーを作成し、ファイル内容をバッファーにコピーし、編集のためにバッファーを表示します。テキストを変更したら、C-x C-s (save-buffer) と入力することにより、ファイルを保存 *(save)* できます。これにより変更されたバッファー内容は、test.emacsに書き戻され永続化されます。保存するまでは、テキストへの変更は Emacs 内部にだけ存在し、ファイル test.emacsは変更されません。

ファイルを作成するには、すでにファイルが存在するかのように C-x C-fでファイルを visit するだけです。これはファイルに書き込みたいテキストを入力できる、空のバッファーを作成します。最初にこのバッファーを C-x C-sで保存するとき、Emacs は実際にファイルを作成します。

Emacs でファイルを使うことについてさらに学ぶには、Chapter 15 [Files], page 145 を参照してください。

4.6 ヘルプ

もしキーが何をするか忘れた場合、C-h k (describe-key) と入力して、それに続けて関心のあるキーを入力します。たとえば C-h k C-nは、C-nが何をするか表示します。

プレフィクスキー C-hは “ヘルプ (help)” が由来です。F1キーは C-hの別名です。C-h k以外にも、異なる種類のヘルプを提供する多くのヘルプコマンドがあります。

詳細については、Chapter 7 [Help], page 42 を参照してください。

4.7 空行

空行を挿入したり削除するための、特別なコマンドとテクニックがあります。

C-o カーソルの後ろに空行を挿入します (open-line)。

C-x C-o 連続する空行を、1 行残してすべて削除します (delete-blank-lines)。

これまで RET (newline) が、どうやってテキストの新しい行を開始するのか見てきました。しかし最初に空行を作ってからテキストを挿入するほうが、何を行っているのかわかりやすいでしょう。これはキー C-o (open-line) を使えば、簡単に行うことができます。これはポイントの後ろに改行を挿入し、ポイントを改行の前に維持します。C-oの後に新しい行のためのテキストを入力します。

複数の空行は C-oを数回入力するか、何個の空行を作るのかを数引数で与えれば作れます。方法については、Section 4.10 [Arguments], page 25 を参照してください。もしフィルプレフィクスがあって、行頭で C-oが入力された場合、新しい行にフィルプレフィクスを挿入します。Section 22.6.3 [Fill Prefix], page 258 を参照してください。

余分な空行を取り除く簡単な方法は、C-x C-o (delete-blank-lines) です。連続する空行の中にポイントがあるとき、C-x C-oは 1 行残してすべての空行を削除します。ポイントが単独の空行にある場合、C-x C-oはその空行を削除します。ポイントが空でない行にある場合、C-x C-oは、後続する空行があれば、それらすべてを削除します。

4.8 継続行

バッファー内のテキストの行 — 論理行 *(logical line)* — がウィンドウに収まらないほど長い場合、Emacs がそれを 2 行以上のスクリーン行 *(screen lines)* で表示するときがあります。これは行の折り返し *(line wrapping)* または継続 *(continuation)* と呼ばれ、論理行は継続された行 *(continued line)* と呼ばれます。グラフィカルなディスプレイでは、Emacs は行の折り返しをウィンドウの左右のフリンジ (fringes、縁) の小さな曲矢印で示します。テキスト端末では、Emacs は右の余白に ‘\’を表示して行の折り返しを示します。

ほとんどのコマンドは、スクリーン行ではなく論理行にたいして作用します。たとえば C-kは論理行を kill します。前に説明したように、C-n (next-line) と C-p (previous-line) は特別な例外です。これらはスクリーン行にたいしてポイントを上下に移動させます (Section 4.2 [Moving Point], page 18 を参照してください)。

Emacs はオプションで長い論理行を継続するかわりに、切り詰める *(truncate)* ことができます。これは論理行が 1 つのスクリーン行を占めることを意味します。もし論理行がウィンドウ幅より長い場合、行の残りは表示されません。グラフィカルなディスプレイでは切り詰められた行は、右フリンジの小さな直矢印で示されます。テキスト端末では右余白の '`$`'で示されます。Section 11.22 [Line Truncation], page 100 を参照してください。

デフォルトでは継続行はウィンドウの右端で折り返されます。折り返しが単語の途中で発生すると、継続された行は読むのが難しくなります。普通の解決策は、行が長くなりすぎる前に改行を挿入することです。もしお好みなら、行が長くなりすぎたときに Emacs が自動的に改行を挿入するように、Auto Fill モードを使うことができます。Section 22.6 [Filling], page 256 を参照してください。

多くの長い論理行を含むファイルを編集する必要があり、それらすべてを改行で分割するのが実用的でない場合があります。そのようなケースでは単語折り返し *(word wrapping)* が有効な Visual Line モードを使うことができます。これは長い行を正確にウィンドウの右端で折り返すのではなく、ウィンドウの右端に一番近い単語境界 (スペースやタブなど) で折り返します。Visual Line モードでは、`C-a`、`C-n`、`C-k`などの編集コマンドも、論理行ではなくスクリーン行を処理するように再定義されます。Section 11.23 [Visual Line Mode], page 101 を参照してください。

4.9 カーソル位置の情報

バッファーのある部分にたいしてサイズや位置、単語数や行数についての情報を得るためのコマンドがあります。

`M-x what-line`
: ポイントの行番号を表示します。

`M-x line-number-mode`
`M-x column-number-mode`
: 現在の行番号および列番号の自動表示を切り替えます。Section 11.19 [Optional Mode Line], page 97 を参照してください。各行の前に行番号を表示したい場合は、Section 11.24 [Display Custom], page 102 を参照してください。

`M-=`
: 現在のリージョンの行 (line)、センテンス (sentence)、単語 (word)、文字 (character) の個数を表示します (`count-words-region`)。リージョンについては Chapter 8 [Mark], page 52 を参照してください。

`M-x count-words`
: 現在のバッファーの行、センテンス、単語、文字の個数を表示します。リージョン (Chapter 8 [Mark], page 52 を参照) がアクティブのときは、かわりにリージョンの数字を表示します。

`C-x =`
: ポイントの後ろの文字の文字コード、ポイントの文字位置、ポイントの列位置を表示します (`what-cursor-position`)。

`M-x hl-line-mode`
: 現在行のハイライト表示を有効または無効にします。Section 11.21 [Cursor Display], page 99 を参照してください。

`M-x size-indication-mode`
: バッファーのサイズの自動表示を切り替えます。Section 11.19 [Optional Mode Line], page 97 を参照してください。

`M-x what-line`は、エコーエリアに現在の行番号を表示します。通常このコマンドは不必要です。なぜならモードラインに現在の行番号が、すでに表示されているからです (Section 1.3 [Mode Line],

page 9 を参照してください)。しかしバッファーがナロー (narrow: 制限) されている場合、モードラインはアクセスできる範囲についての行番号しか表示しません (Section 11.5 [Narrowing], page 81 を参照してください)。それにくらべて `what-line`は、制限されたリージョンとバッファー全体、両方の行番号を表示します。

`M-=` (`count-words-region`) はリージョン内の行、センテンス、単語、文字の個数を報告するメッセージを表示します (リージョンについての説明は Chapter 8 [Mark], page 52 を参照)。プレフィクス引数 `C-u M-=`を指定すると、このコマンドはバッファー全体の数字を表示します。

`M-x count-words`は同じことを行いますが、呼び出し規約が異なります。もしリージョンがアクティブの場合はリージョン、そうでない場合はバッファーの数字を表示します。

コマンド `C-x =` (`what-cursor-position`) は現在のカーソル位置と、その位置にあるバッファー内容についての情報を表示します。エコーエリアには、以下のような行が表示されます:

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

‘`Char:`’には、バッファー中のそのポイントにある文字が表示されます。カッコ内にはその文字に対応する文字コードが 10 進、8 進、16 進で表示されます。`C-x =`が文字の情報について表示する方法については、Section 19.1 [International Chars], page 216 を参照してください。‘`point=`’はポイント位置を文字数 (バッファーの最初の文字は 1、次の文字は 2、...) で表示します。その後ろの数字ではバッファー内の文字数の合計が表示され、カッコ内にはその位置が全体から見て何パーセントの位置なのかが表示されます。‘`column=`’にはポイントの水平位置、すなわちウィンドウの左端から数えて何番目の列かが表示されます。

ユーザーオプション `what-cursor-show-names`が非 `nil`なら Unicode 文字データベース (Unicode Character Database) で定義されておいるような名前が同様に表示されます。カッコ内の部分は以下のようになります:

```
(99, #o143, #x63, LATIN SMALL LETTER C)
```

もしバッファーがナローされている場合、最初と最後の部分のテキストが一時的にアクセス不能になります。`C-x =`は現在アクセス可能な範囲についての追加説明を表示します。たとえば以下のように表示します:

```
Char: C (67, #o103, #x43) point=252 of 889 (28%) <231-599> column=0
```

ここで、新たに追加された 2 つの数字が、ポイントを設定できる文字位置の下限と上限を示します。これら 2 つの位置のあいだの文字が参照可能な文字です。Section 11.5 [Narrowing], page 81 を参照してください。

関連はあるものの異なる機能が `display-line-numbers-mode`(Section 11.24 [Display Custom], page 102 を参照) です。

4.10 数引数

数学や計算機の用語では、引数 (argument) という単語は、“関数や操作に与えるデータ” を意味します。Emacs のコマンドには、数引数 (numeric argument)(プレフィクス引数 (prefix argument) とも呼ぶ) を指定できるものがあります。引数を反復回数として解釈するコマンドもあります。たとえば、引数 10 を `C-f`に指定すると、カーソルを通常の 1 文字ではなく、10 文字分前向きに移動します。これらのコマンドでは、引数を指定しないと引数 1 を指定したのと同等になります。この種のコマンドの多くでは、負の引数を指定すると、逆向きの移動や逆の操作を指示することになります。

数引数を指定するもっとも簡単な方法は、`Meta`キーを押しながら数字またはマイナス記号 (と数字) を入力する方法です。以下はその例です:

```
M-5 C-n
```

これは5行下に移動します。キー M-1、M-2、...、同様に M--は、次のコマンドへの引数をセットアップするコマンド、(digit-argumentと negative-argument) にバインドされています。数字をともなわない M--は、通常 −1 を意味します。

2 桁以上の数字を入力したい場合、2 文字目以降の数字を入力するときに Metaを押しつづける必要はありません。つまり 50 行下に移動するときは、以下のように入力します:

```
M-5 0 C-n
```

これは、(あなたが期待するように)‘0’を 5 つコピーして挿入してから 1 行下がるのではないことに注意してください。‘0’はプレフィクス引数の一部として扱われます。

(‘0’を 5 つコピーして挿入するときは、M-5 C-u 0と入力します。ここで C-uはプレフィクス引数を終端させるので、次のキー入力はあなたが実行したいコマンドです。ここでの C-uの意味はこのケースだけに適用される使い方です。C-uの通常の役割については以下を参照してください。)

数引数を指定する別の方法として、M-1、M-2、... と入力するかわりに、C-u (universal-argument) のあとに数字 (負の引数の場合はマイナス記号と数字) を入力する方法があります。通常、数字をともなわないマイナス記号は −1 を意味します。

単独の C-uは、“4 倍” という特別な意味をもち、次のコマンドの引数を 4 倍にします。C-u C-u は 16 倍です。つまり C-u C-u C-fは 16 文字前方に移動します。その他に便利な使い方としては C-u C-n、C-u C-u C-n(適当な割り合いで画面を下に移動する) や、C-u C-u C-o(16 個の空行を作る)、C-u C-k(4 行削除する)、などがあります。

自分自身を挿入する文字の前に数引数を使えば、指定した分のコピーを挿入できます。これは挿入したい文字が数字でないときは簡単です。たとえば C-u 6 4 a は、‘a’を 64 個コピーして挿入します。しかし数字を挿入したいときは、これではうまくいきません。C-u 6 4 1 は引数に 641 を指定することになってしまいます。このようなときは引数と挿入したい数字を分けるために、他の C-uを使うことができます。たとえば C-u 6 4 C-u 1 とすれば、これは ‘1’を 64 個コピーして挿入します。

引数の有無は確認しても、その値は無視するコマンドもあります。たとえばコマンド M-q (fill-paragraph) は、1 行に収まるようできるだけテキストをフィルしますが、引数をともなうと、余分なスペースを挿入してテキストが正確に 1 行の最大幅を使うよう均等に割り付けてフィルします (M-qについては、Section 22.6 [Filling], page 256 を参照してください)。このようなコマンドは、引数として単に C-uを指定するだけで充分です。

引数の値を繰り返しの回数として使いますが、引数がないときは特別な処理を行うコマンドもあります。たとえばコマンド C-k (kill-line) に引数 *n*を指定すると、これは行末の改行も含めて *n*行を kill します。しかし引数を指定しないで C-kした場合、ポイントから改行までのテキストを kill するか、ポイントが行末にある場合は改行を kill します。つまりコマンド C-kを引数なしで 2 回呼び出すと、C-kに引数 1 を指定したのと同様、空でない行を kill できます (C-kについての情報は、Chapter 9 [Killing], page 59 を参照してください)。

いくつかのコマンドは、C-uだけの引数を通常の引数とは異なるものとして扱います。また、マイナス記号のみの引数を、−1 とは区別するコマンドもあります。これらの例外については、必要になったときに説明します。これらの例外は、それぞれのコマンドを使いやすくするためにあり、コマンドのドキュメント文字列に記載されています。

コマンドの前に引数を入力するという点を強調するために、そしてコマンドが呼び出された後に入力されるミニバッファー引数 (Chapter 5 [Minibuffer], page 28 を参照してください) と区別するために、わたしたちはプレフィクス引数 *(prefix argument)* という言葉を使います。

グラフィカルなディスプレイでは C-0、C-1、... は M-0、M-1、... と同じように振る舞います。

4.11 コマンドの繰り返し

単純なキーで呼び出されるものや、M-x *command-name* RET で実行できるような多くのコマンドは、数引数で繰り返し回数 (Section 4.10 [Arguments], page 25 を参照してください) を与えることで、その回数だけ繰り返すことができます。しかし、入力を求めるものや数引数を別の目的に使うコマンドでは、この方法はうまくいきません。

コマンド C-x z (repeat) は、Emacs コマンドを何回も反復する別の方法です。このコマンドは、直前の Emacs コマンドが何であっても、それを繰り返します。繰り返されるコマンドは、まえと同じ引数を使います。毎回新たに引数を読み取ることはしません。

コマンドを 2 回以上繰り返すには zを追加して入力します。1 つの zでコマンドを 1 回繰り返します。z以外の文字を入力するか、マウスボタンを押すと繰り返しを終了します。

たとえば、20 文字削除するために C-u ２０ C-dと入力したとしましょう。C-x z z zと入力すれば、(引数を含めて) 削除コマンドをさらに 3 回繰り返し、全部で 80 文字削除できます。始めの C-x zでコマンドを 1 回繰り返し、そのあとのそれぞれの zで 1 回ずつ繰り返します。

1 つの文字をタイプすることによって、2 つ以上のキーシーケンスにバインドされたコマンドを繰り返し実行できる repeat-modeモードをアクティブにすることもできます。たとえばもっとも最近行った編集をアンドゥするために C-x u (undoのこと; Section 13.1 [Undo], page 131 を参照) をタイプした後に、u u u...とタイプすることで更に多くの編集をアンドゥすることができます。同様に C-x o C-x o C-x o...のかわりに C-x o o o...とタイプすれば、いくつかのウィンドウを超えて切り替えることができます。これはそのコマンドを呼び出す完全なキーシーケンスをタイプした後に、一時的な繰り返しモードにエンターすることによって機能します。単一キーによるショートカットはエコーエリアに表示されます。

repeat-modeでの繰り返しをサポートしているのは一部のコマンドに限られます。サポートしているコマンドを確認するには、M-x describe-repeat-maps RET とタイプしてください。

一時的な繰り返しモードによって有効になる単一文字のショートカットは同じ文字である必要はありません。たとえば C-x {とタイプした後であれば{、}、^、v、あるいはこれらの文字を任意の順で混ぜ合わせてタイプすると、それに応じて選択されたウィンドウがリサイズされるでしょう。同様に*compilation*や*grep*バッファーでは M-g n や M-g pの後に任意の順に混ぜ合わせて nや pをタイプすることで、next-errorや previous-errorでバッファーを移動することができます (Section 24.2 [Compilation Mode], page 310 を参照)。

前のコマンドを繰り返すよう定義された文字以外の文字をタイプすると一時的な繰り返しモードを抜けて、タイプした文字は通常通り実行されます。キーを実行せずに、一時的な繰り返しモードを抜けるだけのキーを定義することもできます。これを行うには、ユーザーオプション repeat-exit-key にキーの名前を指定します。自然な値としては RETなどが考えられます。最後に何秒かのアイドル時間後に自動的に繰り返しの連鎖を中断させることも可能です。ユーザーオプション repeat-exit-timeoutをカスタマイズして、一時的な繰り返しモードを自動的にオフに切り替えるアイドル時間の秒数を指定してください。

5 ミニバッファー

ミニバッファー *(minibuffer)* とは、Emacs のコマンドがファイル名、バッファー名、Emacs コマンド名、Lisp 式といった、複雑な引数を読み取るための場所です。なぜ “ミニバッファー” と呼ぶかというと、それがスクリーン上の小領域を占める、特別な目的のためのバッファーだからです。ミニバッファーで引数テキストを編集するために、通常の Emacs 編集コマンドを使うことができます。

5.1 ミニバッファーを使う

ミニバッファーを使用中、ミニバッファーはエコーエリアにカーソルとともに表示されます。ミニバッファーは通常、コロンが最後についたプロンプト *(prompt)* から開始されます。プロンプトはどのような入力が期待されるか、そしてそれがどのように使われるのかを示します。プロンプトは、フェイス `minibuffer-prompt`を使ってハイライトされます。

ミニバッファーで入力するもっとも簡単な方法は、テキストを入力してから `RET`で引数入力を完了してミニバッファーを終了する方法です。かわりに `C-g`を入力して引数を求めているコマンドをキャンセルし、ミニバッファーを終了することもできます (Section 34.1 [Quitting], page 536 を参照してください)。

コロンの前のカッコ内にデフォルト引数 *(default argument)* を表示するプロンプトもあります。このデフォルト値は、`RET`だけを入力したときに引数として使用されます。たとえばバッファー名を読み取るコマンドは、通常はデフォルト値としてバッファー名を表示します。`RET`を入力することでデフォルトのバッファーにたいして処理を行うことができます。ユーザーオプション `minibuffer-default-prompt-format`でデフォルト引数の表示方法をカスタマイズできます。

Minibuffer Electric Default モードというグローバルマイナーモードを有効にしている場合には、ミニバッファーの内容の変更を開始すると、Emacs がデフォルト引数を非表示にします。ミニバッファーのテキストを元に戻せば、ふたたびプロンプトにデフォルト値が表示されます。このマイナーモードを有効にするには、`M-x minibuffer-electric-default-mode`とタイプしてください。

エコーエリアにミニバッファーが表示されると、他のエコーエリアの使用と競合するかもしれません。ミニバッファーがアクティブなときは、メッセージはミニバッファーのテキストの後にカッコ内に数秒、あるいは何かをタイプするまで表示されて、その後消えます。ミニバッファーの使用中、Emacs はキーストロークをエコーしません。

ミニバッファー使用中に、たとえば入力を要するテキストをメモするために別フレームに切り替えることができます (Section 18.7 [Frame Commands], page 200 を参照)。デフォルトではアクティブミニバッファーはその新たなフレームへ移動します。ユーザーオプション `minibuffer-follows-selected-frame`を `nil`にセットした場合には、ミニバッファーはそれをオープンしたフレームに留まり、カレントコマンドを完了 (または abort) するためにそのフレームに切り替えて戻らなければなりません。このオプションに `nil`や `t`以外の値をセットすると、ミニバッファーはカレントコマンドでオープンされた再帰的ミニバッファーの後でのみ移動します (Section “Recursive Mini” in `elisp`を参照)。このオプションは主に Emacs 28.1 以前の挙動を (おおよそ) 維持するためです。最終的にミニバッファーの使用を終えた際には、そのコマンドの効果は常にミニバッファーをオープンした最初のフレームで発生することに注意してください。唯一の例外はそのフレームがもはや存在しない場合で、そのときは選択されたフレームにたいして処理を行います。

5.2 ミニバッファーでのファイル名

`C-x C-f` (`find-file`) のようなコマンドは、ミニバッファーを使ってファイル名引数を読み取ります。ファイル名を読み取るためにミニバッファーを使用しているとき、通常は最後にスラッシュがつ

いたテキストで開始されています。これはデフォルトディレクトリー *(default directory)* です。たとえば以下のように開始されていたとします:

```
Find file: /u2/emacs/src/
```

ここで 'Find file: 'はプロンプト、'/u2/emacs/src/'はデフォルトディレクトリーです。ここで buffer.cを入力すると/u2/emacs/src/buffer.cを指定したことになります。デフォルトディレクトリーについての情報は、Section 15.1 [File Names], page 145 を参照してください。

あなたが望むかもしれないファイル名のデフォルト候補は、M-nとタイプすることにより利用できます。Section 5.5 [Minibuffer History], page 36 を参照してください。

..で親ディレクトリー内のファイルを指定できます。つまり/a/b/../foo.elは/a/foo.elと同じです。M-DELを使えば、ディレクトリー名を後方に kill できます (Section 22.1 [Words], page 251 を参照してください)。

デフォルトディレクトリーとは無関係のファイルを指定する場合、デフォルト値全部を C-a C-kで kill できます。かわりにデフォルト値を無視することもできます。これはスラッシュで始まる絶対パスのファイル名か、チルダで始まるファイル名をデフォルトディレクトリーに続けて入力します。たとえば以下のようにして/etc/termcapを指定できます:

```
Find file: /u2/emacs/src//etc/termcap
```

ダブルスラッシュにより、Emacs は 2 番目のスラッシュより前のすべてを無視します。上の例では/u2/emacs/src/は無視されるので、引数は/etc/termcapとなります。無視される部分のファイル名は、端末に可能なら目立たないような表示になります (これを無効にするには、コマンド M-x file-name-shadow-mode で File Name Shadow モードをオフにしてください)。

リモートファイルの名前 (Section 15.15 [Remote Files], page 169 を参照) を補完する際、ダブルスラッシュは若干異なる挙動を示します。この場合のダブルスラッシュは、Emacs がファイル名の部分だけを無視して、他の部分 (method、host、username、... など) を手付かずのままにするようにします。3 つのスラッシュを連続してタイプすると、リモートファイル名の中のすべてを無視します。Section "File name completion" in *The Tramp Manual* を参照してください。

Emacs は~/をホームディレクトリーと解釈します。~/foo/bar.txtはホームディレクトリーにある、fooというディレクトリーの、bar.txtという名前のファイルを指定します。さらに~*user-id*/はログイン名が *user-id* というユーザーの、ホームディレクトリーを意味します。~の前のディレクトリー名は無視されるので、/u2/emacs/~/foo/bar.txtは~/foo/bar.txtと同じです。

MS-Windows と MS-DOS では、ユーザーは常にホームディレクトリーを持つとは限らないので、Emacs はいくつかの代替ディレクトリーを使います。MS-Windows については Section H.5 [Windows HOME], page 612、MS-DOS については Section "MS-DOS File Names" in *the digital version of the Emacs Manual* を参照してください。これらのシステムでは~*user-id*/は現在のユーザーの場合だけ、つまり *user-id* が現在のユーザーのログイン名のときだけがサポートされます。

Emacs がファイル名を読みとるとき、デフォルトディレクトリーを挿入しないようにするには、変数 insert-default-directoryを nilに変更します。この場合、ミニバッファーは空で開始されます。それでも相対パスでのファイル名引数は、同じデフォルトディレクトリーにもとづいて解釈されます。

ミニバッファーにリモートファイル名を入力することもできます。Section 15.15 [Remote Files], page 169 を参照してください。

5.3 ミニバッファーでの編集

ミニバッファーは一風変わっていますが Emacs のバッファーなので、引数テキストを編集するための、通常の Emacs コマンドが利用可能です (しかしプロンプトは読み取り専用 *(read-only)* なので変更できません)。

ミニバッファーでの RETは引数を完了させるので、これを使って改行を挿入することはできません。C-q C-jを使えば制御文字 C-j(改行文字と等しい) を挿入できます (Section 4.1 [Inserting Text], page 17 を参照してください)。かわりに C-o (open-line) を使うこともできます (Section 4.7 [Blank Lines], page 23 を参照してください)。

ミニバッファーの中では TAB、SPC、?は補完コマンド *(completion commands)* にバインドされている場合があります。これによりテキスト全部を入力せずに、入力したいテキストを簡単に入力できます。Section 5.4 [Completion], page 31 を参照してください。RETのときと同様、C-qを使ってTAB、SPC、'?'のような文字を入力できます。補完ではなく通常のように SPCや?を入力したい場合には、init ファイルに以下を記述してください:

```
(keymap-unset minibuffer-local-completion-map "SPC")
(keymap-unset minibuffer-local-completion-map "?")
```

便宜上ミニバッファーでの C-a (move-beginning-of-line) は、プロンプトの先頭ではなく引数テキストの先頭にポイントを移動します。これにより、たとえば C-a C-kで引数全体を kill ことができます。

ミニバッファーがアクティブのとき、エコーエリアは通常の Emacs ウィンドウのように扱われます。たとえば (C-x oで) 他のウィンドウに切り替えて、そこでテキストを編集して、またミニバッファーのウィンドウにもどって引数の入力を完了できます。ほかのウィンドウでテキストを kill してからミニバッファーのウィンドウにもどり、引数にテキストを yank することさえ可能です。しかしミニバッファーのウィンドウは分割 (split) できないなどの制限もあります。Chapter 17 [Windows], page 186 を参照してください。

通常ミニバッファーのウィンドウは、スクリーン行で 1 行を占めます。しかし 2 行以上のテキストをミニバッファーに追加すると、そのテキストに対応して自動的に拡張されます。変数 resize-mini-windowsは、ミニバッファーのサイズ調整を制御します。デフォルト値は grow-onlyで、これは今説明したとおりの振る舞いを意味します。もし値が tの場合、ミニバッファーから行を削除すると。ミニバッファーのウィンドウは自動的に縮小されて、スクリーン行で 1 行まで小さくなります。値が nilの場合、ミニバッファーのウィンドウは自動的にサイズを変更しません。しかし通常のウィンドウのサイズ調整コマンドは使用できます (Chapter 17 [Windows], page 186 を参照してください)。

変数 max-mini-window-heightは、ミニバッファーのウィンドウのサイズ変更するときの、最大高さを制御します。浮動少数を指定した場合は、フレームの高さにたいする比になります。整数を指定した場合は最大行数になります。nilを指定すると、ミニバッファーのウィンドウの自動サイズ調整は行われません。デフォルト値は 0.25 です。

ミニバッファーでの C-M-vコマンドは、他のウィンドウに表示されたコマンドのヘルプテキストをスクロールします。M-PageUpや M-PageDown(または M-priorや M-next) でも、ヘルプテキストをスクロールできます。これは長い補完候補のリストを選ぶときなどに便利です。Section 17.3 [Other Window], page 187 を参照してください。

Emacs ではミニバッファーがアクティブのときには、通常はミニバッファーにたいして多くのコマンドが使用できないようになっています。ミニバッファーでこれらのコマンドを使えるようにするには、変数 enable-recursive-minibuffersに tをセットしてください。ミニバッファーを再帰的に使用中にミニバッファーのプロンプトにカレントの再帰深さを表示するために、minibuffer-depth-indicate-modeも有効にする必要があるかもしれません。

ミニバッファーがアクティブなときは、通常は minibuffer-modeです。これはこれは特別な機能をもたない、内部的な Emacs モードです。

アクティブでないとき、ミニバッファーは minibuffer-inactive-modeになっており、mouse-1をクリックすると、*Messages*バッファーを表示します。ミニバッファー専用のフレームを使用している場合、Emacs はそこでのキー入力も認識します。たとえば nは新しいフレームを作成します。

5.4 補完

引数を入力する助けとなる、補完 *(completion)* という機能が使えるときがあります。これは引数の一部を入力すると、それまでに何を入力したかにもとづいて、Emacs が残りあるいは残りの一部を補完してくれることを意味します。

補完が利用可能なとき、特定のキー (通常は TAB、RET、SPC) が、ミニバッファーの特別な補完コマンド (Section 5.4.2 [Completion Commands], page 31 を参照してください) にリバインドされています。これらのコマンドは、ミニバッファーのテキストを完了させようと試みます。これは引数を要求したコマンドが提供する、補完候補 *(completion alternatives)* にもとづいています。通常?を入力すると、補完候補のリストを見ることができます。

補完は通常ミニバッファー内で行われますが、通常のバッファーないでもこの機能を利用可能なときがあります。Section 23.8 [Symbol Completion], page 302 を参照してください。

5.4.1 補完の例

ここでは簡単な例が理解しやすいでしょう。M-xは、コマンド名を読み取るためにミニバッファーを使います。補完はミニバッファーのテキストと、既存の Emacs コマンドの名前のマッチによって機能します。コマンドを auto-fill-modeを実行したいとします。M-x auto-fill-mode RETをタイプすればよいのですが、補完を使えばもっと簡単になります。

M-x a u TABとタイプすると、TABは‘au’で始まる補完候補 (この例ではコマンド名) を探します。auto-fill-mode、autoconf-modeなどの候補がいくつかありますが、候補はすべて autoで始まるので、ミニバッファーの‘au’は‘auto’に補完されます (あなたの Emacs のセッションには、もっと多くのコマンドが定義されているかもしれません。たとえば authorize-meというコマンドが定義されている場合には、Emacs が補完できるのは‘aut’までです)。

もう一度 TABをタイプしても、次の文字は‘-’、‘a’、‘c’のどれなのか決定できません。そのため文字は追加されず、かわりに TABは可能性のある補完候補の一覧を別のウィンドウに表示します。

次に-fと入力します。ミニバッファーには‘auto-f’が入力されました。この文字で始まるコマンド名は、auto-fill-modeだけです。ここで TABを入力すると、残りの部分が補完されて、ミニバッファーの引数は‘auto-fill-mode’になります。

したがって a u TAB - f TABと入力するだけで、‘auto-fill-mode’と入力できるのです。

ミニバッファーの終端にポイントがなくても TABは機能します。この場合にはポイント位置とミニバッファーの終端の両方でテキストが補完されます。M-x autocmとタイプしてから C-bを押下して‘m’の前にポイントを移動、それから TABをタイプすればポイント位置に‘onf-’、ミニバッファーの終端に‘ode’が挿入されて、ミニバッファーの内容は‘autoconf-mode’になるのです。

5.4.2 補完コマンド

以下は補完が使えるときに、ミニバッファーで定義されている補完コマンドの一覧です。

TAB　　可能な限りミニバッファーのテキストを補完します。補完できないときは、可能性のある補完候補のリストを表示します (minibuffer-complete)。

SPC　　ミニバッファーのテキストを単語単位で補完します (minibuffer-complete-word)。このコマンドは、引数にスペースが含まれる可能性のあるファイル名などでは利用できません。

RET　　最初に可能な限り補完した後で、ミニバッファーのテキストを引数として確定します。Section 5.4.3 [Completion Exit], page 33 を参照してください。

?　　補完候補の一覧を表示します (minibuffer-completion-help)。

TAB (minibuffer-complete) は、もっとも基本的な補完コマンドです。これはミニバッファーのテキストとマッチする可能性のある、すべての補完候補を検索して、できるかぎりの補完を試みます。補完候補が選択される方法については、Section 5.4.4 [Completion Styles], page 34 を参照してください。

SPC (minibuffer-complete-word) は、TABと同じように補完をおこないますが、次のハイフンまたは空白までしか補完しません。ミニバッファーが 'auto-f'の場合、'auto-fill-mode'まで補完できますが、'ill-'しか挿入しないので 'auto-fill-'となります。次に SPCを入力すると 'auto-fill-mode'が補完されます。

TABや SPCが補完できない場合、マッチする補完候補のリスト (複数ある場合) を、別のウィンドウに表示します。同じリストは? (minibuffer-completion-help) でも表示できます。以下は補完一覧で使うことができるコマンドです:

M-DOWN
M-UP　　ミニバッファーにいる間は M-DOWN (minibuffer-next-completionと M-UP (minibuffer-previous-completion) のキーを用いて、補完候補を表示しているバッファー内の候補間を移動することができます。minibuffer-completion-auto-chooseが非 nil(デフォルト) の際には、これらのコマンドを使うことで、補完のカレント候補のミニバッファーへの挿入も行われます。minibuffer-completion-auto-chooseが nilの場合には、M-RET (minibuffer-choose-completion) コマンドを使うことで補完候補をミニバッファーに挿入できます。これによってデフォルトではミニバッファーを exit しますが、C-u M-RETのようにプレフィックス引数を指定することで、ミニバッファーを exit することなくカレントでアクティブな候補を挿入することができます。

M-v
PageUp
prior　　ミニバッファーで M-vを入力すると、候補リストを表示しているウィンドウを選択します (switch-to-completions)。以下のコマンドを使うには、この方法がよいでしょう。PageUp、prior、および M-g M-cは同じことをおこないます。他の方法でもウィンドウを選択できます (Chapter 17 [Windows], page 186 を参照)。

RET
mouse-1
mouse-2　　補完リストを表示しているバッファーにいる際にはポイント位置にある補完候補を選択します (choose-completion)。C-u RETのようにプレフィックス引数を指定することで、ミニバッファーを exit せずにポイント位置の候補が挿入されるので、気が変わったら別の候補を選択することができます。

TAB

RIGHT

`n` 補完候補リストのバッファー内では、これらのキーは次の補完候補にポイントを移動します (`next-completion`)。

`S-TAB`

`LEFT`

`p` 補完候補リストのバッファー内では、これらのキーは前の補完候補にポイントを移動します (`previous-completion`)。

`q` 補完候補リストのバッファー内ではリストバッファーを表示中のウィンドウを quit して、ミニバッファーを表示中のウィンドウを選択します

`z` 補完候補リストのバッファー内ではリストバッファーを kill してそれを表示中のウィンドウを削除します (`kill-current-buffer`)。

5.4.3 補完の終了

コマンドがミニバッファーの補完を使って引数を読みとる場合、引数を確定するために `RET` (`minibuffer-complete-and-exit`) をタイプしたときに、何が起こるかも制御します。これには 4 種類の動作があります:

- 強い補完 *(Strict completion)* は、正確にマッチする補完のみを許します。`RET`でミニバッファーを抜けるのは、ミニバッファーのテキストが正確にマッチしているか、1 つに補完された場合だけです。それ以外の場合、Emacs はミニバッファーからの exit(入力を完了してミニバッファーから抜け出す) を拒絶します。かわりに補完を試み、補完できなかったときは、ミニバッファーのテキストの後ろに数秒 '`[No match]`'と表示します (`C-g`を使えばミニバッファーを離れることができます)。

  この動作をおこなうコマンドの例は `M-x`で、それは存在しないコマンド名を受けとるのは無意味だからです。

- 慎重な補完 *(Cautious completion)* は強い補完と似ていますが、テキストがすでに正確にマッチしているときだけ exit できる点が異なります。テキストが正確なマッチに補完できるとき、`RET`は補完を行いますが、まだ exit しません。exit するには、もう一度 `RET`を入力しなければなりません。

  慎重な補完は、たとえば存在しなければならないファイル名を読みとるときに使用されます。

- 寛大な補完 *(Permissive completion)* は、任意の入力を許容します。補完候補はあくまでも提案です。`RET`では補完は行われず、単に入力された引数を確定します。

- 確認付きの寛大な補完 *(Permissive completion with confirmation)* は、寛大な補完と似ていますが例外があります。`TAB`を入力して、テキストがある中間的な状態まで補完されたとき (たとえばまだ正確なマッチに至らないとき)、次に `RET`を入力しても引数は確定されません。かわりに Emacs はテキストの後ろに '`[Confirm]`'を数秒表示して、確認を求めます。その次の `RET`は確認とみなされテキストが確定されます。これにより `TAB`により希望するマッチまで補完されたと勘違いして、`RET`を押してしまうなどの一般的な間違いを捕らえることができます。

  変数 `confirm-nonexistent-file-or-buffer`をカスタマイズして、確認動作を微調整できます。デフォルト値の `after-completion`は、まさに説明したとおりに動作します。これを `nil`に変更すると、Emacs は確認を求めなくなり、寛大な補完にフォールバックします。他の非 `nil`値に変更した場合、その前のコマンドが `TAB`かどうかにかかわらず、Emacs は確認を求めます。

  この動作はファイル名を読みとる `C-x C-f`や、バッファー名を読み取る `C-x b`など、多くのコマンドで使われています。

5.4.4 補完候補が選択される方法

補完コマンドは、たくさんの可能性のある補完候補を、ミニバッファーに入力したものとマッチ (match) する、より少ないサブセットへと絞り込むことにより機能します。Section 5.4.1 [Completion Example], page 31 では、そのようなマッチングの簡単な例を紹介しました。どのような構成がマッチなのかを決定する手続きはとても複雑です。Emacs は多くの状況下でもっとも妥当と思われる補完を試みます。

Emacs は 1 つ以上の補完スタイル *(completion styles)* を使って補完をおこないます。これはミニバッファーのテキストを補完候補とマッチングするための条件のセットです。補完を行うとき、Emacs は補完スタイルを順番に試します。もしあるスタイルが 1 つ以上のマッチを獲得した場合、それらは補完候補リストのために使用されます。もしあるスタイルがマッチを獲得できなかった場合、Emacs は次のスタイルにフォールバックします。

リスト変数 `completion-styles`は、使用する補完スタイルを定義します。それぞれのリスト要素 (list element) は、補完スタイルの名前 (Lisp シンボル) です。利用できるスタイルシンボルは変数 `completion-styles-alist`に格納されています (Section “Completion Variables” in *The Emacs Lisp Reference Manual* を参照)。デフォルトの補完スタイルは順番に:

`basic` ミニバッファーのポイントより前のテキストと、補完候補の先頭が同じでなければなりません。さらにミニバッファーのポイントより後ろのテキストがある場合、補完候補の残りそれが含まれていなければなりません。

`partial-completion`
このアグレッシブな補完スタイルは、ミニバッファーのテキストをハイフンまたは空白で区切り、各単語ごとに補完をおこないます (たとえばコマンド名を補完する場合、‘`em-l-m`’は、‘`emacs-lisp-mode`’に補完されます)。

さらにミニバッファーのテキスト中の ‘`*`’は、ワイルドカード *(wildcard)* として扱われます。これは補完候補の対応する位置にある文字列の、任意の文字とマッチします。

`emacs22` この補完スタイルは `basic`とにていますが、ミニバッファーのポイントより後のテキストを無視します。この名前は補完の動作が Emacs 22 と同じだからです。

以下の追加の補完スタイルが定義されており、`completion-styles`に追加することもできます (Chapter 33 [Customization], page 499 を参照してください)。

`substring`
補完候補は、ミニバッファーのポイントより前のテキストと、ポイントより後のテキストが同じ順番で含まれていなければなりません。

したがって、ミニバッファーのテキストが ‘`foobar`’で、ポイントが ‘`foo`’と ‘`bar`’の間にある場合、‘`afoobbarc`’にマッチします。この場合 *a*、*b*、*c*は空文字列を含む任意の文字列です。

`flex` この `flx`、`fuzzy`、または `scatter`のような補完としても知られるアグレッシブな補完は部分文字列を順に使用することにより補完を試みます。たとえば ‘`foo`’は、‘`frodo`’や ‘`fbarbazoo`’にマッチするとみなすことができます。

`initials` このとてもアグレッシブな補完スタイルは、頭文字とイニシャルで補完を試みます。たとえばコマンド名の補完をする場合、‘`lch`’は ‘`list-command-history`’とマッチします。

`emacs21`と呼ばれる、とてもシンプルな補完スタイルもあります。このスタイルでは、ミニバッファーのテキストが ‘`foobar`’の場合、‘`foobar`’で始まるものだけにマッチします。

変数 completion-category-overridesを設定することにより、状況に応じて異なる補完スタイルを使うことができます。たとえばバッファー名を補完するときは、デフォルトで basicと substringだけを使うよう指定できます。

5.4.5 補完オプション

大文字小文字の違いは、コマンド名のように大文字小文字を区別する (case-sensitive) 引数では重要です。たとえばコマンド名の補完では、‘AU’では ‘auto-fill-mode’に補完されません。大文字小文字の違いは、それが問題にならない引数の補完では無視されます。

ファイル名を補完するとき、変数 read-file-name-completion-ignore-caseが非 nilなら、大文字小文字の違いは無視されます。GNU/Linux のように、ファイル名の大文字と小文字を区別するシステムでは、デフォルト値は nilです。Microsoft Windows のように、ファイル名の大文字と小文字を区別しないシステムでは、非 nilです。バッファー名を補完するとき、read-buffer-completion-ignore-caseが非 nilなら、大文字小文字の違いは無視されます。デフォルトは nilです。

通常 Emacs はファイル名を補完するとき、選ばれるべきではないと思われる、特定の候補を無視します。これはリスト変数 completion-ignored-extensionsにより決定されます。リストの要素はに文字列を指定します。それらの文字列で終わるファイル名は、補完候補としては無視されます。スラッシュ(/) で終わる要素は、ディレクトリー名を表します。completion-ignored-extensionsの標準的な値は".o"、".elc"、"~"を含むいくつかの要素です。たとえばディレクトリーに ‘foo.c’、‘foo.elc’があるとき、‘foo’は ‘foo.c’に補完されます。しかしすべての補完候補が無視すべき文字列で終わるとき、これらの候補は無視されません。前の例でいうと ‘foo.e’は ‘foo.elc’に補完されます。Emacs は補完候補リストで補完候補を表示するとき、completion-ignored-extensions を無視します。

Shell での補完は、ファイル名補完の拡張されたバージョンです。Section 31.5.7 [Shell Options], page 466 を参照してください。

completion-auto-helpに nilがセットされていると、補完コマンドは補完リストバッファーを表示しません。表示するには?を入力しなければなりません。値が lazyの場合、Emacs は 2 度目の補完を試みたときだけ、補完リストバッファーを表示します。もし補完すべきものがない場合には 1 度目の TABでは ‘Next char not unique’、2 度目の TABで補完リストバッファーが表示されます。値が alwaysであれば補完を試みると常に補完リストバッファーが表示されます。

これによる補完リストバッファーの初回の表示も completion-auto-helpによって制御されます。値が tか lazyなら、補完候補を表示するためにポップアップしたウィンドウは、Emacs が補完できたら削除されます (そして更に何かテキストがタイプされた際に Emacs が補完できなければ再びウィンドウがポップアップするかもしれません)。値が alwaysの場合には、ポップアップしたウィンドウは補完を終了した場合だけ削除されます。ハイブリッドな振る舞いをするのが値 visibleです。これは補完リストバッファーを表示するウィンドウをポップアップするかどうか判断する際は t、そのポップアップしたウィンドウを削除するかどうか判断する際には alwaysのように振る舞います。

Emacs が補完候補を表示するウィンドウをポップアップする際に、オプションでそのウィンドウを選択することができます。この振る舞いを有効にするには、ユーザーオプション completion-auto-selectを tにカスタマイズしてください。これによって Emacs が補完用のウィンドウをポップアップする際の TABキーの振る舞いが変更されます。TABを押下すると補完リストバッファーに切り替わり、カーソル移動コマンドで候補間を移動、そして RETで候補を選択できます。completion-auto-selectの値が second-tabなら 1 回目の TABで補完リストバッファーをポップアップ、2 回目でそのウィンドウに切り替わります。

completion-cycle-thresholdが非 nilのとき、補完コマンドは補完候補を循環 (cycle) することができます。ミニバッファーのテキストで 1 つ以上の補完候補がある場合は通常、補完コマンドは補完できた文字までを表示します。completion-cycle-thresholdを tに変更すると、補完コマンドは補完候補の中から最初の候補を表示します。それ以降の補完コマンドの呼び出しでは、その次の補完候補を循環的に表示します。completion-cycle-thresholdを数値 *n*にすると、補完候補が *n*以下のときだけ循環表示の動作をします。

補完を表示する際、Emacs は通常だと補完を表示するバッファーを新たにポップアップします。デフォルトでは補完はウィンドウ幅に合わせた列数を用いて水平方向にソートされますが、ユーザーオプション completions-formatをカスタマイズして変更できます。値が verticalであれば補完を垂直方向にソート、one-columnなら 1 つの列だけを使用します。

ユーザーオプション completions-sortは、‘*Completions*’バッファーにおける補完候補のソート順を制御します。デフォルトである alphabeticalはアルファベット順、値 nilはソートを無効にします。値は関数でもよく、その関数は補完候補のリストとともに呼び出されて、望ましい順にソートしたリストをリターンする必要があります。

completions-max-heightが非 nilなら、それは補完ウィンドウの高さを制限する値です。この値には、もしあればモードライン、ヘッダーライン、下ディバイダーを含めた行数を指定します。display-buffer-alistを使用すれば、Completion ウィンドウ表示に関するプロパティにたいしてより複雑な制御を行うことができます (Section “Action Alists for Buffer Display” in *The Emacs Lisp Reference Manual* を参照)。

変数 completions-header-formatは、補完リストの候補の前に表示する情報的な行を制御するフォーマット仕様文字列です。この文字列に‘%s’構文が含まれていれば、補完リストバッファーに表示されている補完の候補数に置き換えられます。このヘッダーラインの表示を抑制するには、この変数の値を nilにカスタマイズしてください。この変数の値となる文字列には、ヘッダーラインの外観を変更するためのテキストプロパティをもたせることができます。役に立つプロパティとしては faceや cursor-intangibleが挙げられます (Section “Properties with Special Meanings” in *The Emacs Lisp Reference Manual* を参照)。

completions-highlight-faceがフェイスの名前なら、RETのタイプやマウスのクリックによって選択されるカレントの補完候補をハイライトするためにそのフェイスが使用されます。この変数のデフォルト値は completions-highlightですが、値が nilならハイライトは無効になります。この機能には特別なテキストプロパティ cursor-faceが使用されています。

5.5 ミニバッファーヒストリー

ミニバッファーでタイプしたすべての内容は、後で簡単に再利用できるようにミニバッファーヒストリーリスト *(minibuffer history list)* に保存されます。補助される対象としては補完候補 (ファイル名、バッファー名、コマンド名といったもの等)、およびその他ミニバッファー入力すべて下さい含まれます。: 以下のコマンドにより、以前の入力や選択肢をすばやく探し出してミニバッファーに呼び出すことができます:

M-p ミニバッファーヒストリーの前のアイテム、つまり以前のアイテムに移動します (previous-history-element)。

M-n ミニバッファーヒストリーの次のアイテムに移動します (next-history-element)。

UP
DOWN M-pや M-nと同様ですが、前のヒストリーアイテムに移動する前に、複数行アイテムの前、または次の行に移動します (previous-line-or-history-element、および next-line-or-history-element)。

M-r *regexp* RET
: *regexp*にマッチする、ミニバッファーヒストリーの以前のアイテムに移動します (previous-matching-history-element)。

M-s *regexp* RET
: *regexp*にマッチする、ミニバッファーヒストリーの以降のアイテムに移動します (next-matching-history-element)。

ミニバッファーでの M-p (previous-history-element) は、ミニバッファーのヒストリーリストのアイテムを 1 つずつ移動します。M-pはヒストリーリストの以前のアイテムを取り出して、ミニバッファーの既存の内容を置き換えます。M-n (next-history-element) は、ミニバッファーのヒストリーリストを反対方向、つまり以降のエントリーを取り出してミニバッファーの既存の内容を置き換えます。

ミニバッファーヒストリーに以降のエントリーがないとき (たとえば前に 1 回も M-pを入力していないとき)、Emacs はデフォルト引数のリストから、あなたが入力するだろうと思われる値を取り出します。これは “未来のヒストリー” を移動すると考えることもできます。

ファイルにたいする “未来のヒストリー (future history)” には、カレントバッファー内のポイント位置にあるファイル名や URL のような、あなたが便利だと思うかもしれないいくつかの候補が含まれます。この場合、“未来のヒストリー” に配されるデフォルトは、オプション file-name-at-point-functionsの値にセットされた関数により制御されます。デフォルトでは、この値は ffapパッケージ ((Section 31.12.5 [FFAP], page 486) を参照) を呼び出します。これはポイント周辺のテキストからデフォルトとなるファイルや URL を推測します。この推測を無効にするには、このオプションを nil値にカスタマイズします。これにより “未来のヒストリー” に含まれるのは、(もしあれば) カレントバッファーで visit されたファイル、およびデフォルトディレクトリーのファイルだけになります。

矢印キーの UPと DOWNは、M-pや M-nのように機能しますが、カレントヒストリーアイテムが複数行の場合には、前 (または次) のヒストリーアイテムに移動する前に、前 (または次) の行に移動します。

ミニバッファーコマンドの M-pまたは M-nで挿入されたテキストを編集しても、ヒストリーリストのエントリーは変更されません。かわりに編集された引数が確定されたとき、ヒストリーリストの最後に追加されます。

M-r (previous-matching-history-element) でヒストリーリストの古い要素を、M-s (next-matching-history-element) で新しいエントリーを探すことができます。これらのコマンドはどちらも引数として正規表現 *(regular expression)* を要求し、それにマッチした最初のエントリーをミニバッファーに取り出します。正規表現についての説明は、Section 12.6 [Regexps], page 115 を参照してください。数引数 *n*を指定すると、それは *n*番目に一致したエントリーを取り出すことを意味します。これらのコマンドはミニバッファーから呼び出されたとはいえ、ミニバッファーを使って正規表現を読みとるという点では、変わったコマンドです。正規表現に大文字が含まれていると、大文字小文字を区別する検索 (Section 12.9 [Lax Search], page 120 を参照してください) となります。

ヒストリーをインクリメンタルサーチすることもできます。Section 12.1.7 [Isearch Minibuffer], page 112 を参照してください。

Emacs は引数の種類ごとに個別のリストを保持します。たとえばファイル名のリストは、ファイル名を読みとるすべてのコマンドで使われたファイル名、といった具合です。ほかのヒストリーリストとしてはバッファー名やコマンド名 (M-xで使われたもの)、コマンド引数 (query-replaceのような引数で使われたもの) があります。

変数 history-lengthには、ミニバッファーのヒストリーリストの最大の長さを指定します。リストが長くなりすぎたときは、一番古い要素を削除して新しい要素を追加します。tを指定したときは、長さは無制限になります。

変数 history-delete-duplicatesには、重複したヒストリーを削除するかを指定します。非 nilの場合、新しい要素が追加されるとリストからそれと同じ要素がすべて削除されます。デフォルトは nilです。

5.6 ミニバッファーでのコマンドの繰り返し

ミニバッファーを使用したコマンドはすべて、コマンドヒストリー *(command history)* という特別なヒストリーリストに記録されます。これにはコマンドの引数の値も記録されるので、コマンド全体を再実行できます。特に M-xはコマンド名を読みとるので、M-xを使用したものはすべてそこに記録されます。

C-x ESC ESC
: コマンドヒストリーから最近のミニバッファーのコマンドを再実行します (repeat-complex-command)。

M-x list-command-history
: コマンドヒストリー全体を表示します。表示されたすべてのコマンドは C-x ESC ESCで再実行できます。一番最近のものが先頭に表示されます。

C-x ESC ESCは、ミニバッファーを使った最近のコマンドを再実行します。引数を与えないと、一番最近のコマンドを実行します。数引数で再実行したいコマンドを指定します。1 は一番最近のコマンド、2 はその前、といった具合です。

C-x ESC ESCは、前のコマンドを Lisp 式に変換して、その式でミニバッファーのテキストを初期化します。Lisp を知らなくても、何のコマンドが再実行用に表示されているか明白でしょう。単に RETを入力すると、コマンドを変更せずに再実行します。実行する前に Lisp 式を編集して、コマンドを変更できます。実行されたコマンドは、一番最近のコマンドと等しくなければ、コマンドヒストリーの一番先頭に追加されます。

C-x ESC ESCで一度ミニバッファーの中に入れば、通常のミニバッファーのヒストリーコマンド (Section 5.5 [Minibuffer History], page 36 を参照してください) を使って、ヒストリーリスト中を移動できます。以前に実行したコマンドが見つけたら、式を編集して RETにより実行できます。

厳密に言うとインクリメンタルサーチはミニバッファーを使いません。これが複雑なコマンドのように振る舞うとしても、通常は C-x ESC ESC で表示されるヒストリーリストに含まれません。ヒストリーにインクリメンタルサーチコマンドを表示させるには、isearch-resume-in-command-historyに非 nilをセットしてください。Section 12.1 [Incremental Search], page 105 を参照してください。

ミニバッファーを使用した以前のコマンドのリストは、Lisp のリスト値として command-historyに格納されています。Lisp 式の各要素は、1 つのコマンドとその引数をあらわしています。Lisp プログラムは evalに command-historyの要素を使って呼び出すことで、コマンドを再実行できます。

5.7 パスワードの入力

Emacs でパスワードを入力したい場合があります。たとえば Emacs に FTP のようなネットワークプロトコルを介して他のマシンのファイルを visit し、マシンへのアクセスのためにパスワードを与える場合などです (Section 15.15 [Remote Files], page 169 を参照してください)。

パスワードの入力はミニバッファーの使用と似ています。Emacs は (‘Password: ’のような) プロンプトをエコーエリアに表示します。要求されたパスワードを入力した後、それを確定するために RETを押します。他の人がパスワードを見るのを防ぐために、入力した文字は通常の形式ではなくアスタリスク (‘*’) で表示されます。

ミニバッファーに関連付けられた多くの機能およびコマンドは、パスワード入力では使用できません。ヒストリーや補完はできず、ウィンドウの変更や Emacs での他の処理も、パスワードを submit するまでは行うことができません。

パスワードのタイプ中は、DELを押して後方に、すなわち最後に入力した文字から削除できます。C-uで入力したすべての文字を削除できます。C-gはパスワードプロンプトを閉じます (Section 34.1 [Quitting], page 536 を参照)。C-yは現在の kill リングからパスワードを挿入します (Chapter 9 [Killing], page 59 を参照)。パスワードを確定するには、RETまたは ESCを入力します。他の自己挿入文字は、対応する文字をパスワードに入力します. それ以外の入力は無視されます。

5.8 Yes or No プロンプト

Emacs のコマンドが実行の過程で、yes-or-no(はい/いいえ) 形式で質問して答えを求めるかもしれません。これらの質問は大きく 2 つにわけることができます。

1 番目の yes-or-no 形式の質問は ‘(y or n)’ で終わるプロンプトの質問です。‘y’か ‘n’いずれかの単一キーをタイプして質問に応答して、それによりミニバッーを即座に exit して応答を伝えます。たとえばバッファーの保存で C-x C-w (write-file) を入力して既存のファイル名を入力すると、Emacs は以下のようなプロンプトを表示します:

```
File ‘ foo.el ’ exists; overwrite? (y or n)
```

2 番目の yes-or-no 形式の質問は間違った答えが深刻な事態を招くような場合に使用される質問形式で、‘(yes or no)’を編集する長いプロンプトが特徴です。たとえば変更が保存されていないファイルを visit しているバッファーで C-x k (kill-buffer) を呼び出すと、Emacs はミニバッファーをアクティブにして以下のようなプロンプトを表示します:

```
Buffer foo.el modified; kill anyway? (yes or no)
```

これに答えるにはミニバッファーで ‘yes’か ‘no’の後に RETをタイプしなければなりません。

これらの yes-or-no による 2 つの質問形式では、前のセクションで説明したようにミニバッファーは振る舞います。C-lによる選択されたウィンドウの再センタリング、ウィンドウのスクロール (C-vか PageDownで前方、M-vか PageUpで後方にスクロール)、C-x oによる別ウィンドウへの切り替え、M-pと M-nによるヒストリーコマンドの使用等が可能です。質問を終了するためには C-gをタイプしてミニバッファーと問い合わせコマンドをを quit します (Section 34.1 [Quitting], page 536 を参照)。これに答えるには、ミニバッファーに ‘yes’または ‘no’と入力してから、RETを押さなければなりません。ミニバッファーは前のセクションで説明したように動作します。C-x oによる他のウィンドウへのスイッチ、ヒストリーコマンドの M-pや M-nなどが使用できます。C-gでミニバッファーを閉じて、質問を発したコマンドを終了できます。

6 名前を指定してコマンドを実行する

すべての Emacs コマンドは、それを使えば実行できるような名前を持っています。使いやすいようにキーバインディングされているコマンドも、たくさんあります。これらのコマンドはキーまたは名前で実行できます。キーがバインドされていないコマンドもたくさんあり、そのようなコマンドは名前でしか実行できません (キーバインドをセットアップする方法については、Section 33.3 [Key Bindings], page 519 を参照してください)。

慣例によりコマンド名は `auto-fill-mode`や `manual-entry`のように、ハイフンで区切られた、1 つ以上の単語となっています。コマンド名は覚えやすいように、略されていない英単語が主に使われます。

コマンドを名前で実行するには、最初に `M-x`をタイプしてから、コマンド名をタイプして `RET`で実行します。`M-x`はミニバッファーを使ってコマンド名を読みとります。ミニバッファーの先頭には、実行のために名前の入力が必要なことを気付かせるため、'`M-x`'という文字がプロンプトとして表示されます。`RET`でミニバッファーを抜けだしコマンドを実行します。ミニバッファーについての詳細は、Chapter 5 [Minibuffer], page 28 を参照してください。

コマンド名の入力には補完が使用できます。たとえば `forward-char`を呼び出すには以下のように入力できます

```
M-x forward-char RET
```

または

```
M-x forw TAB c RET
```

`forward-char`は、キー `C-f`で呼び出されるのと同じコマンドであることに注意してください。キーバインドの存在は、名前によるコマンドの実行を妨げません。

`M-x`でコマンドを補完するとき、Emacs の以前のメジャーバージョンのいずれかで *obsolete(*時代遅れ*)* と宣言されたコマンドは無視されます。これらのコマンドにたいしては、完全な名前をタイプする必要があります。Emacs のカレントバージョンで obsolete とマークされたコマンドは一覧されます (obsolete なコマンドとはもっと新しくより良い代替が存在していて、将来の Emacs リリースにおいて廃止が予定されているコマンドのこと)。

更に `M-x`補完ではカレントバッファーのメジャーモード (Section 20.1 [Major Modes], page 240 を参照) やマイナーモード (Section 20.2 [Minor Modes], page 241 を参照) とは無関係で、一般的には共に動作できないコマンドを除外できます。デフォルトでは除外されるコマンドはありませんが、これらの無関係なコマンドを補完結果から除外するために、ユーザーオプション `read-extended-command-predicate`をカスタマイズできます。

これとは逆に、Emacs はカレントバッファーと特に関連するものを除くすべてのコマンドを除外することもできます。`M-S-x` ("meta shift x") コマンドは `M-x`と同様に機能しますが、Emacs が関知するすべて (またはほとんど) のコマンドをリストするかわりに、カレントのメジャーモードや有効なすべてのマイナーモードに "belonging(所属する)" とマークされたコマンドだけをリストします。

`M-x`とコマンドの実行をキャンセルするには、コマンド名の入力のかわりに `C-g`を入力します。これによりコマンドを呼び出したレベルまで戻ります。

`M-x`で呼び出すコマンドに数引数を渡すには、`M-x`の前に数引数を指定します。引数の値はコマンド名が読みとられるときにプロンプトとして表示され、最終的に `M-x`は引数をコマンドに渡します。たとえば、コマンド `forward-char`に数引数として 42 を渡すには、`C-u 42 M-x forward-char RET` とタイプできます。

`M-x`で実行するコマンドがキーバインディングを持つ場合、Emacs はコマンド実行後にその旨をエコーエリアに表示します。たとえば `M-x forward-word`と入力すると、同じコマンドを `M-f`で実行

できるというメッセージが表示されます。このメッセージは変数 suggest-key-bindingsに nilにセットすればオフにできます。suggest-key-bindingsの値には数字も指定でき、この場合 Emaceは指定された秒数の間、キーバインドを表示します。バインディングを表示するときのデフォルトは 2 秒です。

更に suggest-key-bindingsが非 nilの際には、M-xの補完リストはそのキーバインディングをもつすべてのコマンドにたいして、等価なキーバインディングを表示します。

キーバインドをもたないコマンドも、‘M-x’のプロンプトで、完全な名前より短くタイプして呼び出すことができます。Emacs は、短縮入力が完全なコマンド名より明らかに短く、extended-command-suggest-shorterが非 nilの場合は、そのような短縮入力をエコーエリアに表示します。suggest-key-bindingsのセッティングは、そのようなヒントにも同様に効果を及ぼします。

このマニュアルではコマンドを名前で実行するとき、名前を終了させるための RETを省くことがあります。つまり M-x auto-fill-mode RET ではなく M-x auto-fill-modeと表記します。RETはコマンドに引数がある場合に強調させる意味で使用します。

M-xは execute-extended-commandを実行します。これは他のコマンドの名前を読み取って実行するコマンドです。

7 ヘルプ

Emacs はバラエティに富んだヘルプコマンドを提供しており、それらにはプレフィクスキー C-h(ファンクションキー F1でも可) からアクセスできます。以下のセクションではこれらのコマンドについて説明します。C-h C-h (help-for-help) と入力すれば、ヘルプコマンドの一覧を見ることもできます。この一覧は SPCと DELでスクロールでき、それから見たいヘルプコマンドを入力するか、C-gでキャンセルできます。

多くのヘルプコマンドは、ヘルプバッファー *(help buffer)* という特別なバッファーに情報を表示します。このバッファーでは SPCと DELによりスクロールし、RETでハイパーリンクをフォローすることができます。Section 7.4 [Help Mode], page 48 を参照してください。

ヘルプコマンドはデフォルトではヘルプバッファーを選択することなく、別のウィンドウに表示します。これを制御するのが変数 help-window-selectであり、デフォルト値は nilです。この変数の値を tにカスタマイズするとヘルプバッファーはヘルプウィンドウによって無条件に選択されます。値が otherなら選択されたフレームのウィンドウが 3 つ以上の場合のみヘルプウィンドウが選択されます。

これとは対照的に '*Help*'バッファーの多くのコマンドが、結果を表示するために新たなウィンドウをポップアップします。たとえばソースコードを表示するリンクをクリックしたり、マニュアルのエントリーを表示するコマンド iを使用すると、(デフォルトでは) 新たなウィンドウがポップアップされます。しかし help-window-keep-selectedを非 nilに変更すれば、'*Help*'バッファーを表示しているウィンドウが再利用されるようになります。

特定の機能を探しているけどそれが何と呼ばれているかわからない、どこを見ればよいのかわからないときには次の 3 つの方法を推奨します。まず最初に apropos コマンドを試してください。次にマニュアルのインデックスを探してください。そして FAQ とパッケージのキーワードを探してください。最後に外部パッケージのリストを調べてみましょう。

C-h a *topics* RET
: 引数 *topics*と名前が一致するコマンドを検索します。引数にはキーワード、キーワードのリスト、正規表現 (Section 12.6 [Regexps], page 115 を参照してください) を指定できます。Section 7.3 [Apropos], page 46 を参照してください。

C-h i d m emacs RET i *topic* RET
: Emacs の info マニュアルのインデックスから、*topic*を検索して最初にマッチしたものを表示します。,を押すと次にマッチしたものを表示します。*topic*には正規表現を指定できます。

C-h i d m emacs RET s *topic* RET
: 同様ですが、インデックスではなくマニュアル本文のテキストを検索します。

C-h C-f
: Info を使って Emacs FAQ を表示します。

C-h p
: キーワードに基づいて、利用可能な Emacs パッケージを表示します。Section 7.5 [Package Keywords], page 49 を参照してください。

M-x list-packages
: 外部パッケージのリストを表示します。Chapter 32 [Packages], page 490 を参照してください。

他のさまざまなコンテキストでも、C-hや F1は “ヘルプ” を意味します。たとえばプレフィクスキーの後にこれらのキーを入力すると、プレフィクスキーに続けて入力できるキーの一覧を表示する

ことができます (このコンテキストでは?も使用できます。いくつかのプレフィクスキーは C-hや?に別の意味を持たせているものがあるので使えませんが、それらのものでも F1はサポートされています)。

ここではビルトインのドキュメントにアクセスする、ヘルプコマンドの要約を記します。これらの大部分については、以下のセクションで詳細を説明します。

C-h a *topics* RET
: 名前が *topics*にマッチするコマンドの一覧を表示します (apropos-command)。Section 7.3 [Apropos], page 46 を参照してください。

C-h b
: 有効なキーバインディングをすべて表示します。最初はマイナーモード、次にメジャーモード、それからグローバルのバインディングを表示します (describe-bindings)。Section 7.7 [Misc Help], page 49 を参照してください。

C-h c *key*
: キーシーケンス *key*がバインドされているコマンドの名前を表示します (describe-key-briefly)。cは “character” が由来です。*key*についてさらに詳しい情報を得るには C-h kを使います。Section 7.1 [Key Help], page 45 を参照してください。

C-h d *topics* RET
: *topics*にマッチするドキュメントをもつ、コマンドまたは変数を表示します (apropos-documentation)。Section 7.3 [Apropos], page 46 を参照してください。

C-h e
: バッファー*Messages*を表示します (view-echo-area-messages)。Section 7.7 [Misc Help], page 49 を参照してください。

C-h f *function* RET
: *function*という名前の Lisp 関数のドキュメントを表示します (describe-function)。コマンドも Lisp 関数なのでコマンドにも使用できますが、C-h xも使用できます。Section 7.2 [Name Help], page 45 を参照してください。

C-h h
: ファイル HELLOを表示します。このファイルは様々な文字セットの例です。

C-h i
: GNU ドキュメントブラウザー (info) を実行します。Emacs のマニュアルは info で利用できます。Section 7.7 [Misc Help], page 49 を参照してください。

C-h k *key*
: *key*で実行されるコマンドの名前と、ドキュメントを表示します (describe-key)。Section 7.1 [Key Help], page 45 を参照してください。

C-h l
: 最近のキーストローク 300 回分の説明を表示します (view-lossage)。Section 7.7 [Misc Help], page 49 を参照してください。

C-h m
: カレントのメジャーモード、およびマイナーモードのドキュメントを表示します (describe-mode)。Section 7.7 [Misc Help], page 49 を参照してください。

C-h n
: 最近の Emacs の変更に関するニュースを表示します (view-emacs-news)。Section 7.8 [Help Files], page 50 を参照してください。

C-h o *symbol*
: *symbol*という名前の Lisp シンボルのドキュメントを表示します (describe-symbol)。これはすべての種類のシンボル、すなわち関数、変数、フェイスのドキュメントも表示します。Section 7.2 [Name Help], page 45 を参照してください。

C-h p
: トピックのキーワードでパッケージを見つけます (finder-by-keyword)。Section 7.5 [Package Keywords], page 49 を参照してください。これはパッケージメニューバッファーを使ってパッケージを一覧します。Chapter 32 [Packages], page 490 を参照してください。

`C-h P` *package* `RET`
: 指定したパッケージのドキュメントを表示します (`describe-package`)。Section 7.5 [Package Keywords], page 49 を参照してください。

`C-h r`
: Emacs のマニュアルを info で表示します (`info-emacs-manual`)。

`C-h s`
: 現在の構文テーブル *(syntax table)* を表示します (`describe-syntax`)。Section 7.7 [Misc Help], page 49 を参照してください。構文テーブルはどの文字が開始デリミッターで、その文字が単語の一部なのかを定義します。詳細については Section "Syntax Tables" in *The Emacs Lisp Reference Manual* を参照してください。

`C-h t`
: Emacs の対話的なチュートリアルを開始します (`help-with-tutorial`)。

`C-h v` *var* `RET`
: Lisp 変数 *var*のドキュメントを表示します (`describe-variable`)。Section 7.2 [Name Help], page 45 を参照してください。

`C-h w` *command* `RET`
: *command*という名前のコマンドを実行するキーを表示します (`where-is`)。Section 7.1 [Key Help], page 45 を参照してください。

`C-h x` *command* `RET`
: 名前 *command*のドキュメントを表示します (`describe-command`)。Section 7.2 [Name Help], page 45 を参照してください。

`C-h C` *coding* `RET`
: コーディングシステム *coding*を説明します (`describe-coding-system`)。Section 19.5 [Coding Systems], page 223 を参照してください。

`C-h C RET`
: 現在使用されているコーディングシステムを説明します。

`C-h F` *command* `RET`
: info を開始して、Emacs コマンド *command*のドキュメントのノードに移動します (`Info-goto-emacs-command-node`)。Section 7.2 [Name Help], page 45 を参照してください。

`C-h I` *method* `RET`
: インプットメソッド *method*を説明します (`describe-input-method`)。Section 19.4 [Select Input Method], page 222 を参照してください。

`C-h K` *key*
: info を開始して、キーシーケンス *key*のドキュメントのノードに移動します (`Info-goto-emacs-key-command-node`)。Section 7.1 [Key Help], page 45 を参照してください。

`C-h L` *language-env* `RET`
: 言語環境 (language environment) *language-env*で使用される文字セット、コーディングシステム、インプットメソッドに関する情報を表示します。Section 19.2 [Language Environments], page 218 を参照してください。

`C-h S` *symbol* `RET`
: 編集中のプログラム言語に基づいて、シンボル *symbol*の info ドキュメントを表示します (`info-lookup-symbol`)。Section 7.7 [Misc Help], page 49 を参照してください。

`C-h .`
: ポイントが特別なテキスト領域のとき (たとえば`*Help*`のリンクなどが含まれる) にはヘルプメッセージを表示します (`display-local-help`)。Section 7.9 [Help Echo], page 51 を参照してください。このコマンドを `C-u C-h .`のようにプレフィックス引数

とともに呼び出すと、ボタンやウィジェット上にポイントがあればボタンやウィジェットを説明するバッファーを新たにポップアップします。

7.1 キーのドキュメント

キーシーケンスに関する情報を得るためのヘルプコマンドは、C-h c (describe-key-briefly) と C-h k (describe-key) です。

C-h c *key*は、*key*にバインドされているコマンドの名前を、エコーエリアに表示します。たとえば C-h c C-fは、'forward-char'と表示します。

C-h k *key*も同様ですが、さらに多くの情報が得られます。これはコマンドが何をするかが正確に説明する、ドキュメント文字列 *(documentation string)* を含むヘルプバッファーを表示します。

C-h K *key*は、*key*に対応するコマンドが説明されている、Emacs マニュアルのセクションを表示します。

C-h c、C-h k、C-h Kはファンクションキー、メニュー、マウスイベント (ただし C-h cはマウス移動イベントを無視する) を含む、任意のキーシーケンスにたいして動作します。たとえば C-h kの後で、メニューバーからメニューアイテムを選択すれば、それにより実行されるコマンドのドキュメントが表示されます。

C-h w *command* RETは、*command*がバインドされているキーをリストします。リストはエコーエリアに表示されます。コマンドがキーにバインドされていないとき、それは M-xを使って実行しなければならないことを意味します。C-h wはコマンド where-isを実行します。

Emacs では何らかのアクションを起こすクリック可能な種々のボタン (Section "Buttons" in *The Emacs Lisp Reference Manual* を参照) やウィジェット (Section "Introduction" in *Emacs Widgets* を参照) を使用するモードがあります。これらのボタンによって何の関数が最終的に呼び出されるかを調べるために、Emacs は button-describeコマンド (ボタン上にポイントを置いて実行する必要がある) を提供します。

7.2 コマンドと変数名のヘルプ

C-h x *command* RET (describe-command) は *command*という名前のコマンドのドキュメントをウィンドウに表示します。たとえば、

```
C-h x auto-fill-mode RET
```

は auto-fill-modeのドキュメントを表示します。これはキーにバインドされていないコマンド (通常 M-xで実行する) のドキュメントを閲覧する方法です。

C-h f *function* RET (describe-function) は Lisp 関数 *function*のドキュメントを表示します。このコマンドは Lisp プログラム内で使用される Lisp 関数を意図しています。たとえば式 (make-vector len)を記述した直後に、make-vectorを正しく使用できているか確認したいときには C-h f make-vector RET とタイプします。加えてすべてのコマンドは Lisp 関数なので、任意のコマンドのドキュメントを閲覧するためにもこのコマンドを使用できます。

C-h f RETを入力すると、バッファーのポイントがある位置の、一番内側の Lisp 式で呼び出されている関数の名前が有効で、Lisp 関数として定義されていれば、その関数の説明が表示されます (引数を入力するときデフォルトとして関数名が表示されます)。たとえばポイントが '(make-vector (car x)'の後ろにあるとき、ポイントを含む一番内側のリストは '(make-vector'で始まっているので、C-h f RET により関数 make-vectorの説明が表示されます。

C-h fは、関数名を正しく記述しているか確かめたいときも便利です。C-h fのミニバッファーのプロンプトで、編集中のバッファーの関数名がデフォルトとして表示されるなら、それは定義された

Lisp 関数であることを意味します。本当にドキュメントを見たい訳でなければ、C-gを押してコマンド C-h fをキャンセルできます。

autoloadフォーム (Section "Autoload" in *The Emacs Lisp Reference Manual* を参照) でドキュメント文字列を提供しない autoloaded 関数にたいするヘルプを要求する場合には、*Help* バッファーに表示するドキュメント文字列は存在しないでしょう。このような場合、help-enable-symbol-autoloadが非 nilなら Emacs はドキュメント文字列が存在するかどうか確認するためにその関数が定義されているファイルのロードを試みます。

M-x shortdocコマンドを使用して、特定トピックに関連する関数の概要を取得できます。これはたとえば stringのように、興味のある領域の入力を求めて、多くの文字列処理関連の関数をリストしたバッファーをポップアップします。

C-h v (describe-variable) は、C-h fと似ていますが Lisp 関数ではなく、Lisp 関数の説明を表示します。ポイントの周囲または前にある Lisp シンボルが、定義された Lisp 変数名のときはそれがデフォルトとなります。Section 33.2 [Variables], page 508 を参照してください。

通常 Emacs の変数または関数を説明するヘルプバッファーには、ソースファイルがインストールされていれば、対応するソースコードへのハイパーリンクが含まれています (Section 31.12 [Hyperlinking], page 485 を参照してください)。

マニュアルからコマンドのドキュメントを探すには、C-h F (Info-goto-emacs-command-node) を使います。これは Emacs のマニュアルだけでなく、さまざまなマニュアルを認識するので、正しいものを見つけられるでしょう。

C-h o (describe-symbol) は、C-h fや C-h vと似ていますが、これは任意のシンボル (関数、変数、フェイス) を説明します。シンボルが複数の定義をもつ場合 (たとえばシンボルが関数と変数の両方の定義をもつような場合)、このコマンドはそれらすべてのドキュメントを順に表示します。

ユーザーオプション completions-detailedが非 nilなら、いくつかのコマンドは補完の表示時に可能な値に関する詳細を提供します。たとえば C-h o TABならドキュメント文字列の最初を含めて、更にそれぞれのシンボルが関数なのか変数 (あるいはその他) なのかを示します。どの詳細が含まれるかはコマンドの用途に依存します。

7.3 Apropos(適切な)

*apropos(*折よい、適切な*)* コマンドは、"What are the commands for working with files?(ファイルを処理するコマンドは?)" のような質問に答えるものです。より正確に言うと、これはあなたの問いを形式の *apropos* パターン (単語、単語のリスト、あるいは正規表現) として指定するコマンドです。

以下の各 apropos コマンドは、ミニバッファーで apropos パターンを読み取り、パターンにマッチするアイテムを検索して、結果を別のウィンドウに表示します。

C-h a
: コマンドを検索します (apropos-command)。プレフィクス引数を指定すると、非インタラクティブな関数も検索します。

M-x apropos
: 関数と変数を検索します。これを使えばインタラクティブな関数 (コマンド) と非インタラクティブな関数の両方を検索できます。

M-x apropos-user-option
: ユーザーがカスタマイズできる関数を検索します。プレフィクス引数を指定すると、カスタマイズできない変数も検索します。

`M-x apropos-variable`
: 変数を検索します。プレフィクス引数を指定すると、カスタマイズできる変数だけを検索します。

`M-x apropos-local-variable`
: バッファーローカル変数を検索します。

`M-x apropos-value`
: 指定したパターンにマッチする値の変数を検索します。プレフィクス引数を指定すると、定義がパターンにマッチする関数と、プロパティーリストがパターンにマッチする、Lisp シンボルも検索します。

`M-x apropos-local-value`
: 値が指定されたパターンにマッチする、バッファーローカル変数を検索します。

`C-h d`
: 指定されたパターンにマッチするドキュメント文字列の関数、または変数を検索します (`apropos-documentation`)。

最も簡単な種類の apropos パターンは単語です。その単語が含まれていれば、パターンにマッチします。ファイルを処理するコマンドを探すには、`C-h a file RET`と入力します。これは `copy-file`、`find-file`などの、名前に '`file`'を含むすべてのコマンドの一覧を表示します。各コマンド名には簡単な説明と、それを呼び出すためのキーの一覧が一緒に表示されます。この例だと、`find-file`は `C-x C-f`で呼び出せることがわかります。

デフォルトでは問い合わせた結果である apropos バッファーを表示するウィンドウは選択されませんが、変数 `help-window-select`を任意の非 `nil`値にカスタマイズして選択されるようにすることもできます。

apropos バッファーの関数定義、変数、シンボルの属性についてもっと情報が欲しいときは、`mouse-1`または `mouse-2`でクリックするか、そこに移動して `RET`を押してください。

2 つ以上の単語を apropos パターンに指定したときは、マッチするために少なくとも名前に 2 つの単語が含まれていなければなりません。たとえばポイントの前のテキストを kill するコマンドを探すときは、`C-h a kill back backward behind before RET`などと試みることができます。これは実際のコマンド名 `kill-backward`にマッチするでしょう。もし `kill-text-before`というコマンドがあったなら、これも指定した単語を 2 つ含んでいるのでマッチします。

より一層フレキシビリティを高めるために、正規表現 (Section 12.6 [Regexps], page 115 を参照) を指定できます。apropos パターンに正規表現の特殊文字 '`^$*+?.\[`'が含まれていると、それは正規表現として扱われます。

Emacs コマンド命名の慣習に従うと、apropos パターンとして便利な単語がいくつかあります。これらを `C-h a`で使うことにより、命名の慣習の必要性を感じることができるでしょう。

> char、line、word、sentence、paragraph、region、page、sexp、list、defun、
> rect、buffer、frame、window、face、file、dir、register、mode、beginning、end、
> forward、backward、next、previous、up、down、search、goto、kill、delete、
> mark、insert、yank、fill、indent、case、change、set、what、list、find、view、
> describe、default

変数 `apropos-do-all`が非 `nil`の場合、ほとんどの apropos コマンドは、プレフィクス引数が与えられたかのように振る舞います。例外が 1 つあります。プレフィクス引数なしの `apropos-variable`は、`apropos-do-all`の値に関係なく、常にすべての変数を検索します。

デフォルトでは、`apropos-documentation`を除くすべての apropos コマンドは、結果をアルファベット順に一覧します。変数 `apropos-sort-by-scores`が非 `nil`のときは、かわりに結果の関

連度を推測して、一番関連度が高いと思われるものを最初に表示します。apropos-documentation コマンドは、デフォルトで結果を関連度順で一覧します。これをアルファベット順にするには、変数 apropos-documentation-sort-by-scoresを nilに変更してください。

7.4 ヘルプモードのコマンド

ヘルプバッファーのメジャーモードは Help モードです。この Help モードは View モードと同じコマンドが提供されます (Section 11.6 [View Mode], page 82 を参照)。たとえば前方へのスクロールは SPC、後方へのスクロールは DELか S-SPCです。他にも特別なコマンドがいくつか提供されます:

RET ポイント位置のクロスリファレンス先をフォローします (help-follow)。

TAB ポイントを次のハイパーリンクへ進めます (forward-button)。

S-TAB ポイントを前のハイパーリンクへ戻します (backward-button)。

mouse-1
mouse-2 クリックしたハイパーリンク先をフォローします。

n
p Help バッファー内のページを前方または後方に移動します。

C-c C-c ポイント位置のシンボルに関する、すべてのドキュメントを表示します (help-follow-symbol)。

C-c C-f
r ヘルプコマンドのヒストリーを前方に移動します (help-go-forward)。

C-c C-b
l ヘルプコマンドのヒストリーを後方に移動します (help-go-back)。

s (もしあれば) カレントヘルプトピックのソースを閲覧します (help-view-source)。

i マニュアル (複数可) からカレントトピックを照会します (help-goto-info)。

I Emacs Lisp マニュアルからカレントトピックを照会します (help-goto-lispref-info)。

c 変数またはフェイスをカスタマイズします (help-customize)。

もし関数名、変数名、フェイス名 (Section 11.8 [Faces], page 82 を参照してください) がヘルプバッファーのドキュメントにあると、通常はアンダーラインされたハイパーリンク *(hyperlink)* として表示されます。関連するドキュメントを閲覧するには、ポイントをそこに移動して RET (help-follow) とタイプするか、ハイパーリンクを mouse-1または mouse-2でクリックします。するとヘルプバッファーの内容が入れ替わりますが、C-c C-b、または l (help-go-back) で元に戻すことができます。元に戻る途中では、C-c C-fまたは r (help-go-forward) を使用して前に進むことができます。

ヘルプバッファーでは TAB (forward-button) で次のハイパーリンクに前方へ移動、S-TAB (backward-button) で前のハイパーリンクに後方へ移動します。ヘルプバッファーでハイパーリンク間を移動するには TAB (forward-button) で次のハイパーリンクへ前方に移動、S-TAB (backward-button) で前のハイパーリンクに後方へ移動します。

デフォルトではヘルプバッファー内のリンクの多くはクォート文字で囲われて表示されます。ユーザーオプション help-clean-buttonsが非 nilだと、これらのクォート文字はバッファーから取り除かれます。

(キーバインディングの長大なリストを表示する C-h bなど) 一部の Help コマンドが生成する Help バッファーは ‘^L’文字によって複数のページに分割されます。そのようなバッファーではコマンド n (help-goto-next-page) で次ページ先頭、p (help-goto-previous-page) で前ページ先頭に移動することができます。この方法を使えば、ヘルプバッファー内の異なる種類のドキュメント間を素早く移動することができます。

ヘルプバッファーには、info マニュアル、ソースコード定義、URL(ウェブページ) へのハイパーリンクを含むこともできます。最初の 2 つは Emacs 上で表示され、3 番目の URL はコマンド browse-url により、ウェブブラウザーを使って表示されます。

テキスト中のシンボルに関するすべてのドキュメントを閲覧するには、ポイントをシンボルに移動して C-c C-c (help-follow-symbol) とタイプします。これは変数、関数、および/またはフェイスとしての、そのシンボルが意味する、すべてのドキュメントを表示します。

7.5 パッケージのキーワード検索

Emacs のたいていのオプション機能は、パッケージ *(packages)* にグループ化されています。Emacs には数百のビルトインパッケージが含まれており、ネットワークを通じて他のパッケージのインストールもできます (Chapter 32 [Packages], page 490 を参照してください)。

あるトピックに関連するパッケージを探すのを簡単にするため、ほとんどのパッケージは、それが何をするかにもとづき、1 つ以上のキーワード *(keywords)* に関連付けられています。C-h p (finder-by-keyword) とタイプすると、パッケージキーワードとキーワードの意味を説明するリストが表示されます。キーワードに属するパッケージのリストを表示するには、そのキーワードの行で RETをタイプします。これによりパッケージメニューバッファー (Section 32.1 [Package Menu], page 490 を参照してください) で、パッケージの一覧が表示されます。

C-h P (describe-package) はパッケージ名 (Chapter 32 [Packages], page 490 を参照) の入力を求めて、パッケージの属性と、それが実装する機能をヘルプバッファーで表示します。このバッファーは、ボタン形式で関連するパッケージのキーワードを一覧します。ボタン mouse-1か mouse-2 をクリックすると、そのキーワードに関連する他のパッケージを見ることができます。

7.6 国際化言語のサポートに関するヘルプ

特定の言語環境 (language environment: Section 19.2 [Language Environments], page 218 を参照してください) の情報を得るには、C-h Lとタイプします。これはヘルプバッファーを開いて、言語環境でサポートされる言語の説明と、関連する文字セット、コーディングシステム、インプットメソッド、およびその言語環境のサンプルテキストを表示します。

コマンド C-h h (view-hello-file) は、ファイル etc/HELLOを表示します。このファイルはさまざまな言語で “hello” をどのように言うのかを、いろいろな文字セットで表示するデモンストレーションです。

コマンド C-h I (describe-input-method) は、指定されたインプットメソッド、または現在使われているインプットメソッド (デフォルト) の説明します。Section 19.3 [Input Methods], page 220 を参照してください。

コマンド C-h C (describe-coding-system) は、指定されたコーディングシステム、または現在使われているものを説明します。Section 19.5 [Coding Systems], page 223 を参照してください。

7.7 その他のヘルプコマンド

C-h i (info) は info プログラムを実行します。info は構造化されたドキュメントファイルを閲覧するものです。C-h 4 i (info-other-window) は同じことを行いますが、別のウィンドウに Info バッ

ファーを表示します。Emacs マニュアル全体は、GNU システムのための他のマニュアルとともに info で利用可能です。info を開始した後に hをタイプすると、info の使い方のチュートリアルが実行されます。

数引数 *n*を指定すると、C-h iは info バッファー '*info*<*n*>'を選択します。これは同時に複数の info マニュアルを閲覧するとき便利です。プレフィクス引数 C-uだけを指定した場合、C-h iはドキュメントのファイル名を尋ねるので、info メニューのトップレベルにエントリーがないファイルでも閲覧できます。

上記で説明しているヘルプコマンド C-h F *function* RETと C-h K *key*は、info を実行して関数 *(function)*またはキー *(key)*に関するドキュメントを直接開きます。

プログラムを編集しているとき、そのプログラム言語の info 版のマニュアルを持っていれば、C-h S (info-lookup-symbol) を使ってシンボル (キーワード、関数、変数) のエントリーを、適切なマニュアルから探すことができます。コマンドがどのように動作するかの詳細は、メジャーモードに依存します。

何か予期しないことが起こって、何をタイプしたかわからなくなったときは、C-h l (view-lossage) を使います。C-h lは最近のキーストロークと、それらが呼び出したコマンドを表示します。デフォルトでは、Emacs は最近の 300 回分のキーストロークを保持します。これを変更したければ、コマンド lossage-sizeで回数を変更できます。馴染みのないコマンドを見つけたら、C-h kや C-h fを使用して、それらのコマンドが何を行なうか調べることができます。

最近のエコーエリアのメッセージを調べるには、C-h e (view-echo-area-messages) を使います。これはそれらのメッセージを保持するバッファー、*Messages*を表示します。

Emacs の各メジャーモードでは、一般的にいくつかのキーが再定義されていて、編集動作も異なります。C-h m (describe-mode) は、現在のメジャーモードのドキュメントを表示します。これには通常このモードで変更されているコマンドや機能、およびそれらのキーバインドについても説明されています。

C-h b (describe-bindings) と C-h s (describe-syntax) は、現在の Emacs 環境に関する、それ以外の情報を表示します。最初は現在のマイナーモードのローカルバインディング、次に現在のメジャーモードで定義されているローカルバインディング、最後にグローバルバインディングが表示されます。C-h sは各文字の構文の説明とともに、構文テーブルの内容を表示します (Section "Syntax Tables" in *The Emacs Lisp Reference Manual* を参照してください)。

プレフィクスキーの後に C-h、?、F1をタイプすることにより、特定のプレフィクスキーにたいする、サブコマンドのリストを得ることができます (この方法が機能しないプレフィクスキーもあり、たとえば ESCを例にすると ESC C-hは実際には C-M-h (mark-defun)、ESC ?は M-? (xref-find-references) だが、ESC F1 はうまく機能する)。

最後に M-x describe-keymapは補完つきでキーマップ名の入力を求めて、そのキーマップ内のすべてのキーバインディングのリストを表示します。

7.8 ヘルプファイル

ビルトインのドキュメントとマニュアル以外にも、Emacs にはコピー条件 (copying conditions)、リリースノート (release notes)、デバッグ説明書 (instructions for debugging)、バグ報告 (reporting bugs) などのトピックを説明する、いくつかのファイルが含まれています。これらのファイルは、以下のコマンドで閲覧することができます。C-h gをのぞき、これらはすべて、C-h C-*char*という形式になっています。

C-h C-c　Emacs をコピー、再頒布する場合の規則を表示します (describe-copying)。

`C-h C-d` Emacs をデバッグするためのヘルプを表示します (`view-emacs-debugging`)。

`C-h C-e` 外部のパッケージをどこで入手するかについての情報を表示します (`view-external-packages`)。

`C-h C-f` Emacs の FAQ(frequently-answered-questions: 頻繁に答えられる質問) のリストを表示します (`view-emacs-FAQ`)。

`C-h g` GNU プロジェクトに関する情報がある、ページ (`http://www.gnu.org`) を visit します (`describe-gnu-project`)。

`C-h C-m` Emacs マニュアルの印刷されたコピーの注文に関する情報を表示します (`view-order-manuals`)。

`C-h C-n` このバージョンの新しい機能の一覧が含まれる news ファイルを表示します (`view-emacs-news`)。

`C-h C-o` Emacs および他の GNU ソフトウェアーの最新バージョンを、注文またはダウンロードする方法を表示します (`describe-distribution`)。

`C-h C-p` Emacs の既知の問題 (それにどう対処するかの提案がある場合もあります) を表示します (`view-emacs-problems`)。

`C-h C-t` Emacs の TODO リストを表示します (`view-emacs-todo`)。

`C-h C-w` GNU Emacs が完全に無保証なことにたいする、すべての詳細を表示します (`describe-no-warranty`)。

7.9 アクティブテキストのヘルプとツールチップ

Emacs では拡大解釈されるアクティブテキスト (active text: マウスのクリックや `RET`に特別な反応をするテキスト) は、しばしばヘルプテキストに関連付けられています。これには Emacs のバッファーのハイパーリンク、同様にモードラインの一部が含まれます。グラフィカルなディスプレー、同様にいくつかのテキスト端末は、マウストラッキングをサポートしており、アクティブテキストの上をマウスが通過することにより、ヘルプテキストをツールチップ *(tooltip)* で表示します。Section 18.19 [Tooltips], page 213 を参照してください。

マウストラッキングをサポートしない端末では、バッファーのアクティブテキストにポイントを移動して、`C-h .` (`display-local-help`) をタイプすれば、ヘルプテキストを表示できます。これはヘルプテキストをエコーエリアに表示します。ポイントがある場所のヘルプテキストが利用可能な場合に、常に表示させるには、変数 `help-at-pt-display-when-idle`に `t`をセットしてください。

8 マークとリージョン

Emacs は他の多くのアプリケーションと同じようにバッファーのテキストの任意の部分を選択して、そのような選択されたテキスト *(selected text)* を操作するコマンドを呼び出します。この選択されたテキストのことを、Emacs では *(region)* と呼んでいます。リージョンにたいする処理の仕方は、他のプログラムが選択されたテキストを処理する方法と非常に似ていますが重要な違いも存在します。

リージョンとはバッファーのマーク *(mark)* と現時点のポイント *(point)* の間にある部分のことです。(たとえば `C-SPC`コマンドで) どこかにマークをセットして、その後にリージョンの終端としたい場所までポイントを移動することによってリージョンを定義できます (マウスウォッシュリージョンの定義に使うことも可)。

ポイントとマークのどちらが前にあったとしても、リージョンは常にポイントとマークの間にあります。ポイントが動くたびにリージョンは変化します。

テキストのある位置にマークをセットすると、マークがアクティブ *(active)* になります。マークがアクティブのときは、そのリージョンもアクティブになります。Emacs はアクティブなリージョンの中にあるテキストを、フェイス `region`でハイライト表示します (Section 33.1.5 [Face Customization], page 503 を参照してください)。

バッファーのテキストを変更するコマンドを含む、特定の非移動系コマンド (non-motion commands) の後で、Emacs は自動的にマークを非アクティブ *(deactivates)* にし、これによりハイライトも解除されます。`C-g`をタイプすれば、いつでも明示的に非アクティブにすることができます (Section 34.1 [Quitting], page 536 を参照してください)。

操作をアクティブなリージョンのテキストに制限するコマンドが多数あります。たとえば (マッチしたテキストを置換する)`M-%`コマンドは、通常ならバッファーのアクセス可能な部分全体にたいして機能しますが、リージョンをアクティブにしたときにはそのリージョンだけにたいして機能するのです。

たとえアクティブでなくても、マークは役に立ちます。たとえばマークリングを使えば、前のマーク位置に戻ることができます。更に非アクティブなリージョンであっても効果があるコマンドもいくつかあります (たとえば *upcase-region*)。`C-x C-x`のようなコマンドを使えば、そのリージョンを再びアクティブにすることもできます。

インタラクティブなセッションではデフォルトである上記動作は、Transient Mark モード (暫定マークモード) という名で知られています。Transient Mark モードを無効にすると、Emacs は通常ではリージョンをハイライトしなくなります。Section 8.7 [Disabled Transient Mark], page 57 を参照してください。

あるバッファーでマークをセットしても、他のバッファーのマークは影響を受けません。アクティブなマークがあるバッファーに戻ったとき、マークは以前と同じ場所にあります。複数のウィンドウで同じバッファーを表示しているとき、これらのウィンドウはそれぞれのポイント位置をもっているので、リージョンも異なります。しかしこれらのウィンドウでは、マークの位置は共通です。Chapter 17 [Windows], page 186 を参照してください。通常、選択されたウィンドウのリージョンだけがハイライトされます。しかし変数`highlight-nonselected-windows`が非`nil`の場合、各ウィンドウのリージョンがハイライトされます。

rectangular region(矩形リージョン) という、違う種類のリージョンもあります。Section 9.5 [Rectangles], page 68 を参照してください。

8.1 マークのセット

マークをセットするためのコマンドがいくつかあります:

`C-SPC`　　ポイント位置にマークをセットしてアクティブにします (`set-mark-command`)。

C-@ 同じです。

C-x C-x ポイント位置にマークをセットしてアクティブにしてから、以前のマークがあった位置にポイントを移動します (exchange-point-and-mark)。

Drag-mouse-1
ドラッグしたテキストの周りにポイントとマークをセットします。

mouse-3 ポイント位置にマークをセットしてから、クリックした場所にポイントを移動します (mouse-save-then-kill)。

シフトを押したカーソル移動キー
マークが非アクティブなら、ポイント位置にマークをセットしてポイントを移動します。Section 8.6 [Shift Selection], page 57 を参照してください。

マークをセットするもっとも一般的な方法は、C-SPC (set-mark-command) です [1]。これはポイントがある位置にマークをセットしてから、アクティブにします。その後、マークをそこに残したままポイントを移動できます。

たとえばバッファーの一部を大文字に変換したいとします。これを行うには対象のテキストの一方の端に移動して、C-SPCをタイプし、対象のテキストがハイライトされるまでポイントを移動します。そして C-x C-u (upcase-region) をタイプすると、リージョンのテキストが大文字に変換されて、マークが非アクティブになります。

マークがアクティブなときに非アクティブにしたいときは、C-gをタイプします (Section 34.1 [Quitting], page 536 を参照してください)。リージョンにたいして操作を行うほとんどのコマンドは、上記の例の C-x C-uのように、自動的にマークを非アクティブにします。

リージョンにたいしての操作は行わず、バッファーの位置を覚えておくためにマークをセット (C-SPC C-SPCとタイプ) して、後でそこに戻る (C-u C-SPCとタイプ) こともできます。詳細については、Section 8.4 [Mark Ring], page 56 を参照してください。

コマンド C-x C-x (exchange-point-and-mark) は、ポイントとマークの位置を交換します。ポイントの位置に問題はないが、リージョンのもう一方の端にポイントを移動したいとき C-x C-xは便利です。2 回目の C-x C-xで、マークを新しいポイント位置にマークをセットしてから、ポイントを元の位置に戻すことができます。このコマンドはマークが非アクティブのとき、最初にマークをアクティブにします。これはマークが最後にどこにセットされたかを明確にするために、リージョンをハイライトするためです。しかしプレフィクス引数とともに呼び出せば、マークは非アクティブのままでリージョンもハイライトされません。これを使えば C-u C-SPCと同様の方法で、マークの位置にジャンプできます。

マウスでマークをセットすることもできます。マウスの左ボタン (down-mouse-1) をクリックしてから、テキスト範囲をドラッグすると、最初にマウスボタンを押した位置にマークがセットされ、マウスボタンを話した位置にポイントが置かれます。かわりにマウスの右ボタン (mouse-3) をクリックすれば、ポイントのある位置にマークがセットされ、クリックした位置にポイントが移動します。これらのマウスコマンドに関する詳細な説明は、Section 18.1 [Mouse Commands], page 194 を参照してください。

最後にシフトキーを押しながらカーソルを移動するコマンド (S-RIGHT、S-C-f、S-C-nなど) でマークをセットできます。これはシフト選択 *(shift-selection)* と呼ばれ、以前にシフト選択やマウスコマンドでセットされたアクティブなマークがないときに限り、ポイントを移動する前の位置にマー

[1] ASCIIには、文字 C-SPCはありません。テキスト端末で C-SPCをタイプすると、通常は文字 C-@が与えられます。このキーも set-mark-commandにバインドされているので、もし異なる挙動を示すテキスト端末の場合は、C-SPCのかわりに C-@を使うことを考えるのがよいかもしれません。

クをセットします。マウスコマンドやシフト選択によるマークのセットは、通常のマークとは少し異なります。続けてシフトを押さないカーソル移動コマンドを実行するにより、マークは自動的に非アクティブになります。詳細は、Section 8.6 [Shift Selection], page 57 を参照してください。

`C-y` (`yank`) のようなテキストを挿入するコマンドの多くは、挿入されたテキストの先頭に、非アクティブなマークをセットします。これにより簡単にその位置に戻ることができます (Section 8.4 [Mark Ring], page 56 を参照してください)。コマンドがこれを行っていることは、エコーエリアに'`Mark set`'が表示されることで見分けることができます。

X ではアクティブなリージョンが変化するたびに、Emacs はリージョンのテキストをプライマリー選択 *(primary selection)* に保存します。これにより `mouse-2`をクリックして、他の X アプリケーションへテキストを挿入することができるようになります。Section 9.3.2 [Primary Selection], page 66 を参照してください。

8.2 テキストオブジェクトをマークするコマンド

単語 (word)、リスト (list)、パラグラフ (paragraph: 段落)、ページ (page) などのテキストオブジェクトの周辺に、ポイントを配置してマークするコマンドがあります:

`M-@` 次の単語の末尾にマークをセットします (`mark-word`)。ポイントは移動しません。

`C-M-@` 次の対応のとれた式の後にマークをセットします (`mark-sexp`)。ポイントは移動しません。

`M-h` ポイントを現在のパラグラフの先頭に移動して、パラグラフの最後にマークをセットします (`mark-paragraph`)。

`C-M-h` ポイントを現在の defun の先頭に移動して、defun の最後にマークをセットします (`mark-defun`)。

`C-x C-p` ポイントを現在のページの先頭に移動して、ページの最後にマークをセットします (`mark-page`)。

`C-x h` ポイントを現在のバッファーの先頭に移動して、バッファーの最後にマークをセットします (`mark-whole-buffer`)。

`M-@` (`mark-word`) は、次の単語の最後にマークをセットします (単語についての情報は、Section 22.1 [Words], page 251 を参照してください)。繰り返し呼び出されると、マークを 1 度に 1 単語進めてリージョンを拡張します。例外として、マークがアクティブでポイントの前にある場合、`M-@` はマークを現在の位置から 1 単語後方に移動します。

このコマンドに数引数 *n*を指定することにより、*n*単語進めてマークするよう指定できます。負の引数 $-n$は、*n*単語後方にマークを移動します。

同様に `C-M-@` (`mark-sexp`) は、対応のとれた式の最後にマークをセットします (Section 23.4.1 [Expressions], page 292 を参照してください)。繰り返し呼び出すことにより、後続の式にリージョンを拡張します。正または負の数引数を指定するとその数に応じて前方または後方にマークを移動します。

上記のリストの他のコマンドは、ポイントとマークの両方をセットするので、バッファー内のオブジェクトを区切るコマンドです。`M-h` (`mark-paragraph`) はパラグラフ (Section 22.3 [Paragraphs], page 253 を参照してください)、`C-M-h` (`mark-defun`) はトップレベルの関数定義 (Section 23.2.2 [Moving by Defuns], page 286 を参照してください)、`C-x C-p` (`mark-page`) はページ (Section 22.4 [Pages], page 254 を参照してください) をマークします。繰り返して呼び出すと、

同種の連続するオブジェクトへと、リージョンを拡張します。数引数も同様で、マークを移動したいオブジェクトの数を指定します。

`C-x h` (`mark-whole-buffer`) はポイントをバッファーの先頭、マークを最後にセットすることによりバッファー全体をリージョンとします。

8.3 リージョンを操作する

一度リージョンを設定すると、それを処理するいくつかの方法があります:

- `C-w` (Chapter 9 [Killing], page 59 を参照) で kill します。
- `M-w` で kill リングにコピーします (Section 9.2 [Yanking], page 62 を参照してください)。
- `C-x C-l`または `C-x C-u`で、大文字小文字を変換します (Section 22.7 [Case], page 260 を参照してください)。
- `C-u C-/`で変更をアンドゥ(undo) します (Section 13.1 [Undo], page 131 を参照してください)。
- `M-%`でリージョンの中のテキストを置換します (Section 12.10.4 [Query Replace], page 124 を参照してください)。
- `C-x TAB`または `C-M-\`でインデントします (Chapter 21 [Indentation], page 247 を参照してください)。
- `M-x fill-region`でテキストとしてフィルします (Section 22.6 [Filling], page 256 を参照してください)。
- `M-$`で単語のスペルをチェックします (Section 13.4 [Spelling], page 134 を参照してください)。
- `M-x eval-region`で Lisp コードとして評価します (Section 24.9 [Lisp Eval], page 329 を参照してください)。
- `C-x r s`でレジスターに保存します (Chapter 10 [Registers], page 72 を参照してください)。
- バッファーまたはファイルに保存します (Section 9.4 [Accumulating Text], page 67 を参照してください)。

マークが非アクティブのときにはデフォルトの動作をするが、マークがアクティブのときはリージョンを処理するコマンドがいくつかあります。たとえば `M-$` (`ispell-word`) は、通常はポイントのある単語のスペルをチェックしますが、マークがアクティブのときはリージョンの中のテキストをチェックします (Section 13.4 [Spelling], page 134 を参照してください)。通常そのようなコマンドはリージョンが空のとき (たとえばマークとポイントが同じ位置のとき) は、デフォルトの動作をします。空のリージョンにたいして処理を行いたいときは、変数 `use-empty-active-region`を `t`に変更してください。

Section 4.3 [Erasing], page 21 で説明したように、`DEL` (`backward-delete-char`) と `Delete` (`delete-forward-char`) もこの方法で動作します。マークがアクティブのときはリージョンのテキストを削除します (例外として数引数 *n*に 1 以外が指定されたとき、これらのコマンドはマークがアクティブか関係なく、*n*文字を削除します)。変数 `delete-active-region`を `nil`に変更すると、これらのコマンドはマークがアクティブのとき異なる動作をしなくなります。これを `kill`に変更するとリージョンを削除するかわりに、*kill* するようになります (Chapter 9 [Killing], page 59 を参照してください)。

その他のコマンドにはデフォルトの動作はなく、常にリージョンを処理します。通常このようなコマンドには、`C-w` (`kill-region`) や `C-x C-u` (`upcase-region`) のように、名前に `region`がついています。マークが非アクティブのときは非アクティブなリージョン、すなわちポイントと最後にマークをセットした位置の間にあるテキストにたいして処理を行います (Section 8.4 [Mark Ring], page 56 を参照してください)。この動作を無効にするには、変数 `mark-even-if-inactive`を `nil`

に変更してください。そうするとこれらのコマンドはマークが非アクティブのときエラーをシグナルします。

デフォルトでは、たとえマークがアクティブであっても通常のテキスト挿入のように、たとえばaをタイプすれば文字aが挿入されて、その後にマークが非アクティブになります。Delete Selectionというマイナーモードでは、この挙動を変更できます。このモードが有効な場合には、マークがアクティブなときにテキストを挿入すると、まずリージョン内のテキストが最初に削除されるのです。ただしこの挙動は`delete-selection-temporary-region`オプションをカスタマイズして調整できます。このオプションのデフォルト値は`nil`ですが`t`にセットすることもできます。その場合にはマウス (Section 8.1 [Setting Mark], page 52 を参照)、シフト選択 (Section 8.6 [Shift Selection], page 57 を参照)、あるいはTransient Markモード無効時の`C-u C-x C-x`によってセットされた一時的にアクティブなリージョンだけは置き換えられるようになります。`delete-selection-temporary-region`に`selection`をセットすることによって、挙動を更に調整できます。この場合には`C-u C-x C-x`によってセットされたリージョンは置き換えられずに、マウスのドラッグやシフト選択によってアクティブになったリージョンだけが置き換えられるのです。Delete Selectionのオンとオフを切り替えるには`M-x delete-selection-mode`をタイプしてください。

8.4 マークリング

各バッファーは、マークリング *(mark ring)* の中に、以前のマークの位置を記録しています。マークをセットするコマンドは、古いマークをこのリングにpushします。マークリングの1つの使い方として、後で戻りたい場所を記録させる使い方があります。

`C-SPC C-SPC`
: マークをアクティブにせずにマークをセットしてから、マークリングにpushします。

`C-u C-SPC`
: マークがあった場所にポイントを移動し、マークリングから1つ前のマークを復元します。

コマンド`C-SPC C-SPC`は、後で戻ってきたい位置をマークするときに便利です。これは現在の位置をマークを (Emacsがリージョンをハイライト表示してしまう) アクティブにすることなく、マークリングにpushします。実際にこれは`C-SPC` (`set-mark-command`) を連続して2回呼び出しています。最初の`C-SPC`はマークをセットし、2回目の`C-SPC`はそれを非アクティブにしています (Transient Markモードがオフの場合、`C-SPC C-SPC`は一時的にTransient Markモードを有効にします。Section 8.7 [Disabled Transient Mark], page 57 を参照してください)。

マークした位置に戻るには、`C-u C-SPC`のようにプレフィクス引数を指定して、`set-mark-command`を使います。これはマークがあった場所にポイントを移動して、もしマークがアクティブのときは非アクティブにします。`C-u C-SPC`を連続して呼び出すと、マークリングに保存された前の位置へジャンプしていきます。この方法で移動した位置の情報は失われません。それらはリングの最後に移動します。

`set-mark-command-repeat-pop`を非`nil`にセットすると、`C-u C-SPC`の後に続けて、`C-u C-SPC`ではなく、`C-SPC`でマークリングを巡回できます。デフォルトでは`set-mark-command-repeat-pop`は`nil`です。

各バッファーは自身のマークリングを持ちます。すべての編集コマンドは現在のバッファーのマークリングを使います。特に`C-u C-SPC`は常に同じバッファーに留まります。

変数`mark-ring-max`は、マークリングに保持する最大のエントリー数を指定します。デフォルトは16エントリーです。もしエントリー数が最大の場合、他のエントリーをpushするとリストの一番古いものが捨てられます。`C-u C-SPC`を繰り返すと、リングの現在位置を巡回します。

もし何度も同じ場所に戻りたいときは、マークリングでは不十分でしょう。そのような場合は後で使うために、その位置をレジスターに記録できます (Section 10.1 [Position Registers], page 72 を参照してください)。

8.5 グローバルマークリング

各バッファーに属する普通のマークリングに加えて、Emacs にはグローバルマークリング *(global mark ring)* が 1 つあります。以前マークをセットしてからバッファーを切り替えた場合、マークをセットすると、マークはカレントバッファーのマークリングに加えて、グローバルマークリングにも記録されます。その結果、グローバルマークリングには訪れていたバッファーの系列が記録され、各バッファーではマークを設定した箇所が記録されます。グローバルマークリングの長さは、`global-mark-ring-max` で制御され、デフォルトは 16 です。

コマンド `C-x C-SPC` (`pop-global-mark`) は、グローバルリングの最新のバッファー位置にジャンプします。これもリングを巡回するので、連続して `C-x C-SPC`を使うことにより、古いバッファーのマーク位置に移動します。

8.6 シフト選択

シフトキーを押しながらカーソル移動コマンドをタイプすると、ポイントを移動する前の位置にマークをセットするので、リージョンが元のポイント位置から新しいポイント位置に拡張されます。この機能はシフト選択 *(shift-selection)* と呼ばれます。これは他のエディターでテキストを選択する方法と似ています。

シフト選択によるマークのセットは、これまでの説明とは少し異なる振る舞いをします。最初に、マークを非アクティブにする通常の方法 (バッファーのテキストを変更したり `C-g`をタイプするなど) に加え、シフトキーを押さない *(unshifted)* カーソル移動コマンドでも、マークが非アクティブになります。次に、連続するシフトキーを押した *(shifted)* カーソル移動コマンドでは、マークの更新はされません。つまりシフトキーを押しながらカーソル移動コマンドを繰り返すと、リージョンは継続的に変更されます。

シフト選択は、シフトキーを押したカーソル移動キーが、別のコマンドにバインドされていない場合のみ動作します (Chapter 33 [Customization], page 499 を参照してください)。たとえば `S-C-f` を他のコマンドにバインドしていると、`S-C-f`はシフト選択バージョンの `C-f` (`forward-char`) ではなく、バインドされたコマンドを実行します。

マウスコマンドによるマークのセットも、シフト選択によるマークのセットと同様です (Section 8.1 [Setting Mark], page 52 を参照してください)。たとえばマウスをドラッグしてリージョンを指定すると、シフトキーを押したカーソル移動コマンドを使って、そのリージョンの拡張を続けることができます。どちらのケースも、シフトキーを押さないカーソル移動コマンドで、マークが非アクティブになります。

シフト選択をオフにするには、`shift-select-mode`を `nil`にセットします。これを行ってもマウスコマンドを通じたマークのセットは無効になりません。`shift-select-mode`に値 `permanent`をセットするとシフト変換されていないカーソル移動キーによるマークの非アクティブ化が行われなくなり、たとえば前のコマンドでセットされたリージョンはシフト選択で拡張でき、シフト選択でセットされたリージョンはシフトキーを押していないカーソルキーで拡張されることになります。

8.7 Transient Mark モードを無効にする

マークとリージョンのデフォルト動作では、マークをアクティブにセットすると、リージョンがハイライトされます。これは Transient Mark モードと呼ばれます。これはインタラクティブなセッショ

ンではデフォルトで有効になっているマイナーモードです。`M-x transient-mark-mode`、または‘`Options`’メニューの‘`Highlight Active Region`’で切り替えることができます。オフにすることにより Emacs の操作モードは変更されます。

- `C-SPC`や`C-x C-x`のようなコマンドでマークをセットしても、リージョンはハイライトされません。そのためマークがどこにあるか見分けることができないので、覚えている必要があります。

  マークをセットしたらどこにセットしたか忘れる前にすぐ使うというのが、この問題にたいする通常の解決策です。ポイントとマークの位置を交換する`C-x C-x`で、マークがどこかチェックすることもできます。
- 通常マークがアクティブのときリージョンにたいして処理を行ういくつかのコマンドは、そのような振る舞いをしなくなります。たとえば普通`M-%` (`query-replace`) は、マークがアクティブのときはリージョンにたいして置換を行います。Transient Mark がオフだと、常にポイントからバッファーの最後までを処理します。このような方法で動作するコマンドは、コマンド自身のドキュメントにより識別できます。

Transient Mark モードがオフのときは、`C-SPC C-SPC`または`C-u C-x C-x`を使って一時的にアクティブにすることができます。

`C-SPC C-SPC`
: ポイント位置にマークをセット (普通の`C-SPC`と同様) して、マークが非アクティブになるまでの間、1 度だけ Transient Mark モードを有効にします (実際にはこれは独立したコマンドではなく`C-SPC`コマンドを 2 回行っています)。

`C-u C-x C-x`
: ポイントとマークを交換してからマークをアクティブにして、次にマークが非アクティブになるまでの間、Transient Mark モードを一時的に有効にします (これはプレフィクス引数を指定した`C-x C-x` (`exchange-point-and-mark`) コマンドです)。

これらのコマンドはマークをセットまたはアクティブにして、マークが非アクティブになるまでの間 Transient Mark モードを有効にします。これらを使う 1 つの理由は、いくつかのコマンドは Transient Mark モードがオフのとき、リージョンにたいしてではなくバッファー全体を処理するからです。Transient Mark モードを一時的に有効にできれば、これらのコマンドをリージョンにたいして処理させることができます。

リージョンをマウス (Section 8.1 [Setting Mark], page 52 を参照してください)、またはシフト選択 (Section 8.6 [Shift Selection], page 57 を参照してください) で指定したときも、一時的に Transient Mark モードが有効になり、リージョンがハイライトされます。

9 テキストの kill と移動

Emacs で *kill* とはテキストを消去して、*kill* リングにコピーすることを意味します。*yank* とは、kill リングからテキストを取り出して、バッファーに戻すことを意味します (“cut(カット、切り取り)” と “paste(ペースト、貼り付け)” という用語を使うアプリケーションもあります)。これはテキストブロックのセットが、循環的にアクセスできるリングに格納されているイメージから、kill リングと名付けられました。Section 9.2.1 [Kill Ring], page 62 を参照してください。

kill と yank は、Emacs でテキストを移動したりコピーするための、もっとも一般的な方法です。これは用途が広いコマンドです。なぜなら、多くの異なる種類の構文単位を kill するためのコマンドが存在するからです。

9.1 削除と kill

バッファーからテキストを消去するコマンドの多くは、それを kill リング (Section 9.2.1 [Kill Ring], page 62 を参照) に保存します。これらは *kill* コマンドとして知られており、通常名前に ‘`kill`’が含まれます (例 `kill-line`)。kill リングには、最近 kill したものが、1 つだけではなくいくつか格納されているので、kill はとても安全な操作と言えます。なぜなら、以前に kill したテキストが失われる心配をする必要がないからです。kill リングは、すべてのバッファーで共有されているので、あるバッファーで kill したテキストを、別のバッファーに yank することができます。

`C-/` (`undo`) を使うと、kill コマンドはアンドゥ(Section 13.1 [Undo], page 131 を参照してください) されるので、kill したテキストはバッファーに戻されますが、kill リングからは削除されません。

グラフィカルなディスプレイでは、テキストを kill すると、それはシステムのクリップボードにもコピーされます。 Section 9.3 [Cut and Paste], page 64 を参照してください。

テキストを消去して kill リングに保存しないコマンドは、削除 *(delete)* コマンドとして知られており、名前に ‘`delete`’が含まれています。これらは `C-d` (`delete-char`) や `DEL` (`delete-backward-char`) のように、一度に 1 文字削除するものや、スペースや改行だけを削除するものが含まれます。重要なデータの有意な量を消去するコマンドには、一般的に kill 操作が用いられます。

kill と yank でマウスを使うこともできます。Section 9.3 [Cut and Paste], page 64 を参照してください。

9.1.1 削除

削除とは、テキストを消去して kill リングに保存しないという意味です。テキストを削除するたいていの Emacs コマンドは、1 文字または空白文字しか消去しません。

`DEL`
`BACKSPACE`
: 前の文字を削除します。リージョンがアクティブのときは、リージョンのテキストを削除します (`delete-backward-char`)。

`Delete`
: 次の文字を削除します。リージョンがアクティブのときは、リージョンのテキストを削除します (`delete-forward-char`)。

`C-d`
: 次の文字を削除します (`delete-char`)。

`M-\`
: ポイントの周囲のスペースとタブを削除します (`delete-horizontal-space`)。

`M-SPC`
: スペースを 1 つ残して、ポイントの周囲のスペースとタブを削除します (`just-one-space`)。

C-x C-o　現在行の周囲の空行を削除します (delete-blank-lines)。

M-^　行間にある改行をインデントと共に削除して 2 行を 1 行にします (delete-indentation)。

基本的な削除コマンド DEL (delete-backward-char)、delete (delete-forward-char)、C-d (delete-char) については既に説明しました。Section 4.3 [Erasing], page 21 を参照してください。数引数を指定すると、指定した数の文字を削除します。リージョンがアクティブのとき、数引数に 1 を指定するか省略した場合、DELと deleteは、リージョンのすべてのテキストを削除します。

他の削除コマンドはスペース、タブ、改行といった空白文字だけを削除します。M-\ (delete-horizontal-space) はポイント前後にあるすべてのスペースとタブ文字を削除します。プレフィックス引数を指定した場合には、ポイントの前にあるスペースとタブ文字だけを削除します。

just-one-spaceも同様なことを行いますが、前にあったスペースの数に関わらず (たとえ 1 つも存在しなくても)、ポイントの前にスペースを 1 つ残します。数引数 nを指定した場合には、nが正ならポイントの前に n個のスペースを残し、nが負ならポイントの前に $-n$個のスペースを残すとともに、スペースとタブに加えて改行も削除します。

just-one-spaceのより柔軟なバージョンのように動作するコマンドが cycle-spacing (M-SPC) です。このコマンドは繰り返し連続で呼び出すと、cycle-spacing-actionsに定義された異なるクリーンアップ動作を循環的に実行します。

C-x C-o (delete-blank-lines) は、現在行の下にあるすべての空行を削除します。現在行が空行のときは、現在行の上にあるすべての空行も削除します (空行を 1 つ、つまり現在行は残します)。単独の空行で実行するとその行を削除します。

M-^ (delete-indentation) は、改行と周囲のスペース (通常 1 つのスペースを残す) を削除することにより、現在行とその上の行を結合します。Chapter 21 [Indentation], page 247 を参照してください。

コマンド delete-duplicate-linesは、リージョン内の重複した行を検索して、それぞれ 1 行を残して削除します。通常は重複した行の最初の行を残しますが、プレフィクス引数 C-uを指定すると、最後の行を残します。プレフィクス引数 C-u C-uを指定すると、隣接した重複行だけを検索します。これは行がソート済みのとき効果的です。プレフィクス引数 C-u C-u C-uを指定すると、連続する空行は残します。

9.1.2 行の kill

C-k　行の残り、または 1 行以上を kill します (kill-line)。

C-S-backspace
　1 度に行全体を削除します (kill-whole-line)。

もっとも簡単な kill コマンドは、C-k (kill-line) です。これを行末で使うと、その行を終端している改行を kill して、現在行と次の行を継げます (空行なら削除します)。そうでない場合、C-kはポイントから行末までを削除します。ポイントの元の位置が行頭の場合は、空行が残ります。

どちらのケースを適用するか決める際には、行末のスペースとタブは無視されます。ポイントが行の一番最後の非空白文字の後にあるとき、C-kは改行を kill することに注意してください。空でない行全体を kill するときは、行頭で C-kを 2 回タイプしてください。

このコンテキストで “行” とは、スクリーン行ではなく論理行を意味します (Section 4.8 [Continuation Lines], page 23 を参照してください)。

C-kに正の数値 nを与えると、n行とそれに続く改行を kill します (現在行のポイントの前にあるテキストは kill されません)。負の引数 $-n$を与えると、現在行のポイントの前にあるテキストと、前の n行を kill します。C-kに 0 を指定すると、現在行のポイントの前にあるテキストを kill します。

変数 kill-whole-lineが非 nilのときは、行頭での C-kにより行末の改行も含めて行全体が kill されます。この変数は通常 nilです。

C-S-backspace (kill-whole-line) は行中のポイントの位置に関わらず、改行を含めた行全体を kill します。キーシーケンス C-S-backspaceをタイプできないテキスト端末がたくさんあることに注意してください。

9.1.3 その他の kill コマンド

C-w リージョンを kill します (kill-region)。

M-w リージョンを kill リングにコピーします (kill-ring-save)。

M-d 次の単語を kill します (kill-word)。Section 22.1 [Words], page 251 を参照してください。

M-DEL 後方に 1 単語 kill します (backward-kill-word)。

C-x DEL センテンスの先頭までを後方に kill します (backward-kill-sentence)。Section 22.2 [Sentences], page 252 を参照してください。

M-k 文の末尾までを kill します (kill-sentence)。

C-M-k 後に続く対応のとれた式 (balanced expressions) を kill します (kill-sexp)。Section 23.4.1 [Expressions], page 292 を参照してください。

M-z *char* 次の *char*までを kill します (zap-to-char)。

M-x zap-up-to-char *char*
次の *char*まで (*char*を含まず) を kill します。

一般によく使われる kill コマンドは C-w (kill-region) で、これはリージョンのテキストを kill します (Chapter 8 [Mark], page 52 を参照してください)。同様に M-w (kill-ring-save) は、バッファーからテキストを消去せずに、リージョンのテキストを kill リングにコピーします。C-wまたは M-wをタイプしたとき、マークが非アクティブの場合、これらのコマンドはポイントと最後にセットしたマークの間にあるテキストにたいして処理を行います (Section 8.3 [Using Region], page 55 を参照してください)。

Emacs は特定の構文単位にたいする kill コマンドを提供します。単語 (words) にたいしてはM-DELと M-d(Section 22.1 [Words], page 251 を参照してください)、対応のとれた式 (balanced expressions) にたいしては C-M-k(Section 23.4.1 [Expressions], page 292 を参照してください)、センテンス (sentences: 文) にたいしては C-x DELM-k(Section 22.2 [Sentences], page 252 を参照してください) です。

コマンド M-z (zap-to-char) は kill と検索の組み合わせです。これは文字を読み取り、ポイントからバッファー内の次にその文字が現れる場所までを kill します。数引数は繰り返し回数です。負の引数の場合は後方に検索することを意味し、ポイントの前のテキストを kill します。以前に使用した文字のヒストリーが保守されていて M-pと M-nのキーストロークでアクセスできます。これは主に複雑な入力手段で入力する必要が文字の場合に有用です。類似コマンドの zap-up-to-charはポイントから次の文字まで kill しますがその文字は含みません。数引数は繰り返し回数として機能します。

9.1.4 kill のオプション

いくつかの特別なバッファーは、読み取り専用 *(read-only)* のテキストを含んでいて、それらは変更できないので kill もできません。kill コマンドは読み取り専用のバッファーにたいして特別な動作をします。バッファーから実際にテキストを削除せずに、kill リングにコピーします。通常はビープ音を

ならし、その旨のエラーメッセージを表示します。しかし変数 kill-read-only-okを非 nilにセットすると、なぜテキストが消去されないのかをエコーエリアにメッセージ表示します。

kill した文字列を kill リングに保存する前に、kill-transform-functionを使用してその文字列を変換できます。これは kill された文字列で呼び出されて、kill リングに保存したい文字列をリターンする必要があります。nilもリターンでき、その場合には文字列は kill リングに保存されません。たとえば空白のみの文字列を kill リングに保存したくなければ、以下のようにできます:

```
(setq kill-transform-function
      (lambda (string)
        (and (not (string-blank-p string))
             string)))
```

変数 kill-do-not-save-duplicatesを非 nilに変更すると、同じものにたいする kill は重複なく kill リングの 1 つのエントリーとなります。

9.2 yank

yank するとは、以前 kill したテキストを再び挿入するという意味です。テキストを移動またはコピーする通常の方法は、それを kill してからどこかに yank する方法です。

C-y 最後に kill したものをポイント位置に yank します (yank)。

M-y yank したテキストを、それより前に kill したテキストに置き換える、または以前に kill した一連のテキストのリストから選択できるようにします (yank-pop)。Section 9.2.2 [Earlier Kills], page 63 を参照してください。

C-M-w 次のコマンドが kill コマンドのときは、kill したものを、以前に kill したものに追加します (append-next-kill)。Section 9.2.3 [Appending Kills], page 64 を参照してください。

基本的な yank コマンドは、C-y (yank) です。これはもっとも最近 kill されたものを挿入し、カーソルを挿入されたテキストの最後に移動します。また挿入されたテキストの先頭にマークをセットして、それを非アクティブにします。これにより C-u C-SPCで簡単にその位置にジャンプできます (Section 8.4 [Mark Ring], page 56 を参照してください)。

C-u C-yのようにプレフィクス引数を指定すると、カーソルを挿入されたテキストの前に移動して、マークをテキストの最後にセットします。他のプレフィクス引数は、何回前の kill かを指定します。たとえば C-u 4 C-yは、もっとも最近 kill されたものから 4 番目に古いものを挿入します。

グラフィカルなディスプレイや多機能なテキストモードディスプレイでは、C-yはまず最後に Emacs が kill した後に、他のアプリケーションがシステムのクリップボードに、何らかのテキストをコピーしていないか調べます。もしコピーしていたなら、かわりにクリップボードのテキストを挿入します。このように Emacs は効果的に “カット (cut)” や “コピー (copy)” などの、他のアプリケーションで処理されたクリップボード操作を、Emacs の kill のように扱います (ただし kill リングには記録されません)。詳細については、Section 9.3 [Cut and Paste], page 64 を参照してください。

9.2.1 kill リング

kill リング (kill ring) とは、以前に kill されたテキストブロックからなるリストです。すべてのバッファーにたいして、kill リングは 1 つしかないので、あるバッファーで kill したテキストを、他のバッファーに yank することができます。これはバッファーから他のバッファーへテキストを移動する、通常の方法です (他の方法もいくつかあります。たとえばテキストをレジスターに格納することもできます。Chapter 10 [Registers], page 72 を参照してください。テキストを移動する他の方法については、Section 9.4 [Accumulating Text], page 67 を参照してください)。

kill リングのエントリーの最大数は、変数 kill-ring-maxで制御されます。デフォルトは 120 です。エントリー数が制限に達しているとき新たに kill すると、Emacs は kill リングの一番古いエントリーを削除して空きを作ります。

kill リングの実際の内容は、kill-ringという名前の変数に格納されています。kill リングのエントリーの内容は、C-h v kill-ringで見ることができます。

9.2.2 過去に kill したテキストを yank する

Section 9.2 [Yanking], page 62 で説明したように、C-yに数引数を指定して、最後に kill されたものではないテキストを yank できます。これは kill リングのどのエントリーが欲しいか覚えているとき便利です。もし覚えていないときは、M-y (yank-pop) コマンドを使って候補を巡回したり、もっと前に kill したものを選択できます。

もし前のコマンドが yank コマンドのとき、M-yは yank されたテキストを、1 つ前に kill されたテキストで置き換えます。つまり 2 番目に新しい kill されたテキストを復元するには、最初に C-yで最後に kill されたテキストを yank し、次に M-yでその 1 回前に kill されたテキストで置き換えます。これはは C-yまたは他の M-yの後でのみ機能します (別のコマンドの後に M-yを呼び出すと動作が異なる。以下参照)。

kill リングのエントリーを指す (ポイントする)、last yank ポインターという概念で、この M-yの操作モードを理解できるでしょう。なにかを kill する度に、last yank ポインターはリングの先頭に新たに作られたエントリーを指すように移動します。C-yは、last yank ポインターが指すエントリーを yank します。M-yは last yank ポインターが違うエントリーを指すように移動して、バッファーのテキストをポインターが指すテキストに変更します。C-yや別の M-yの後の M-yにより、last yank ポインターを前のエントリーに移動するとともにバッファー内のテキストはそのマッチに変更されます。M-yコマンドを充分な回数繰り返せばリング内の任意のエントリーにポインターを移動できるので、任意のエントリーをバッファーに取り込むことができます。やがてリングの最後に到達すると、次の M-yにより再び最初のエントリーに戻ります。

M-yはリング内で last yank ポインターを移動させますが、リング内のエントリーの順番は変更しません。リングのエントリーは、常に最後に kill されたものを先頭に、記憶されているもので一番古いエントリーへと並んでいます。

C-yか M-yを使用した後には、数引数によって last-yank ポインターを何回進めるかを M-yに指定できます。負の引数はリングの先頭に向かってポインターを移動させます。リングの先頭では一番古いエントリーに戻り、そこから先頭へと移動します。

望むテキストを見つけてバッファーに取り込んだら、M-yコマンドを止めれば、最後に yank されたテキストはそこに残ります。このテキストは kill リングのエントリーの単なるコピーなので、それを編集してもリングの中のエントリーは変更されません。新しく何かを kill しない限り、last-yank ポインターは同じ位置に留まるので、C-yでそのテキストの別のコピーを yank できます。

C-yに数引数を指定するときも、yank するエントリーに last-yank ポインターをセットします。

yank 以外のコマンドの後に M-yを呼び出すこともできます。この場合には、M-yはミニバッファーで以前の kill のいずれかの入力を求めます。再挿入したいエントリーが見つかるまで、ミニバッファーのヒストリーコマンド (Section 5.5 [Minibuffer History], page 36 を参照) を使用して、kill リング内のエントリーの操作や検索ができます。あるいは kill リング内のエントリーリストのエントリーにたいして補完を行うための補完コマンド (Section 5.4.2 [Completion Commands], page 31 を参照) や、選択可能な候補エントリーによる*Completions*バッファーをポップアップできます。オプションとして kill エントリーを選択後に、ミニバッファーでそれを編集できます。最後に RETでミニバッファーを exit して、選択した kill リングエントリーのテキストを挿入します。別の yank コマンドの後の M-yのように、last-yank は yank したばかりのテキストがあった位置を指したままになり

ます。これは前の kill のいずれか、または挿入前に編集した kill リングのエントリーのいずれかです (後者の場合には kill リング先頭に編集したテキストが追加される)。そのためこの場合にも C-yにより挿入したばかりのテキストの別のコピーが yank されます。

yank 以外のコマンドの後に単なるプレフィックス引数と共に呼び出した際 (C-u M-y) には、M-y は C-yが行うように挿入されたテキストの前にカーソルを残します。

9.2.3 kill したテキストの追加

通常は kill コマンドを実行するごとに、新しいエントリーが kill リングに push されます。しかし 2 回以上の連続する kill コマンドにより、kill されたテキストを 1 つのエントリーとしてまとめ、すべてのテキストを 1 単位として、あたかもそれが kill されたかのように、1 回の C-yで yank できます。

つまりテキストを 1 つの単位として yank したいとき、そのテキストすべてを 1 回で kill する必要はありません。すべてが kill されるまで行から行、単語から単語へと kill を続け、それを一度に取得することができます。

ポイントから前方に kill するコマンドは、直前に kill されたテキストの最後に追加します。ポイントから後方に kill するコマンドは、テキストの先頭に追加します。この方法により前方と後方を併用した連続する kill コマンドは、すべての kill されたテキストを再配置しなくてもよいように、1 つのエントリーにまとめます。数引数を指定しても kill の連続性は途切れません。たとえば以下のようなテキストを含むバッファーがあるとしましょう:

```
This is a line ⋆of sample text.
```

ポイントの位置は ⋆ で示された場所です。M-d M-DEL M-d M-DELとタイプして、前方と後方への kill を交互に行うと、最後に kill リングには ‘a line of sample’という 1 つのエントリー、バッファーには ‘This is text.’ が残ります (‘is’と ‘text’の間には 2 つのスペースがあることに注意してください。これは M-SPCまたは M-qで取り除くことができます)。

同じテキストを kill する別の方法は、M-b M-bで単語 2 つ後方に移動してから、C-u M-dで前方の単語 4 つを kill する方法です。これはバッファーと kill リングに、正確に同じ結果をもたらします。M-f M-f C-u M-DELで後方に kill しても結果は同じです。kill リングのエントリーは、常にバッファーから kill される前と同じ順番になります。

kill コマンドと最後のキルコマンドの間に、(単なる数引数ではない) 他のコマンドが入ると、kill リングには新たなエントリーが作られます。しかし、あらかじめ C-M-w (append-next-kill) とタイプすることにより、最後に kill されたテキストに追加するように強制できます。C-M-wは、後に続くコマンドが kill コマンドであれば、kill したテキストを前に kill したテキストに付け加えるよう指示します。この方法でも、前方に kill するコマンドの場合は、前に kill されたテキストの最後に追加され、後方に kill するコマンドの場合は、先頭に追加されます。この方法により、1ヶ所に yank するために離れた場所にあるいくつかのテキスト断片を、kill して集めることができます。

M-w (kill-ring-save) の後の kill コマンドは、M-wで kill リングにコピーされたテキストへの追加はしません。

9.3 グラフィカルなディスプレイでのカットアンドペースト

ほとんどのグラフィカルなデスクトップ環境では、異なるアプリケーション間のデータ転送 (通常はテキスト) に、クリップボード *(clipboard)* と呼ばれるシステム機能を使います。X では他にプライマリー選択 (primary selection) とセカンダリー選択 (secondary selection) という、同様の機能が利用可能です。Emacs をグラフィカルなディスプレイで実行している場合、kill と yank コマンドはこれらの機能に統合されているので、Emacs と他のグラフィカルアプリケーション間で、簡単にテキストを転送できます。

デフォルトでは、Emacs はプログラム間テキスト転送のコーディングシステムとして、UTF-8 を使います。もしコピーしたテキストが期待したものでない場合、`C-x RET x`または`C-x RET X`とタイプして、他のコーディングシステムを指定できます。`x-select-request-type`をカスタマイズして、異なるデータタイプを要求することもできます。Section 19.10 [Communication Coding], page 229 を参照してください。

9.3.1 クリップボードを使う

クリップボード *(clipboard)* とは、ほとんどのグラフィカルなアプリケーションが、“カットアンドペースト” のために使う機能です。もしクリップボードが存在する場合、Emacs の kill および yank コマンドもそれを使います。

何らかのテキストを、`C-w` (`kill-region`) のようなコマンドで kill したり、`M-w` (`kill-ring-save`) のようなコマンドで kill リングにコピーしたとき、そのテキストはクリップボードにも転送されます。

Emacs の kill コマンドがテキストをクリップボードに転送すると、通常ならクリップボードの既存の内容は失われます。古いクリップボードデータの喪失を防ぐために、Emacs はオプションでクリップボードの既存コンテンツを kill リングに保存できます。`save-interprogram-paste-before-kill`を数値にセットすると、そのデータの文字数がこの数値より小さければ kill リングにコピー、数値以外の非`nil`値なら常にコピーされます (データが大きくなるとメモリー消費が増えるというリスクがある)。

`C-y` (`yank`) のような yank コマンドもクリップボードを使います。他のアプリケーションがクリップボードを “所有” する場合 (たとえば Emacs で最後に kill コマンドを実行した後に、他のアプリケーションでテキストをカットまたはコピーした場合)、Emacs は kill リングではなくクリップボードから yank します。

通常 kill リングを `M-y` (`yank-pop`) で巡回することでは、クリップボードは変更されません。しかし`yank-pop-change-selection`を`t`に変更すると、`M-y`は新しい yank をクリップボードに保存します。

kill および yank コマンドがクリップボードにアクセスしないようにするには、変数`select-enable-clipboard`を`nil`に変更してください。

プログラムは平文テキスト以外のオブジェクトをクリップボードに置くことができます。たとえばウェブブラウザではイメージ上で通常は “Copy Image(イメージをコピー)” を選択でき、それによってイメージがクリップボードに配置されます。プラットフォームが対応していれば、`yank-media`コマンドによって Emacs がそれらのオブジェクトを yank できます。ただしこれをサポートしているモードに限られます (Section “Yanking Media” in *The Emacs Lisp Reference Manual* を参照)。

多くの X デスクトップ環境は、クリップボードマネージャー *(clipboard manager)* と呼ばれる機能をサポートします。もし Emacs がクリップボードのデータの現在の “持ち主” のときに Emacs を終了し、そのときクリップボードマネージャーが実行されていると、Emacs はクリップボードのデータをクリップボードマネージャーに転送するのでデータは失われません。ある状況において、これは Emacs が終了するが遅くなる原因となります。Emacs がクリップボードマネージャーにデータを転送しないようにするには、変数`x-select-enable-clipboard-manager`を`nil`に変更してください。

通常、クリップボードを通じて渡される NUL バイトを含む文字列は切り詰められるため、Emacs はシステムのクリップボードに転送する前に、そのような文字を “\\0” に置き換えます。

Emacs 24 以前は、kill および yank コマンドは、クリップボードではなくプライマリー選択 (Section 9.3.2 [Primary Selection], page 66 を参照してください) を使っていました。もしこのほうがよいなら、`select-enable-clipboard`を`nil`、`select-enable-primary`を`t`、`mouse-drag-copy-region`を`t`に変更してください。この場合は、次のコマンドを使って、クリップボードに明示的にア

クセスできます。リージョンを kill してクリップボードに保存するには `clipboard-kill-region`、リージョンを kill リングにコピーするとともにクリップボードに保存するには `clipboard-kill-ring-save`、クリップボードの内容をポイント位置に yank するには `clipboard-yank`です。

9.3.2 他のウィンドウアプリケーションにたいするカットアンドペースト

X ウィンドウシステム、PGTK、Haiku ではプライマリー選択 *(primary selection)* に、X アプリケーションで最後に選択されたテキスト (通常はマウスのドラッグで選択される) が存在します。一般的に、このテキストは他の X アプリケーションに `mouse-2`をクリックして挿入することができます。プライマリー選択はクリップボードとは別のものです。プライマリー選択の内容は、より脆弱です。なぜなら、クリップボードは明示的なカットまたはコピーだけにより上書きされるのにくらべ、プライマリー選択はマウスでテキストが選択される度に上書きされるからです。

X の下では、リージョンがアクティブ (Chapter 8 [Mark], page 52 を参照してください) になればいつでも、リージョンのテキストはプライマリー選択に保存されます。これは、そのリージョンの選択がマウスでドラッグやクリック (Section 18.1 [Mouse Commands], page 194 を参照してください) されたのか、キーボードコマンド (たとえば `C-SPC`をタイプしてからポイントを移動したなど。Section 8.1 [Setting Mark], page 52 を参照してください) なのかによらず適用されます。

変数 `select-active-regions`を `only`に変更すると、Emacs は一時的にアクティブになったリージョン (たとえばマウスやシフト選択など。Section 8.6 [Shift Selection], page 57 を参照してください) だけをプライマリー選択に保存します。`select-active-regions`を `nil`に変更すると、Emacs はアクティブなリージョンをプライマリー選択に保存しません。

プライマリー選択を Emacs のバッファーに挿入するには、挿入したい場所で `mouse-2` (`mouse-yank-primary`) をクリックします。Section 18.1 [Mouse Commands], page 194 を参照してください。`select-enable-primary` (Section 9.3.1 [Clipboard], page 65 を参照) がセットされていれば、そのテキストを挿入するために Emacs の通常の yank コマンド (`C-y`) の使用できます。

たとえ他のプログラムによってテキストが選択された後でも、典型的な X の挙動に反そて Emacs はリージョンをアクティブに保ちます。他のプログラムがプライマリー選択にデータを置いた後は、Emacs にリージョンを非アクティブにさせるには、グローバルなマイナーモード `lost-selection-mode`を有効にしてください。

MS-Windows はプライマリー選択を提供しませんが、Emacs は単一の Emacs セッション内で選択されたテキストを内部に格納することにより、これをエミュレートします。したがって Windows でも、プライマリー選択に関するすべての機能とコマンドは、X と同様に機能します。しかしこれは同一セッションにおけるカットやペーストなどの場合で、Emacs セッションと他のアプリケーション間では機能しません。

9.3.3 セカンダリー選択

プライマリー選択に加えて、X ウィンドウシステムはセカンダリー選択 *(secondary selection)* として知られる、同様な第 2 の機能を提供します。最近ではセカンダリー選択を使う X アプリケーションの数は多くありませんが、以下の Emacs コマンドによりアクセスできます:

`M-Drag-mouse-1`
: ボタンを押した場所からボタンを話した場所までを、セカンダリー選択としてセットします (`mouse-set-secondary`)。ドラッグして選択されたテキストは、フェイス `secondary-selection`を使ってハイライトされます。ウィンドウの上端または下端を越えてマウスをドラッグすると、`mouse-set-region`と同様にウィンドウは自動的にスクロールします (Section 18.1 [Mouse Commands], page 194 を参照してください)。

このコマンドは kill リングを変更しません。

`M-mouse-1`
: セカンダリー選択 *(secondary selection)* の終端をセットします (`mouse-start-secondary`)。もう一方のをセットして選択を完了するには、`M-mouse-3`を使用します。このコマンドは、新たなセカンダリー選択開始時に、既存のセカンダリー選択をキャンセルします。

`M-mouse-3`
: 以前に `M-mouse-1`で指定された位置から、`M-mouse-3`でクリックされた点を終点とするセカンダリー選択をセットします (`mouse-secondary-save-then-kill`)。これは選択されたテキストを kill リングにも保存します。同じ場所での 2 回目の `M-mouse-3` は、作成されたセカンダリー選択で選択されたテキストを kill します。

`M-mouse-2`
: クリックした場所にセカンダリー選択を挿入し、ポイントを yank したテキストの最後に配します (`mouse-yank-secondary`)。

`Mouse-1`と同様、`M-mouse-1`のダブルクリックで単語、トリプルクリックで行を処理します。

`mouse-yank-at-point`が非`nil`の場合には、`M-mouse-2`はポイント位置に yank します。どこをクリックしたか、さらにはフレームのどのウィンドウをクリックしたかは関係ありません。Section 18.1 [Mouse Commands], page 194 を参照してください。このユーザーオプションはインタラクティブな検索にも影響します。非`nil`ならフレームの任意の場所でのマウスによるテキストの yank は、検索文字列に追加されます。

9.4 テキストの追加

テキストのコピーや移動は、それを kill して yank することにより通常行います。しかし多くの箇所にあるテキストブロックをコピーしたり、たくさんのテキストの断片を 1ヶ所にコピーする便利な方法があります。ここではテキストの断片を、バッファーやファイルに追加するコマンドを説明します。

`M-x append-to-buffer`
: リージョンを指定したバッファーの内容の後に追加 (append) します。

`M-x prepend-to-buffer`
: リージョンを指定したバッファーの内容の前に追加 (prepend) します。

`M-x copy-to-buffer`
: リージョンを指定したバッファーにコピーして、バッファーの古い内容は削除されます。

`M-x insert-buffer`
: 指定したバッファーの内容を、現在のバッファーのポイント位置に挿入します。

`M-x append-to-file`
: リージョンを指定したファイルの内容の最後に追加します。

テキストをバッファーに追加するには、`M-x append-to-buffer`を使います。これはバッファー名を読み取り、リージョンのコピーを指定したバッファーに挿入します。存在しないバッファーを指定すると、`append-to-buffer`はそのバッファーを作成します。テキストは、そのバッファーのポイント位置に挿入されます。バッファーを編集用に使っていると、コピーされたテキストはその時ポイントがあった位置に挿入されます。

バッファーのポイントは、コピーされたテキストの最後に残ります。連続して`append-to-buffer`を使うと、テキストは指定したバッファーにコピーした順番で追加されていきます。厳密に言うと

append-to-bufferは既存のバッファーのテキストにたいして、常に追加をする訳ではありません。これはポイントがバッファーの最後にあるときに追加をします。しかしバッファーを変更するのにappend-to-bufferしか使わない場合、ポイントは常にバッファーの最後に位置することになります。

M-x prepend-to-bufferもappend-to-bufferと同様ですが、他のバッファーのポイントはコピーされたテキストの前に置かれるので、連続してこのコマンドを使用すると、テキストは逆の順番に追加されます。M-x copy-to-bufferも同様ですが、他のバッファーの既存の内容は削除されるので、バッファーの内容は新しくコピーされたテキストだけになります。

コマンド C-x x i (insert-buffer) は、追加するテキストを他のバッファーから取得するために使われます。これはバッファー名の入力を求め、そのバッファーのすべてのテキストのコピーを、現在のバッファーのポイント位置に挿入します。ポイントは挿入されたテキストの先頭になります。挿入されたテキストの最後の位置もマークリングに追加されます。マークは非アクティブになります。バッファーに関する背景情報は、Chapter 16 [Buffers], page 176 を参照してください。

バッファーのテキストを追加するかわりに、M-x append-to-fileでテキストを直接ファイルに追加できます。これはファイル名の入力を求め、リージョンのテキストを指定されたファイルの最後に追加します。ディスク上のファイルはすぐに変更されます。

append-to-fileは、Emacs が visit していないファイルだけに使うべきです。Emacs で編集中のファイルにたいして使用すると、それは Emacs の背後でファイルが変更されることになるため、編集内容が失われる可能性があります。

テキストの移動に関する他の方法はレジスターに格納する方法です。Chapter 10 [Registers], page 72 を参照してください。

9.5 矩形領域 (Rectangles)

矩形領域 *(rectangle)* コマンドは、テキストの矩形領域を操作します。矩形領域のテキストとは、特定の行範囲内にある、特定の 2 つの列の間にある文字すべての文字です。Emacs には矩形領域にたいして kill、yank、クリアー、スペースやテキストでフィル、削除を行うコマンドがあります。矩形領域コマンドは、複数列のテキストを操作したり、テキストをそのように変更したり戻したりする場合に便利です。

コマンドで操作する矩形領域を指定するには、一方の角にマークを設定し、その対角にポイントを置きます。このように設定した矩形領域を矩形リージョン (region-rectangle) と呼びます。ポイントとマークが同じ列の場合、矩形リージョンは空になります。ポイントとマークが同じ行の場合、矩形リージョンの高さは 1 行になります。

矩形リージョンは、リージョンの制御と大体同じ方法で制御できます。しかし、ポイントとマークの組がリージョンとして解釈されるのか、あるいは矩形領域として解釈されるかは、それらを使うコマンドに依存することに注意してください。

矩形リージョンはマウスを使用してマークすることもできます。矩形の一方の隅を C-M-mouse-1 でクリックして向かい側の隅までドラッグしてください。

C-x r k　矩形リージョンを kill して、最後に kill された矩形領域として、その内容を保存します (kill-rectangle)。

C-x r M-w　矩形リージョンのテキストを、最後に kill された矩形領域として保存します (copy-rectangle-as-kill)。

C-x r d　矩形リージョンのテキストを削除します (delete-rectangle)。

C-x r y　最後に kill された矩形領域の左上隅がポイント位置になるように yank します (yank-rectangle)。

`C-x r o` 矩形領域にスペースを挿入します (`open-rectangle`)。これにより矩形リージョンの以前の内容は右にずれます。

`C-x r N` 矩形リージョンの左端に行番号を挿入します (`rectangle-number-lines`)。これにより矩形リージョンの以前の内容は右にずれます。

`C-x r c` 矩形リージョンの内容をスペースに置き換えてクリアーします (`clear-rectangle`)。

`M-x delete-whitespace-rectangle`
指定された矩形領域の各行で、矩形領域の左端の列から空白文字を削除します。

`C-x r t` *string* `RET`
矩形領域の各行にたいして、内容を *string* に置き換えます (`string-rectangle`)。

`M-x string-insert-rectangle RET` *string* `RET`
矩形領域の各行にたいして、*string* を挿入します。

`C-x SPC` Rectangle Mark モードを切り替えます (`rectangle-mark-mode`)。このモードがアクティブのとき矩形領域はハイライトされ、拡大・縮小が可能になります。標準の kill および yank コマンドは、それにたいして操作を行います。

矩形領域の操作は 2 種類に分類できます。それは矩形領域を消去または挿入するものと、空の矩形領域を作るものです。

矩形領域のテキストを消去するには 2 つの方法があります。`C-x r d` (`delete-rectangle`) はテキストを無条件に削除します。`C-x r k` (`kill-rectangle`) はテキストを取り除いて、それを最後に *kill* された矩形領域として保存します。両方とも矩形領域の各行の指定したテキストを消去するように、矩形リージョンを消去します。その行の後に続くテキストがある場合、削除による隙間を生めるために後方に移動されます。

矩形領域の kill は普通の kill とは異なります。矩形領域は kill リングには保存されず、一番最後に kill された矩形領域だけを記録する、特別な場所に保存されます。矩形領域の yank は線形テキストの yank とは大きく異なるので、異なる yank コマンドが使われるからです。矩形領域にたいして yank の pop は定義されていません。

`C-x r M-w` (`copy-rectangle-as-kill`) は矩形領域用の `M-w`に相当します。これはバッファーからテキストを削除することなく、最後に kill された矩形領域として、矩形領域を記録します。

kill された矩形領域を yank するには、`C-x r y` (`yank-rectangle`) とタイプします。矩形領域の最初の行はポイント位置に挿入されます。矩形領域の 2 行目はポイントの 1 行下の位置に挿入され、以下同様に挿入されていきます。影響を受ける行数は、保存された矩形領域の高さにより決定されます。

たとえば 1 列のリスト 2 つを 2 列のリストに変換できます。一方の 1 列リストを矩形領域として kill してもう一方の 1 列リストの隣に yank すればよいのです。

`C-x r r` *r* と `C-x r i` *r* で、矩形領域をレジスターにコピーしたり取り出したりできます。Section 10.3 [Rectangle Registers], page 73 を参照してください。

空の矩形領域を作るために使うことのできるコマンドが 2 つあります。`C-x r c` (`clear-rectangle`) は、矩形リージョンの既存のテキストを空白に置き換えます。`C-x r o` (`open-rectangle`) は空白の矩形領域を挿入します。

`M-x delete-whitespace-rectangle`は、指定した列を起点に水平方向の空白文字を削除します。これは矩形領域の各行に適用され、開始列は矩形領域の左端です。矩形領域の右端はこのコマンドに影響を及ぼしません。

コマンド `C-x r N` (`rectangle-number-lines`) は、矩形領域の左端に行番号を挿入します。通常は矩形領域の最初の行を 1 として番号が開始されます。プレフィクス引数を指定すると、このコマ

ンドは開始番号と、番号を出力する際の書式文字列 (Section “Formatting Strings” in *The Emacs Lisp Reference Manual* を参照してください) の入力を求めます。

コマンド C-x r t (string-rectangle) は、矩形リージョンの各行を文字列で置き換えます。文字列の幅は矩形領域と同じ幅である必要はありません。矩形領域の後ろのテキストは、文字列の幅が少ないときは左に、文字列の幅が大きいときは右にシフトされます。

コマンド M-x string-insert-rectangleは、string-rectangleと同様ですが、各行に文字列を挿入し、元の文字列は右にシフトされます。

コマンド C-x SPC (rectangle-mark-mode) は、矩形リージョンをハイライトするか、標準のリージョンをハイライトするかを切り替えます (最初にリージョンをアクティブにする必要があります)。このモードが有効な場合、C-f、C-nなどのコマンドは矩形領域に合ったやり方でリージョンのサイズを変更し、kill、yank は矩形領域を処理します。Chapter 9 [Killing], page 59 を参照してください。このモードはリージョンがアクティブな間だけ持続します。

矩形リージョンが機能するのはマークがアクティブなときだけです。特に Transient Mark モードがオフ (Section 8.7 [Disabled Transient Mark], page 57 を参照) の場合には、マークをアクティブにするために更に C-x SPCをタイプする必要があるでしょう。

標準のリージョンとは異なり、バッファー終端を越えたり、TAB のような伸長された空白スペースの中間のような、通常はポイントを置けない場所にも矩形リージョンのコーナーを置くことができます。

リージョンがアクティブ (Chapter 8 [Mark], page 52 を参照)、かつそのリージョンが rectangle-mark-mode にある場合には、C-x C-xは矩形リージョンの 4 隅のコーナーを巡回するコマンド rectangle-exchange-point-and-markを実行します。これはマークされたテキストにたいする処理を呼び出す前に、矩形リージョンをのサイズを変更したい場合に便利です。

9.6 CUA バインド

コマンド M-x cua-modeは、多くのアプリケーションで使われている、CUA(Common User Access) 互換のキーバンドをセットアップします。

CUA モードが有効な場合、C-x、C-c、C-v、C-zなどのキーは、カット (kill)、コピー、ペースト (yank)、アンドウのコマンドを呼び出します。C-xと C-cによるカットとコピーは、リージョンがアクティブなときだけ処理されます。リージョンが非アクティブのときはプレフィクスキーとして動作するので、C-x C-cのような標準の Emacs コマンドは正常に機能します。変数 mark-even-if-inactive は C-xと C-cに影響を及ぼさないことに注意してください (Section 8.3 [Using Region], page 55 を参照してください)。

マークがアクティブのときに C-x C-fのような Emacs コマンドを入力するには、Shiftを押しながらプレフィクスキーを押す (例 S-C-x C-f) か、プレフィクスキーを素早く 2 回タイプ (例 C-x C-x C-f) します。

CUA モードが Emacs 標準のキーバインドをオーバーライドするのを無効にしつつ、以下で説明するそれ以外の CUA モードの機能は使う場合は、変数 cua-enable-cua-keysに nilをセットしてください。

CUA モードはデフォルトで Delete-Selection モード (Section 18.1 [Mouse Commands], page 194 を参照してください) を有効にするので、アクティブなリージョンがあるときテキストをタイプすると、そのテキストで置き換えられます。CUA モードでこれを無効にするには、変数 cua-delete-selectionを nilにセットしてください。

CUA モードは矩形領域を明白にハイライトする、強化された矩形領域サポートを提供します。C-RETを使うことにより矩形領域の選択が開始され、移動コマンドを使って拡張したり、C-xと C-c

で切り取りとコピーができます。RETにより、矩形領域の四隅に時計方向へカーソルを移動させるので、任意の方向に簡単に領域を拡張できます。タイプされた通常のテキストは、矩形領域の各行の左か右 (カーソルのある側) に挿入されます。

この矩形領域サポートは、cua-rectangle-mark-modeコマンドを呼び出すことにより、CUA モードを有効にせずに使うこともできます。標準コマンド rectangle-mark-modeもあります。Section 9.5 [Rectangles], page 68 を参照してください。

CUA モードでは、テキストや矩形領域を簡単にレジスターに保存したり、取り出すことができます。これは 1 桁の数引数を kill、copy、yank コマンドに指定します。たとえば C-1 C-cはリージョンをレジスター 1にコピーし、C-2 C-vはレジスター 2の内容を yank します。

CUA モードは、バッファー間での簡単にテキストを移動したりコピーするためのグローバルマーク機能も提供します。C-S-SPCを使って、グローバルマークのオンとオフが切り替えられます。グローバルマークがオンのときは、kill またはコピーされたすべてのテキストは自動的にグローバルマークの位置に挿入され、タイプしたテキストも現在のカーソル位置ではなくグローバルマークに挿入されます。

たとえば複数のバッファーから単語をコピーして単語リストを作るには、単語リストを作るバッファーにグローバルマークをセットします。次にリストにしたい単語をマーク (S-M-fなど) してから、C-cか M-wでリストにコピーします。そして RETでリストにコピーされた単語の後ろに改行を挿入すればよいのです。

10 レジスター

Emacs のレジスター *(registers)* は、テキストや矩形領域、位置、その他、後で使うものを保存するための区画です。一度テキストや矩形領域をレジスターに保存すれば、それをバッファーに何度もコピーできます。一度場所をレジスターに保存すれば、何度でもその場所にジャンプして戻ることができます。

各レジスターは 1 文字からなる名前があり、ここでは *r* と表記することにします。*r* には、英字 (‘a’など) または数字 (‘1’など) を使用できます。大文字小文字は区別されるので、レジスター ‘a’とレジスター ‘A’は同じではありません。たとえば ‘*’や ‘C-d’のような、非英数字にレジスターをセットすることもできます。‘C-g’と ‘ESC’は対話的なコマンドを終了させるために予約済みなので、これらのキーにレジスターをセットすることはできないことに注意してください。

レジスターには位置、テキスト、矩形領域、数字、ウィンドウやフレームの構成、、バッファー名、ファイル名が保存できますが、一度に保存できるのは 1 つです。レジスターに何か保存すると、他の何かをそのレジスターに保存するまで残ります。レジスター *r* に何が含まれているのか見るには。`M-x view-register`を使います:

`M-x view-register RET` *r*
: レジスター *r* に何が含まれるかの説明を表示します。

レジスター名の入力を求めるコマンドはすべて、既存のレジスターを一覧するプレビュー (preview) ウィンドウを遅延表示します。遅延の長さは `register-preview-delay`でカスタマイズできます。遅延を無効にするには、`nil`をセットしてください。この場合、`C-h`か `F1`で、明示的にプレビューウィンドウを要求できます。

ブックマーク *(Bookmarks)* はファイルと位置を記録するので、ファイルを再び見るときは記録された位置から閲覧できます。ブックマークも本質的にレジスター同じなので、このチャプターに記載します。

10.1 レジスターに位置を保存する

`C-x r SPC` *r*
: 現在のバッファーのポイント位置をレジスター *r* に記録します (`point-to-register`)。

`C-x r j` *r*　レジスター *r* に記録されたバッファーの位置にジャンプします (`jump-to-register`)。

`C-x r SPC` (`point-to-register`) と、それに続けて文字 *r* をタイプすると、ポイント位置と現在のバッファーの両方をレジスター *r* も保存します。レジスターは他の何かが保存されるまでこの情報を保持します。

コマンド `C-x r j` *r* は *r* に記録されたバッファーに切り替えて、マークを push するとともに記録された位置にポイントを移動します (すでにポイントが記録された位置にあるとき、またはこのコマンドの連続呼び出し時にはマークは push されない)。レジスターの内容は変わらないので、保存した位置に何度でもジャンプできます。

`C-x r j`を使って保存した位置に移動するとき、保存されたバッファが kill されていた場合、`C-x r j`は同じファイルを visit してバッファーを生成しようと試みます。もちろんこれはファイルを visit したバッファーだけの動作です。

10.2 レジスターにテキストを保存する

同じテキストのコピーを何回も挿入したいとき、kill リングから yank するのは不便です。なぜなら何か kill するたびに、そのエントリーはリングの下の方へ移動してしまうからです。代替として、テキストをレジスターに保存して、後で取り出す方法があります。

`C-x r s r` リージョンをレジスター *r*にコピーします (`copy-to-register`)。

`C-x r i r` レジスター *r*からテキストを挿入します (`insert-register`)。

`M-x append-to-register RET r`
: リージョンをレジスター *r*のテキストに追加します。

  レジスター *r*の内容がテキストの場合、そのレジスターに追加するのに `C-x r +` (`increment-register`) も使用できます。レジスター *r*に数値が含まれている場合、コマンド `C-x r +`は違う動作をすることに注意してください。Section 10.5 [Number Registers], page 74 を参照してください。

`M-x prepend-to-register RET r`
: リージョンをレジスター *r*の先頭に追加します。

`C-x r s r`は、リージョンのテキストのコピーを、*r*という名前のレジスターに保存します。マークが非アクティブのとき、Emacs はまず最後にセットされたマークをアクティブにします。マークはこのコマンドの最後に非アクティブになります。Chapter 8 [Mark], page 52 を参照してください。同じコマンドにプレフィクス引数を指定した `C-u C-x r s r`は、テキストのコピーをレジスター *r*に保存してから、バッファーのテキストを削除します。これはリージョンのテキストを、レジスターに移動したと考えることができます。

`M-x append-to-register RET r`は、リージョンのテキストのコピーを、*r*という名前のレジスターにすでに保存されているテキストに追加します。プレフィクス引数を指定した場合、レジスターに追加した後にリージョンを削除します。コマンド `prepend-to-register`も同様ですが、これはリージョンのテキストをレジスターのテキストの最後ではなく先頭に追加します。

`append-to-register`と`prepend-to-register`を使ってテキストを集める場合、セパレーターを使って個々に集めたテキストを分割したい場合があります。そのようなときは`register-separator`を構成して、セパレーター文字列をそのレジスターに保存します。たとえばテキストを収集する過程で、個々のテキストを 2 つの改行で分けたい場合、以下の設定を使うことができます。

```
(setq register-separator ?+)
(set-register register-separator "\n\n")
```

`C-x r i r`は、レジスター *r*のテキストをバッファーに挿入します。通常はポイントをテキストの後に置き、非アクティブのマークをテキストの前にセットします。プレフィクス引数を指定したときは、ポイントをテキストの前、マークをテキストの後にセットします。

10.3 レジスターに矩形領域を保存する

レジスターには線形のテキストだけでなく、矩形領域も保存できます。バッファーで矩形領域を指定する方法は、Section 9.5 [Rectangles], page 68 を参照してください。

`C-x r r r` 矩形リージョンをレジスター *r*にコピーします (`copy-rectangle-to-register`)。プレフィクス引数を指定するとコピー後に矩形リージョンを削除します。

`C-x r i r` レジスター *r*に矩形リージョンが保存されている場合、それを挿入します (`insert-register`)。

前に Section 10.2 [Text Registers], page 73 でも説明した C-x r i r (insert-register) コマンドは、レジスターに矩形領域が保存されているときはテキストではなく矩形領域を挿入します。

10.4 レジスターにウィンドウやフレームの構成を保存する

選択されたフレームのウィンドウの設定や、すべてのフレームのすべてのウィンドウの設定もレジスターに保存して、後で設定を復元することができます。ウィンドウの設定については、Section 17.7 [Window Convenience], page 192 を参照してください。

C-x r w *r* 選択されたフレームのウィンドウの設定を、レジスター *r*に保存します (window-configuration-to-register)。

C-x r f *r* すべてのフレームおよびフレームに含まれるすべてのウィンドウの状態 (フレームセットとも呼ばれる) を、レジスター *r*に保存します (frameset-to-register)。

C-x r j *r*を使うと、ウィンドウまたはフレームの設定を復元できます。これはカーソル位置を復元するコマンドと同じです。フレームの設定を復元するとき、設定に含まれていないフレームは非表示になります。もしこれらのフレームを削除したいときは、かわりに C-u C-x r j *r*を使います。

10.5 レジスターに数字を保存する

数字をレジスターに保存して、その数字 (10 進) をバッファーに挿入したり、増加させるコマンドがあります。これらのコマンドはキーボードマクロで使うと便利です (Chapter 14 [Keyboard Macros], page 137 を参照してください)。

C-u *number* C-x r n *r*
*number*をレジスター *r*に保存します (number-to-register)。

C-u *number* C-x r + *r*
*r*に数字が保存しているときは、レジスターの数字を *number*だけ増加させます。コマンド C-x r + (increment-register) は、*r*にテキストが保存されているときは異なる動作をすることに注意してください。Section 10.2 [Text Registers], page 73 を参照してください。

C-x r i *r* レジスター *r*の数字をバッファーに挿入します。

C-x r iは、他のレジスターの内容をバッファーに挿入するコマンドと同じです。C-x r +に数字の引数を与えない場合、レジスターの値は 1 増加します。C-x r nに数字の引数を与えない場合、レジスターには 0 が保存されます。

10.6 レジスターにファイルやバッファーの名前を保存する

特定の名前のファイルを頻繁に visit する場合、その名前をレジスターにセットしておけば、より便利にファイルを visit することができます。以下は *name*というファイルをレジスター *r*にセットする Lisp コードです:

```
(set-register r '(file . name))
```

たとえば、

```
(set-register ?z '(file . "/gd/gnu/emacs/19.0/src/ChangeLog"))
```

はレジスター ‘z’にファイル名をセットします。

レジスター *r*に名前がセットされているファイルを visit するには、C-x r j *r*とタイプします。これはある位置にジャンプしたり、フレームの設定を復元するのと同じコマンドです。

同様にもし特定のバッファーを頻繁に visit するのであれば、それらのバッファー名をレジスターに置くことができます。たとえばよく ‘*Messages*’バッファーを visit するのなら、以下のスニペットを使えばそのバッファーをレジスター ‘m’に置くことができます:

```
(set-register ?m '(buffer . "*Messages*"))
```

レジスター *r*に名前がセットされているバッファーに切り替えるには C-x r j *r*とタイプしてください。

10.7 キーボードマクロのレジスター

あるキーボードマクロ (Chapter 14 [Keyboard Macros], page 137 を参照してください) を頻繁に実行する必要がある場合、それをレジスターにセットしたり保存することができればより便利でしょう (Section 14.5 [Save Keyboard Macro], page 142 を参照してください)。C-x C-k x *r* (kmacro-to-register) は、最後のキーボードマクロをレジスター *r*に保存します。

レジスター *r*のキーボードマクロを実行するには、C-x r j *r*とタイプします (これはある位置にジャンプしたりフレームを復元するのと同じコマンドです)。

10.8 ブックマーク

ブックマーク *(Bookmarks)* とは、ジャンプしたい位置を記録するレジスターのようなものです。レジスターとの違いは、長い名前をもつことができ、次の Emacs セッションに自動的に引き継がれることです。ブックマークの典型的な使い方は、さまざまなファイルのどこを読んでいたかを記録することです。

C-x r m RET
: visit しているファイルのポイント位置に、ブックマークをセットします。

C-x r m *bookmark* RET
: ポイント位置に、*bookmark*という名前のブックマークをセットします (bookmark-set)。

C-x r M *bookmark* RET
: C-x r mと同様ですが、既存のブックマークを上書きしません。

C-x r b *bookmark* RET
: *bookmark*という名前のブックマークにジャンプします (bookmark-jump)。

C-x r l
: すべてのブックマークを一覧します (list-bookmarks)。

M-x bookmark-save
: 現在のすべてのブックマークの値を、デフォルトのブックマークファイルに保存します。

visit しているファイル内のカレント位置を記録するには、コマンド C-x r mを使用します。これは、ブックマーク名のデフォルトとしてファイル名を使います。ブックマークが指すファイルをもとにブックマークの名前を付ければ、C-x r bで任意のファイルを再び visit して、同時にブックマーク位置に移動するという操作を楽に行えます。

コマンド C-x r M (bookmark-set-no-overwrite) は C-x r m と同じように機能しますが、指定されたブックマークがすでに存在する場合は、上書きするかわりにエラーをシグナルします。

すべてのブックマークのリストを別のバッファーに表示するには、C-x r l (list-bookmarks) とタイプします。そのバッファーに切り替えて、ブックマークの定義の編集やブックマークに注釈をつけることができます。ブックマークバッファーで C-h mとタイプすれば、特別な編集コマンドに関する情報を見ることができます。

Emacs を終了するとき、もしブックマークの値を変更していたら、Emacs はブックマークを保存します。M-x bookmark-saveコマンドで、いつでもブックマークを保存できます。ブックマークは ~/.emacs.d/bookmarksというファイルに保存されます (古いバージョンの Emacs との互換性を保つため、もし~/.emacs.bmkというファイルがあればそのファイルに保存します)。ブックマークコマンドは、デフォルトのブックマークファイルを自動的にロードします。この保存とロードにより、ブックマークの内容を次の Emacs セッションに引き継ぐことができるのです。

bookmark-save-flagに 1 をセットすると、ブックマークをセットするコマンドはブックマークの保存も行ないます。こうすることにより、Emacs がクラッシュしてもブックマークを失わずにすみます。この変数の値が数字の場合、それはブックマークを何回変更したら保存するという意味です。この変数に nilをセットすると、Emacs は明示的に M-x bookmark-saveを使ったときだけブックマークを保存します。

変数 bookmark-default-fileには、ブックマークを保存するデフォルトのファイルを指定します。

変数 bookmark-use-annotationsを tにセットすれば、ブックマークへの問い合わせは注釈について行われるようにセットされます。ブックマークに注釈があれば、ブックマークへのジャンプ時には別ウィンドウに注釈が自動的に表示されます。

ブックマークの位置は周囲のコンテキストとともに保存されるので、ファイルが多少変更されていても bookmark-jumpは正確な位置を見つけることができます。変数 bookmark-search-sizeはブックマーク位置のコンテキストの前後何文字を記録するかを指定します (暗号化されたファイルを visit しているバッファーでは、この変数の値に関係なくブックマークファイルにコンテキストは保存されない)。

以下はブックマークを処理する追加のコマンドです:

M-x bookmark-load RET *filename* RET
: ブックマークのリストを含む、*filename*という名前のファイルをロードします。このコマンドは bookmark-writeと同様に、デフォルトのブックマークファイルに加えて、他のファイルのブックマークを使うことができます。

M-x bookmark-write RET *filename* RET
: 現在のすべてのブックマークをファイル *filename*に保存します。

M-x bookmark-delete RET *bookmark* RET
: *bookmark*という名前のブックマークを削除します。

M-x bookmark-insert-location RET *bookmark* RET
: ブックマーク *bookmark*が指すファイル名をバッファーに挿入します。

M-x bookmark-insert RET *bookmark* RET
: ブックマーク *bookmark*が指すファイルの内容をバッファーに挿入します。

11 ディスプレイの制御

ウィンドウに入りきらない大きなバッファーでは、Emacs はその一部しか表示できません。このチャプターでは、見たい部分のテキストを指定するコマンドや変数と、どのようにしてテキストが表示されるかを説明します。

11.1 スクロール

ウィンドウがバッファーのすべてのテキストを表示するには小さい場合、その一部だけが表示されます。スクロールコマンドは、バッファーで表示される部分を変更します。

前方 (forward) または上 (up) へのスクロールは、ウィンドウに表示される部分を先に進めます。これはウィンドウに表示されるバッファーのテキストを、上に移動させるのと同じです。後方 (backward) または下 (down) へのスクロールは、ウィンドウに表示される部分を前に戻します。これはウィンドウに表示されるバッファーのテキストを下に移動させます。

Emacs での up と down は、ウィンドウでテキストが移動する方向に基づいており、テキストにたいしてウィンドウが移動する方向ではありません。この用語は現在の scrolling up や scrolling down が広まる前に、Emacs で採用されました。そのため `PageDown`は、Emacs 的には上 (up) にスクロールするという、奇妙な結果となりました。

ウィンドウに表示されているバッファー部分には、常にポイントが含まれています。もしウィンドウの下端か上端を越えてポイントを移動させると、ポイントを画面に表示させるために自動的にスクロールが発生します (Section 11.3 [Auto Scrolling], page 79 を参照してください)。以下のコマンドで明示的にスクロールができます:

`C-v`
`PageDown`
`next`　　ウィンドウのほぼ全画面分、前方にスクロールします (`scroll-up-command`)。

`M-v`
`PageUp`
`prior`　　後方にスクロールします (`scroll-down-command`)。

`C-v` (`scroll-up-command`) は、ウィンドウ全体の高さに近い量、前方にスクロールします。これにより下端の 2 行が上端になるようスクロールして、それに続く表示されていなかった行を表示します。ポイントが上端より上になってしまう場合、ウィンドウの新たな上端の行に移動します。The `PageDown`(または `next`) は、`C-v`と同じです。

`M-v` (`scroll-down-command`) は、同様の方法で後方にスクロールします。`PageUp`(または `prior`) は、`M-v`と同じです。

スクロールコマンドでオーバーラップして表示される行数は、変数 `next-screen-context-lines`で制御され、デフォルトは 2 です。数引数 *n*を与えたときは、*n*行スクロールします。Emacs はポイントを変更しないよう試みるので、テキストとポイントは一緒に上または下に動きます。`C-v`に負の引数を与えると、`M-v`のように反対方向へスクロールします。

デフォルトでは、ウィンドウがバッファーの先頭または最後に到達していて、これ以上スクロールできない場合、これらのコマンドは (beep 音をならしたり画面をフラッシュして) エラーをシグナルします。変数 `scroll-error-top-bottom`を `t`に変更すると、コマンドは可能な限り、つまりバッファーの先頭の文字または最後の文字にポイントを移動します。ポイントがすでにそこにあるときは、エラーをシグナルします。

スクロールしたとき、ポイントが同じスクリーン位置に留まることを好むユーザーもいます。そうすれば同じスクリーンにスクロールして戻ったとき、ポイントが元の位置にあると便利だからです。こ

の動作は変数 scroll-preserve-screen-positionを通じて利用可能です。変数の値が tのとき、スクロールコマンドによりポイントがウィンドウの外にでるような場合、Emacs はポイントを先頭行または最終行に移動させるのではなく、同じスクリーン位置にポイントを調整して、カーソルを維持します。その他の非 nil値の場合、Emacs はスクロールコマンドによりポイントがウィンドウに残っている場合にも、この方法でポイントを調整します。この変数はこのセクションで説明するすべてのスクロールコマンド、同様にマウスホイールによるスクロール (Section 18.1 [Mouse Commands], page 194 を参照してください) に影響を与えます。一般的にいうと、この変数は scroll-commandプロパティが非 nilのコマンドに影響を及ぼします。Section “Property Lists” in *The Emacs Lisp Reference Manual* を参照してください。インクリメンタル検索においてこのプロパティが非 nilのコマンドが呼び出された際に、isearch-allow-scrollが非 nilであればインクリメンタル検索を終了させないという効果もあります (Section 12.1.6 [Not Exiting Isearch], page 110 を参照)。

ときどき、特に C-vや M-vのようなキーを押したままにすると、キーボードのオートリピートがアクティブになり、要求された高レートのスクロール要求に Emacs が対応できなくなるときがあります。そのような場合、表示は更新されず、かなり長時間の間、Emacs が応答しなくなる可能性があります。変数 fast-but-imprecise-scrollingを非 nil値にセットすることにより、この状況に対処できます。これはスクロールコマンドにフォント表示化 (Section 11.13 [Font Lock], page 89 を参照してください) を行なわないように指示します。スクロールされるテキストは、それらがデフォルトフェイスをもつと仮定されなくなり、フォント表示化されなくなります。これにより、そのフェイスがすべて同じサイズのフォントを使用していない場合には、(たとえばオートリピートではない)1 回のスクロールでも、Emacs が誤ったバッファー位置にスクロールするかもしれません。

fast-but-imprecise-scrollingをセットするかわりに JIT Lock による遅延されたフォント表示化の有効化のほうが好ましいと思うかもしれません (Section 11.13 [Font Lock], page 89 を参照)。これを行うには jit-lock-defer-timeに 0.25、タイプ速度が速い場合には 0.1 のように小さい正の数値をセットします。これにより C-v押下時のぎくしゃくしたスクロールが幾分改善されますが、バッファーの新たな部分へスクロールするようなアクション後のウィンドウのコンテンツは一時的に非フォント化されるでしょう。

最後にこれらの変数にかわる 3 つ目の選択肢が redisplay-skip-fontification-on-inputです。この変数が非 nilなら、保留中の入力がある場合にはいくつかのフォント表示化をスキップします。いずれにせよ入力の保留中は再表示を完全にスキップするので通常なら表示に影響はありませんが、不要なフォント表示化を避けることでスクロールがスムーズになる可能性があります。

コマンド M-x scroll-upおよび M-x scroll-downは、scroll-up-commandおよび scroll-down-commandと同様に動作しますが、scroll-error-top-bottomを考慮しません。これらのコマンドは、Emacs 24 以前ではスクロールアップおよびスクロールダウンのためのデフォルトでした。コマンド M-x scroll-up-lineおよび M-x scroll-down-lineは、現在のウィンドウを 1 行スクロールさせます。もしこれらのコマンドを使う場合は、それにキーバインドを割り当てたくなるでしょう (Section 33.3.6 [Init Rebinding], page 522 を参照してください)。

11.2 センタリング

C-l 選択されているウィンドウで、現在行が中央になるようスクロールします。連続して呼び出すと、次は現在行が上端になり、その次は現在行が下端に、という順番で循環します。画面の再描画が行われる可能性があります (recenter-top-bottom)。

C-M-S-l 他のウィンドウをスクロールします。これは他のウィンドウに作用する C-lと等価です。

M-x recenter
選択されているウィンドウで、現在行が中央になるようにスクロールします。画面の再描画が行われる可能性があります。

C-M-l 有用な情報が表示されるよう、発見的な手法でスクロールします (reposition-window)。

C-l (recenter-top-bottom) コマンドは、選択されたウィンドウにたいしてセンタリング *(recenters)* をします。これにより現在のスクリーン行がウィンドウの中央、または中央に一番近い行になります。

(C-l C-l) のように C-lを 2 回タイプすると、ポイントのある行が上端になるようにスクロールします。C-lを 3 回タイプするとポイントのある行が下端になるようにスクロールします。連続して C-lをタイプすることにより、上記の 3 つの位置を循環してスクロールできます。

リスト変数 recenter-positionsをカスタマイズすることにより、循環する順序を変更できます。リスト要素にはシンボル top、middle、bottomまたは数値を指定します。数値に整数を指定すると、現在行が指定したスクリーン行になるようスクロールします。数値に 0.0 から 1.0 の浮動小数点数は、ウィンドウの上端から現在行までを割合で指定します。デフォルトは (middle top bottom)で、これは上述した循環順序です。さらに変数 scroll-marginを、0 以外の値 *n*に変更すると、C-lは常にウィンドウの上端または下端からスクリーン行で、*n*行を残してスクロールします (Section 11.3 [Auto Scrolling], page 79 を参照してください)。

C-lにプレフィクス引数を指定することもできます。C-u C-lのようにプレフィクス引数だけを指定すると、単にポイントを示す行を中央にします。正の引数 *n*は、ポイントを示す行がウィンドウの上端から *n*行目になるようにスクロールします。0 を指定すると、ポイントのある行がウィンドウ上端になるようにスクロールします。負の引数 $-n$は、ポイントのある行がウィンドウの下端から *n*行目になるようにスクロールします。引数を与えたときは、C-lは画面をクリアーせず、異なるスクリーン位置への循環も行いません。

変数 recenter-redisplayが非 nil値の場合、C-lはスクリーンのクリアーと再描画を行います。特別な値 tty(デフォルト) は、これをテキスト端末上のフレームだけに限定します。再描画はスクリーンが何らかの理由により文字化けしてしまったときなどに便利です (Section 34.2.2 [Screen Garbled], page 538 を参照してください)。

より原始的なコマンド M-x recenterは、recenter-top-bottomと同じように振る舞いますが、スクリーン位置を循環しません。

C-M-l (reposition-window) は、有用な情報がスクリーンに表示されるように、現在のウィンドウを発見的な手法によりスクロールします。たとえば Lisp ファイルの場合、このコマンドは可能な限り現在の defun 全体がスクリーン上に表示されるよう試みます。

11.3 自動スクロール

ポイントが表示されているテキスト部分から外に移動すると、Emacs は自動スクロール *(automatic scrolling)* の処理を行います。通常自動スクロールは、ウィンドウの垂直方向の中央にポイントをセンタリングしますが、この振る舞いを変えるいくつかの方法があります。

scroll-conservativelyに小さい数字 *n*をセットすると、ポイントが少し (*n*行以下) スクリーンの外に出たら、Emacs はポイントがスクリーンに表示されるのに充分なだけスクロールします。これでもしポイントの表示に失敗した場合、Emacs はそのウィンドウの中央にポイント行が表示されるのに充分なだけのスクロールをします。scroll-conservativelyに大きな数字 (100 より大) も数字をセットすると、どれだけポイントを移動させようと、自動スクロールはポイント行を中央にセンタリングしなくなります。Emacs はポイントが表示されるように常にテキストをスクロールします。ウィンドウの上端または下端かはスクロールの方向に依存します。デフォルトでは scroll-conservatively は 0 で、これは常にポイント行がウィンドウの中央にセンタリングされることを意味します。つまりミニバッファーウィンドウでは、scroll-conservativelyより優先される scroll-minibuffer-conservativelyがデフォルトでは非 nilなので、常に conservative(保守的) なスクロールになると言うことです。

自動スクロールを制御する他の方法は、変数scroll-stepをカスタマイズすることです。この変数の値はポイントがスクリーンから外れたとき、何行を自動スクロールさせるかを決定します。その行数スクロールしてもポイントが表示されない場合、かわりにポイント行が中央にきます。デフォルト値は 0 で、スクロール後は常にポイント行が中央にきます。

自動コントロールを制御する 3 番目の方法は、変数scroll-up-aggressivelyとscroll-down-aggressivelyをカスタマイズすることで、これは直接スクロール後のポイントの垂直位置を指定します。scroll-up-aggressivelyの値には、nil(デフォルト)、または 0 から 1 までの浮動小数点数 *f* を指定します。ポイントがウィンドウの下端を越えたとき (たとえば前方にスクロールしたとき)、Emacs はウィンドウの高さとウィンドウの下端からポイント行までの割合が、*f* になるようスクロールします。つまり *f* を大きくするとより積極的 (aggressive)、つまり新しいテキストがより多く表示されることを意味します。デフォルト値 nilは 0.5 と同じです。

同様に scroll-down-aggressivelyは、ポイントがウィンドウの上端を越えたとき (たとえば後方にスクロールしたとき) の振る舞いを設定します。値にはスクロール後のウィンドウの上端からポイント行までのマージンを指定します。つまり scroll-up-aggressivelyを大きくすると、より積極的になります。

変数 scroll-conservatively、scroll-step、および scroll-up-aggressivelyとscroll-down-aggressivelyは、互いに矛盾する方法で自動スクロールを制御します。したがって自動スクロールをカスタマイズする場合は、2 つ以上の手法を選ぶべきではありません。もし 2 つ以上の変数をカスタマイズする場合は、scroll-conservatively、次に scroll-step、そして最後に scroll-up-aggressivelyと scroll-down-aggressivelyという優先順でカスタマイズしてください。

変数 scroll-marginは (たとえ scroll-up-aggressively や scroll-down-aggressively に、上端または下端からのマージンがウィンドウにたいして占める割合より大きくなるような *f* を指定していても)、ポイントがウィンドウの上端または下端にどれだけ近づけるかを制限します。変数の値にはスクリーン行の行数です。もしポイントがウィンドウの上端または下端から指定した行数の位置にくると、Emacs は自動的にスクロールします。デフォルトでは scroll-marginは 0 です。デフォルトではそのウィンドウの高さの 1/4 に制限されていますが、maximum-scroll-marginをカスタマイズすることにより 1/2 まで増加 (または 0 まで減少) させることができます。

11.4 水平スクロール

水平スクロール *(Horizontal scrolling)* は、ウィンドウの行を右方向に移動させます。そのため左端の近くのテキストは表示されなくなります。ウィンドウのテキストが水平スクロールされると、テキスト行は折り返されるのではなく、切り詰め (truncated) られます。ウィンドウが切り詰められた行を表示しているとき、ポイントがスクリーンの左端か右端を越えて移動すると、Emacs は自動的に水平スクロールを行います。デフォルトではそのウィンドウ内のすべての行は一緒に水平スクロールされますが、変数 auto-hscroll-modeに特別な値 current-lineをセットした場合は、カーソルを表示する行だけがスクロールされます。自動的な水平スクロールを完全に無効にするには、変数auto-hscroll-modeに nilをセットしてください。また自動的な水平スクロールがオフになっている場合、ポイントがスクリーンの端を越えると、それを知らせるためにカーソルが表示されなくなることに注意してください (テキスト端末の場合カーソルは端に残されます)。

変数 hscroll-marginは、自動的なスクロールが起こる前に、ポイントがウィンドウの左端または右端に、どれだけ近づけるかを制御します。変数の値は列数で指定します。たとえば変数の値が 5 のときは、端から 5 列目にポイントが移動すると、水平スクロールが発生します。

変数 hscroll-stepは m ポイントが端に近づきすぎたときに、何列スクロールするかを決定します。デフォルト値の 0 は、ポイントがウィンドウの中央になるようにスクロールされることを意味し

ます。正の整数はスクロールされる列数を指定します。浮動小数点数 (0 から 1 の値であること) は、スクロールされる量を、ウィンドウの幅にたいする割合で指定します。

以下のコマンドで明示的に水平スクロールすることもできます:

`C-x <` 現在のウィンドウのテキストを左にスクロールします (`scroll-left`)。

`C-x >` 右にスクロールします (`scroll-right`)。

`C-x <` (`scroll-left`) は選択された、ウィンドウをウィンドウ幅から 2 列少ない列数、左にスクロール (いいかえればウィンドウのテキストは左に移動) します。数引数 *n*を指定すると、*n*列スクロールします。

テキストが左にスクロールされて、ポイントがウィンドウの左端を越えると、ポイントが表示されているテキストに戻るまで、カーソルはフリーズします。これは `auto-hscroll-mode`の設定とは独立しています (これはテキストを左にスクロールするときのウィンドウの右端での振る舞いだけに影響します)。

`C-x >` (`scroll-right`) は、同様に右にスクロールします。ウィンドウが通常の表示 (行の先頭がウィンドウの左端に表示されている状態) のときは、それ以上スクロールできないので何も起こりません。これは `C-x >`の引数を正確に計算する必要がないことを意味します。充分に大きな引数を与えれば、通常の表示が復元されます。

これらのコマンドでウィンドウを水平方向にスクロールすると、自動水平スクロールの下限値がセットされます。自動スクロールはウィンドウのスクロールを続けますが、前に `scroll-left`にセットされた値を越えて右にスクロールできなくなります。`auto-hscroll-mode`が `current-line`にセットされているときは、カーソルを表示する行以外の行は、最小限度だけスクロールされるでしょう。

11.5 ナローイング

ナローイング *(Narrowing)* とはバッファーのある範囲にフォーカスを置き、他の部分を一時的にアクセス不能にすることを意味します。扱うことのできる範囲のことを、アクセス可能範囲 *(accessible portion)* と呼びます。ナローイングを取り消すと、バッファー全体に再びアクセスできるようになります。これをワイドニング *(widening)* と呼びます。バッファーにたいして、ナローイングにより境界を設けることを、バッファーの制限 *(restriction)* と呼びます。

ナローイングにより、他の部分に気を取られずに、1 つのサブルーチンやパラグラフに集中することが容易になります。ナローイングは、置換コマンドやキーボードマクロの繰り返しにより操作される範囲を制限するためにも使われます。

`C-x n n` ポイントとマークの間にナローイングします (`narrow-to-region`)。

`C-x n w` バッファー全体をワイドニングして、再びアクセス可能にします (`widen`)。

`C-x n p` 現在のページにナローイングします (`narrow-to-page`)。

`C-x n d` 現在の defun にナローイングします (`narrow-to-defun`)。

バッファーをナローイングしているときは、表示されている範囲がすべてです。残りの部分を見ることはできず、移動もできず (移動コマンドによりアクセス可能範囲の外に移動することはできません)、変更もできません。しかし残りの部分がなくなったわけではないので、ファイルを保存するとアクセス不能範囲のテキストもすべて保存されます。ナローイングが有効なときは、モードラインに‘`Narrow`’という単語が表示されます。

主要なナローイングコマンドは、`C-x n n` (`narrow-to-region`) です。これは現在のバッファーを制限するので、現在のリージョンだけがアクセス可能になり、リージョンの前後のすべてのテキストはアクセス不能になります。ポイントとマークは変化しません。

かわりに C-x n p (narrow-to-page) を使うと、現在のページにナローイングされます。ページの定義については、Section 22.4 [Pages], page 254 を参照してください。C-x n d (narrow-to-defun) は、ポイントを含む defun にナローイングします (Section 23.2 [Defuns], page 286 を参照してください)。

ナローイングを取り消す方法は、C-x n w (widen) です。これにより再びバッファーのテキストすべてにアクセス可能になります。

バッファーのどの範囲にナローイングされているかは、C-x =コマンドを使って情報を得ることができます。Section 4.9 [Position Info], page 24 を参照してください。

ナローイングは、それを理解していないユーザーを容易に混乱させるので、通常 narrow-to-regionコマンドは無効になっています。このコマンドを使おうとすると、Emacs は確認を求め、有効にするオプションを提供します。このコマンドを有効にすると、それ以降は確認を求められなくなります。Section 33.3.11 [Disabling], page 527 を参照してください。

11.6 View モード

View モードは、バッファーをスクリーン上でスキャンするためのマイナーモードです。このモードは、バッファーを変更せずにスクロールする、便利なコマンドを提供します。Emacs のカーソル移動コマンドとは別に、SPCで前方にスクロール、S-SPCまたは DELで後方にスクロール、sでインクリメンタルサーチができます。

q (View-quit) とタイプすると View モードが無効になり、View モードが有効になる前のバッファーの位置に戻ります。e (View-exit) とタイプすると View モードが無効になり、現在のバッファーと位置は維持されます。

M-x view-bufferは、既存の Emacs バッファー名を入力として求め、そのバッファーに切り替えて View モードを有効にします。M-x view-fileはファイル名を入力として求め、そのファイルを visit して View モードを有効にします。

11.7 Follow モード

Follow モードは、同じバッファーを表示する 2 つのウィンドウを、1 つの仮想ウィンドウとしてスクロールするマイナーモードです。Follow モードを使うには、ウィンドウが 1 つだけのフレームを選択して、それを C-x 3を使って縦に並べて 2 分割してから、M-x follow-modeとタイプします。それ以降はバッファーをどちらのウィンドウでも編集でき、どちらかのウィンドウをスクロールすると、他方のウィンドウも追従してスクロールします。

Follow モードでは、一方のウィンドウで表示されている部分の外にポイントを移動して、もう一方のウィンドウで表示されている部分にポイントを移動させると、そのウィンドウが選択されます。つまり 2 つのウィンドウを 1 つの大きなウィンドウとして扱えるのです。

Follow モードをオフにするには、もう一度 M-x follow-modeとタイプしてください。

11.8 テキストのフェイス

Emacs はフェイス *(faces)* と呼ばれる仕組みを通じて、テキストをいくつかの異なるスタイルで表示できます。フェイスには font(フォント)、height(高さ)、weight(太さ)、slant(傾き)、foreground(前景) および background(背景)、underline(アンダーライン)、overline(オーバーライン) などの様々なフェイス属性 *(face attributes)* を指定できます。ほとんどのメジャーモードは Font Lock モードを通じて、テキストに自動的にフェイスを割り当てます。これらのフェイスを割り当てる方法については、Section 11.13 [Font Lock], page 89 を参照してください。

現在定義されているフェイスと、それがどのような外観なのかを見るには、`M-x list-faces-display`とタイプします。プレフィクス引数を指定すると、このコマンドは正規表現の入力を求め、その正規表現にマッチするフェイスだけを表示します (Section 12.6 [Regexps], page 115 を参照してください)。

あるフェイスが、フレームが異なると違って見えるのことがあり得ます。たとえばいくつかのテキスト端末ではすべてのフェース属性、特に特に font、height、width はサポートされておらず、指定できる color も限られているものがあります。加えて、ほとんどの Emacs フェースは視認性をよくするために、フレームのバックグラウンドが明るい (light) か暗い (dark) かで属性が異なります。デフォルトでは、Emacs はフレームの現在のバックグラウンドカラーに基づいて、表示するフェイスの属性を自動的に選択します。しかし変数`frame-background-mode`に非`nil`値を与えると、これをオーバーライドできます。値`dark`ではすべてのフレームの背景色が暗い色であるかのように処理し、値`light`ではすべてのフレームの背景色が明るい色であるかのように処理させることができます。

フェイスの属性を変えてフェイスをカスタマイズして、将来の Emacs セッション用にカスタマイズ結果を保存することができます。詳細については、Section 33.1.5 [Face Customization], page 503 を参照してください。

`default`フェイスはテキストを表示するデフォルトのフェイスで、そのすべての属性は指定されています。バックグラウンドカラーは、フレームのバックグラウンドカラーとしても使用されます。Section 11.9 [Colors], page 83 を参照してください。

他の特別なフェイスとしては、`cursor`フェイスがあります。グラフィカルなディスプレイでは、このフェイスのバックグラウンドカラーは、テキストカーソルを描画するのに使用されます。このフェイスで効果があるのはこの属性だけです。カーソルの下のテキストのフォアグラウンドカラーには、そのテキストのバックグラウンドカラーが使われます。テキスト端末でのテキストカーソルの外観は、`cursor`フェイスではなく端末により決定されます。

特定のフェイスの属性を指定するのに X のリソースを使うこともできます。Section D.1 [Resources], page 591 を参照してください。

Emacs は可変幅フォント (variable-width fonts) を表示できますが、いくつかのコマンド、特にインデントを行うコマンドは、可変幅の文字幅の表示をうまく処理できません。そのため、ほとんどのフェイスにたいして可変幅フォントを使わないこと、特にそれが Font Lock モードに割り当てられている場合は、使わないことを推奨します。

11.9 フェイスのカラー

フェイスには、さまざまなフォアグラウンドカラーとバックグラウンドカラーをもたせることができます。フェイスにカラーを指定するとき、たとえばフェイスをカスタマイズ (Section 33.1.5 [Face Customization], page 503 を参照してください) するときは、カラーネーム *(color name)* か、*RGB* トリプレット *(RGB triplet)* で指定することができます。

11.9.1 カラー名

カラーネームとは、‘`dark orange`’や‘`medium sea green`’のような、事前に定義された名前です。カラーネームの一覧を見るには、`M-x list-colors-display`とタイプします。表示されるカラーの順番を制御するには、`list-colors-sort`をカスタマイズします。このコマンドをグラフィカルなディスプレイで実行すると、Emacs で既知のカラーネームのすべてが表示されます (これらは標準の X11 のカラーネームで、X の`rgb.txt`で定義されています)。コマンドをテキスト端末で実行すると、端末で安全に表示することができる一部のカラーだけが表示されます。フェイスには、さまざまなフォアグラウンドカラーとバックグラウンドカラーを持たせることができます。しかし Emacs は、テキ

スト端末でも X11 のカラーネームを理解できます。もしフェイスに X11 のカラーネームが指定されている場合、最も近い端末の色で表示されます。

11.9.2 RGB トリプレット

RGB トリプレットは '`#RRGGBB`'という形式の文字列で指定します。主要カラーコンポーネントはそれぞれ '`00`' (色強度 0) から '`FF`' (最大強度) の 16 進数として表現されます。各コンポーネントにたいして 1 桁、3 桁、あるいは 4 桁の 16 進数を使用することもできるので '`red`'は '`#F00`'、'`#fff000000`'、あるいは '`#ffff00000000`'と表すことができます。各コンポーネントは同じ桁数でなければなりません。16 進数の A から F は、大文字小文字を区別しません。

`M-x list-colors-display`は、カラーネームと、それに相当する RGB トリプレットを表示します。たとえば '`medium sea green`'は '`#3CB371`'と同じです。

`M-x set-face-foreground`と `M-x set-face-background`で、フェイスのフォアグラウンドとバックグラウンドのカラーを変更できます。これらのコマンドは、ミニバッファーでフェイス名とカラーの入力を求め (補完機能あり)、指定したカラーをフェイスにセットします。フェイスのカラーは全フレームに影響しますが、カスタマイズバッファーや X リソースを使うのとは異なり、将来の Emacs セッションには引き継がれません。フレームパラメーターを使って、特定のフレームのフォアグラウンドとバックグラウンドのカラーをセットすることもできます。Section 18.11 [Frame Parameters], page 206 を参照してください。

11.10 標準フェイス

以下はテキストの外見を指定する標準フェイスです。これらのフェイスの効果が欲しい場合は、特定のテキストに適用することができます。

`default` このフェイスは特定のフェイスをもたない普通のテキストに使われます。フェイスのバックグラウンドカラーは、フレームのバックグラウンドカラーとして使用されます。

`bold` このフェイスは、デフォルトフォントの bold(太字) バージョンです。

`italic` このフェイスはデフォルトフォントの italic(斜体) バージョンです。

`bold-italic`
このフェイスはデフォルトフォントの bold italic(太字斜体) バージョンです。

`underline`
このフェイスは underline(下線) のテキストです。

`fixed-pitch`
このフェイスは fixed-width font(固定幅フォント) の使用を強制します。もし望むなら、このフェイスから他の固定幅フォントにカスタマイズするのは妥当ですが、可変幅フォントにするべきではありません。

`fixed-pitch-serif`
このフェイスは `fixed-pitch`と似ていますが、フォントはセリフ (serif: H や I などの上下のひげ飾り) をもち、伝統的なタイプライター文字に、より似ています。

`variable-pitch`
このフェイスは可変幅のフォント (プロポーショナルフォント) の使用を強制します。このフォント用に選択されたフォントのサイズは、デフォルトのフォント (通常は固定幅) として選択されたフォントのサイズと一致します。

`variable-pitch-text`
: これは `variable-pitch` (継承元) と似ていますが、若干大きめのフェイスです。プロポーショナルフォントは通常は同じ高さのモノスペースフォントよりも視覚的に小さい外観になるので、読み難くなる可能性があります。長いテキストを表示する際には、(小さめの)`variable-pitch`フェイスよりも、こちらのフェイスのほうが良い選択かもしれません。

`shadow`
: このフェースはまわりのテキストに比べて、そのテキストを目立たなくします。通常これはデフォルトの黒または白のフォアグラウンドカラーではなく、グレーが使われます。

以下は特別な目的のために、一時的にテキストの一部をハイライトするのに使われるフェイスの、不完全なリストです (他にも多くのモードが、そのモードの目的のために、独自のフェイスを定義しています)。

`highlight`
: このフェイスはさまざまなコンテキスト、たとえばハイパーリンク上をマウスカーソルが通過したときなどに、テキストをハイライトするのに使われます。

`isearch`
: このフェイスは、現在の Isearch(インクリメンタル検索) のマッチをハイライトするのに使われます (Section 12.1 [Incremental Search], page 105 を参照してください)。

`query-replace`
: このフェイスは、現在の問い合わせ置換 (Query Replace) のマッチをハイライトするのに使われます (Section 12.10 [Replace], page 122 を参照してください)。

`lazy-highlight`
: このフェイスは、Isearch および問い合わせ置換で、カレントのマッチ (現在カーソルがあるマッチ) 以外のマッチ (lazy matches) をハイライトするのに使われます。

`region`
: このフェイスは、アクティブなリージョンを表示するのに使われます (Chapter 8 [Mark], page 52 を参照してください)。Emacs を GTK+サポートつきでビルドした場合、カラーは現在の GTK+のテーマから提供されます。

`secondary-selection`
: このフェイスは、X のセカンダリー選択 (secondary X selection) を表示するのに使われます (Section 9.3.3 [Secondary Selection], page 66 を参照してください)。

`trailing-whitespace`
: このフェイスは、`show-trailing-whitespace`は非`nil`のとき、行末の余分なスペースやタブをハイライトするためのものです (Section 11.17 [Useless Whitespace], page 95 を参照してください)。

`escape-glyph`
: このフェイスは、制御文字やエスケープシーケンスを表示するためのものです (Section 11.20 [Text Display], page 98 を参照してください)。

`homoglyph`
: このフェイスは、類似文字 (表示しようとする文字と似ているが異なる文字) を表示するためのものです (Section 11.20 [Text Display], page 98 を参照)。

`nobreak-space`
: このフェイスは、no-break スペース文字を表示するためのものです (Section 11.20 [Text Display], page 98 を参照してください)。

`nobreak-hyphen`
: このフェイスは、no-break ハイフン文字を表示するためのものです (Section 11.20 [Text Display], page 98 を参照してください)。

以下のフェイスは、Emacs フレームの一部の外見を制御します:

`mode-line`
: これはモードラインやヘッダーライン、およびツールキットのメニューを使っていない際のメニューバーに使用される基本フェイスです。デフォルトでは、グラフィカルなウィンドウでは raised(浮き彫り) 効果をだすため影つきで描画され、非ウィンドウの端末ではデフォルトのフェイスを反転して描画されます。

  (モードラインに使用されている)`mode-line-active`と `mode-line-inactive`はこのフェイスから派生したフェイスです。

`mode-line-active`
: `mode-line`と似ていますが、カレントで選択されているウィンドウのモードラインに使用されます。このフェイスは `mode-line`を継承するので、フェイスを変更するとすべてのウィンドウのモードラインが影響を受けます。

`mode-line-inactive`
: `mode-line`と似ていますが、選択されていないウィンドウのモードラインに使われます (`mode-line-in-non-selected-windows`が非 `nil`のとき)。このフェイスは `mode-line`を継承するので、フェイスを変更するとすべてのウィンドウのモードラインが影響を受けます。

`mode-line-highlight`
: `highlight`と似ていますが、モードライン上でマウスセンシティブ (マウスに感応する) なテキスト範囲に使われます。通常このようなテキスト範囲は上にマウスポインターがくると、ツールチップ (Section 18.19 [Tooltips], page 213 を参照してください) をポップアップします。

`mode-line-buffer-id`
: このフェイスは、モードライン上でバッファーを識別する部分に使われます。

`header-line`
: `mode-line`と似ていますが、ウィンドウのヘッダーラインのためのものです。モードラインがウィンドウの一番下に表示されるように、ヘッダーラインはウィンドウの一番上に表示されます。ほとんどのウィンドウはヘッダーラインを持ちません。Info モードのような特別なモードだけがヘッダーラインを持ちます。

`header-line-highlight`
: `highlight`や `mode-line-highlight`と似ていますが、ヘッダー行のマウスに感応する部分に使用されます。`header-line`フェイスは `highlight`とは無関係にカスタマイズされるかもしれないので、このフェイスが別に設けられています。

`tab-line` `mode-line`と似ていますが、ウィンドウのタブラインのためのものです。タブラインはウィンドウのバッファーを表し、ウィンドウの上端に表示されます。Section 17.8 [Tab Line], page 193 を参照してください。

`vertical-border`
: このフェイスは、テキスト端末上でウィンドウを縦に分割するとき使われます。

`minibuffer-prompt`
: このフェイスは、ミニバッファーで入力を求めるプロンプトのテキストに使われます。デフォルトでは、Emacsは自動的にプロンプトのテキストの、テキストプロパティ(Section "Text Properties" in *the Emacs Lisp Reference Manual* を参照してください) のリスト`minibuffer-prompt-properties`に、このフェイスを追加します(この変数はミニバッファーに入ったときに効果をあらわします)。

`fringe`
: グラフィカルなウィンドウでの、左右のフリンジのためのフェイスです(フリンジはEmacsフレームで、テキストエリアとウィンドウの左右の境界線の間にある、狭い領域です)。Section 11.15 [Fringes], page 93 を参照してください。

`cursor`
: このフェイスの`:background`属性は、テキストカーソルのカラーを指定します。Section 11.21 [Cursor Display], page 99 を参照してください。

`tooltip`
: このフェイスは、ツールチップのテキストに使われます。デフォルトでは、Emacsが GTK+サポートつきでビルドされた場合、ツールチップはGTK+を通じて描画されるので、このフェイスは効果がありません。Section 18.19 [Tooltips], page 213 を参照してください。

`mouse`
: このファイスは、マウスポインターのカラーを決定します。

以下のフェイスは、Emacsフレームの一部の外見を制御するときと同様ですが、テキスト端末またはEmacsをXサポートつき (ただしツールキットサポートなし) でビルドしたときだけ使われます(それ以外の場合、フレームの対応する各要素は広義なシステム設定により決定されます)。

`scroll-bar`
: このフェイスは、スクロールバーの外見を決定します。Section 18.12 [Scroll Bars], page 206 を参照してください。

`tool-bar`
: このフェイスは、ツールバーのアイコンのカラーを決定します。Section 18.16 [Tool Bars], page 209 を参照してください。

`tab-bar`
: このフェイスはタブバーのアイコンのカラーを決定します。Section 18.17 [Tab Bars], page 210 を参照してください。

`menu`
: このフェイスはEmacsメニューのカラーとフォントを決定します。Section 18.15 [Menu Bars], page 209 を参照してください。

`tty-menu-enabled-face`
: このフェイスは、テキスト端末で利用可能なメニューアイテムを表示するのに使われます。

`tty-menu-disabled-face`
: このフェイスは、テキスト端末で利用不可なメニューアイテムを表示するのに使われます。

`tty-menu-selected-face`
: このフェイスは、テキスト端末でマウスをクリックするか、`RET`を押せば選択できるメニューアイテムを表示するのに使われます。

11.11 アイコン

Emacsはクリックできるボタン (あるいは情報を提供するためのアイコン) を表示するときがあり、これらの表示のされかたをカスタマイズできます。

ここでのカスタマイズにおける主要ポイントはユーザーオプション`icon-preference`です。これを使うことによって、アイコンにたいするあなたの全般的な嗜好を伝えることができるのです。こ

のユーザーオプションはアイコンのタイプのリストで、サポートされている最初のアイコンタイプが使用されます。サポートされているタイプは:

`image` アイコンにイメージを使用します。

`emoji` アイコンにカラフルな絵文字を使用します。

`symbol` アイコンに白黒のシンボルを使用します。

`text` アイコンにシンプルなテキストを使用します。

更に個々のアイコンを `M-x customize-icon`でカスタマイズでき、テーマによってアイコンの見栄えを大きく変えることができます。

あるアイコンについての説明を手早く入手するには、`M-x describe-icon`コマンドを使用してください。

11.12 テキストのスケール

カレントバッファーの `default`フェイスのフォントサイズを大きくするには、`C-x C-+`または `C-x C-=`、小さくするには `C-x C--`をタイプします。デフォルトのフォントサイズ (グローバル) に復元するには、`C-x C-0`とタイプします。これらのキーは、すべて同じコマンド `text-scale-adjust`にバインドされています。このコマンドは最後にタイプされたキーを判断してどの処理を行うかを判断、処理に応じてデフォルトフェイスの高さを変更してフォントサイズを調節します。

多くのフェイスの`:height`属性には明確なセッティングがされておらず、高さを `default`フェイスから継承するものがほとんどです。したがって上記のコマンドによってこれらのフェイスもスケーリングされます。

`default`以外で`:height`属性に明確なセッティングをもつフェイスでは、これらのフォントサイズの影響を受けません。ただし `header-line`フェイスは例外です。たとえ`:height`属性に明確なセッティングがあっても、このフェイはスケーリングされます。

同様にマウスポインターがバッファーテキスト上にある際に `Ctrl`修飾キーを押下してマウスホイールをスクロールすると、スクロール方向に応じて影響を受けるフェイスのフォントサイズを増加あるいは減少します。

これらのコマンドの最後のキーは、`C-x`を前置せず修飾キーなしで繰り返すことができます。たとえば `C-x C-= C-= C-=`と `C-x C-= = =`は、どちらもフェイスの大きさを 3 段階に大きくします。各ステップで大きくなる倍率は 1.2 です。この倍率を変更するには、変数 `text-scale-mode-step`をカスタマイズします。`text-scale-adjust`コマンドに数引数 0 を指定すると、`C-x C-0`とタイプしたのと同様に、デフォルトの大きさに復元します。

同様にフォントサイズをグローバルに変更するには `C-x C-M-+`、`C-x C-M-=`、`C-x C-M--`、`C-x C-M-0`、あるいは修飾キーの `Ctrl`と `Meta`を両方押したままマウスホイールをスクロールしてください。フォントのサイズがグローバルに変更された際のフレーズのリサイズを有効にするには、変数 `global-text-scale-adjust-resizes-frames`をカスタマイズしてください (Section 33.1 [Easy Customization], page 499 を参照)。

コマンド `text-scale-increase`と `text-scale-decrease`は、`C-x C-+`や `C-x C--`と同じようにカレントバッファーのフォントサイズを増加または減少させます。キーをバインドする場合には、これらのコマンドのほうが `text-scale-adjust`より便利でしょう。

コマンド `text-scale-set`は、数引数でカレントバッファーのフォントサイズを絶対倍率で指定します。

上記のコマンドは、現在のフォント倍率が 1 以外のときは、自動的にマイナーモード `text-scale-mode`を有効にし、そうでない場合は無効にします。

コマンド `text-scale-pinch`はタッチパッドに 2 本の指を置いて、互いに近づけたり離したりというピンチジェスチャーを行った際に、指の間の距離にもとづいてテキストのサイズを拡大あるいは縮小します。このコマンドはサポートされているハードウェアをもつ一部のシステムでのみ利用できます。

コマンド `mouse-wheel-text-scale`でもテキストのスケールの変更ができます。このコマンドは、通常は `Ctrl`を押したままマウスホイールを動かすことにより実行されます。テキストのスケーリングはホイールを下方に動かすと拡大、上方に動かすと縮小されます。

11.13 Font Lock モード

Font Lock モードはマイナーモードで、常に特定のバッファーにローカルで、バッファーのテキストにフェイスを割り当てます (またはフォント表示化 *(fontifies)* します)。各バッファーのメジャーモードは、Font Lock モードにどのテキストをフォント表示可するか指示します。たとえばプログラム言語のモードは、コメントや文字列、関数名のような、構文に関連する構成をフォント表示化します。

Font Lock モードは、それをサポートするメジャーモードでは、デフォルトで有効になっています。カレントバッファーでこれを切り替えるには、`M-x font-lock-mode`とタイプします。正の数引数は無条件に Font Lock モードを有効にし、負または 0 の数引数を指定すると無効になります。

`M-x global-font-lock-mode`とタイプすると、すべてのバッファーで Font Lock モードを切り替えます。このセッティングを将来の Emacs セッションに引き継ぐには、変数 `global-font-lock-mode`をカスタマイズ (Section 33.1 [Easy Customization], page 499 を参照してください) するか、以下の行を init ファイルに追加します。

```
(global-font-lock-mode 0)
```

Global Font Lock モードを無効にしていたとしても、モードフック (mode hooks) に関数を追加することにより、特定のメジャーモードで Font Lock モードを有効にできます。たとえば C ファイルの編集で Font Lock モードを有効にするには、以下のように記述します:

```
(add-hook 'c-mode-hook 'font-lock-mode)
```

Font Lock モードは、`font-lock-string-face`、`font-lock-comment-face`のような、いくつかの特別な名前のフェイスを使って処理を行います。これらすべてを簡単に探す方法には、`M-x customize-group RET font-lock-faces RET`を使います。それからカスタマイズバッファーでそれらのフェイスの外見をカスタマイズできます。Section 33.1.5 [Face Customization], page 503 を参照してください。

非常に大きいバッファーのフォント表示化には、長い時間を要することもあります。ファイルを visit したとき大きな遅延を避けるには、Emacs が最初はバッファーの表示された部分だけをフォント表示化するようにします。バッファーをスクロールすると、新たに表示される部分がフォント表示化されます。このタイプの Font Lock は、*Just-In-Time*(または *JIT*) Lock と呼ばれます。カスタマイズグループ‘`jit-lock`’の値をカスタマイズすることにより、アイドル状態のときにフォント表示可を行うことも含めて、JIT Lock がどのように振る舞うか制御できます。Section 33.1.6 [Specific Customization], page 505 を参照してください。

バッファーのどの部分のテキストをフォント表示するか、そしてどのフェイスを使用するかをメジャーモードが判断するために用いるのは、異なる複数のテキスト分析手法にもとづく情報かもしれません:

- 正規表現によるキーワードやその他テキストパターンの検索 (Section 12.5 [Regular Expression Search], page 114 を参照)。

- 組み込み構文テーブルにもとづき構文的に異なる部分をテキストから検索 (Section “Syntax Tables” in *The Emacs Lisp Reference Manual* を参照)。
- tree-sitter ライブラリーのような特殊用途用ライブラリーや外部プログラムを通じて、本格的パーサーが生成した構文木を使用する (Section “Parsing Program Source” in *The Emacs Lisp Reference Manual* を参照)。

11.13.1 従来の Font Lock

フォントロック情報を提供するための “伝統的” な手法は正規表現検索、および Emacs に組み込まれた構文テーブルを用いた構文解析にもとづいています。このサブセクションでは、これら伝統的手法を用いるメジャーモードでのフォントロックの用法とカスタマイズについて説明します。

変数 `font-lock-maximum-decoration`をカスタマイズして、この機能をサポートするメジャーモードにたいして、Font Lock モードで適用されるフォント表示化の量を制御できます。この変数の値には数字を指定します (1 は最小限のフォント表示化で、3 という高いレベルのモードもあります)。`t`は “可能な限り高く” という意味です (デフォルト)。効果を得るには、そのファイルが visit される前に `font-lock-maximum-decoration`をカスタマイズするべきです。この変数をカスタマイズする時点で、すでにそのファイルをバッファーで visit している場合は、そのバッファーを kill してから、スタマイズ後に再度そのファイルを visit してください。

特定のモードに異なる数字を指定することもできます。たとえば C/C++モードにはレベル 1 を指定して、他のモードにはデフォルトのレベルを適用するには、以下の値を使います

```
'((c-mode . 1) (c++-mode . 1)))
```

コメントと文字列のフォント表示化 (または “構文的” なフォント表示化) は、バッファーのテキストの構文構造の解析に依存します。速度向上のため、Lisp モードを含めたいくつかのモードでは、特別な慣習に依存しています。たとえば一番左の列の開きカッコ (open-parenthesis) または開き大カッコ (open-brace) は常に defun の開始であり、すなわち常に文字列またはコメントの外部にあるとみなす、というように解析します。したがって文字列やコメントの中で、一番左の列に開きカッコや開き大カッコを記述するのは避けるべきです。詳細については、Section 23.2.1 [Left Margin Paren], page 286 を参照してください。

Font Lock は、ほとんどのモードで既存のパターンをハイライトしますが、追加のパターンをフォント表示化したいときもあるでしょう。特定のモードでハイライトするパターンを追加するには、関数 `font-lock-add-keywords`を使うことができます。たとえば C コメント中の ‘`FIXME:`’という単語をハイライトするには、以下を使います:

```
(add-hook 'c-mode-hook
          (lambda ()
           (font-lock-add-keywords nil
            '(("\\<\\(FIXME\\):" 1
               font-lock-warning-face t)))))
```

font-lock のハイライトパターンからキーワードを削除するには、関数 `font-lock-remove-keywords`を使います。Section “Search-based Fontification” in *The Emacs Lisp Reference Manual* を参照してください。かわりに `font-lock-ignore`オプションをカスタマイズすれば、いくつかのキーワードによって選択的にハイライトを無効にすることもできます。Section “Customizing Keywords” in *The Emacs Lisp Reference Manual* を参照してください。

11.13.2 パーサーベースの Font Lock

Emacs を tree-sitter ライブラリーとともにビルドした場合には、フォント表示化に際してそのライブラリーがバッファーのテキストをパースした結果を使うことができます。tree-sitter ライブラリーは

サポートしているプログラミング言語やその他のフォーマットされたテキストにたいして本格的なパーサーを提供するので、通常は前のサブセクションで説明した “伝統的” な手法より高速かつ正確です。tree-sitter ライブラリーを利用するメジャーモードには *foo*`-ts-mode`という名前がつけられています (‘`-ts-`’の部分がそのライブラリーを使用することを示す)。このサブセクションでは、tree-sitter ライブラリーにもとづく Font Lock サポートについて説明します。

変数`treesit-font-lock-level`をカスタマイズして、tree-sitter にもとづくメジャーモードの Font Lock モードに適用するフォント表示化の量を制御できます。変数の値は 1 から 4 の数値です:

Level 1 このレベルでは、通常はコメントおよび関数定義の関数名だけをフォント表示します。

Level 2 このレベルではキーワード、文字列、データタイプへのフォント表示化が追加されます。

Level 3 デフォルトのレベルです。割り当て、数値等のフォント表示化が追加されます。

Level 4 このレベルでは演算子、区切り文字、カッコ、その他の句読点文字、関数呼び出し内の関数名、プロパティ参照、変数などフォント表示化が可能なすべてが追加されます。

上記で示した構文的カテゴリーそれぞれが正確には何によって構成するかはメジャーモード、およびそのメジャーモードの言語にたいして tree-sitter が使用するパーサーグラマー (parser grammar: 解析文法) に依存します。とはいうものの、このカテゴリーは一般的にはそのメジャーモードによってサポートされるプログラミング言語やファイルフォーマットの規約にしたがいます。変数`treesit-font-lock-feature-list`のバッファーローカル値には、tree-sitter ベースのメジャーモードがサポートしているフォント表示化機能が保持されています。このリストの要素は、その要素が対応するフォント表示化レベルが提供する機能を示すリストです (訳注: GNU Emacs のソースに含まれる admin/notes/tree-sitter/starter-guide に値の例があるので参考にしてください)。

`M-x customize-variable` (Section 33.1.6 [Specific Customization], page 505 を参照) を通じて`treesit-font-lock-level`の値を変更すれば既存のすべてのバッファー、および同一セッションにおいて今後 visit するすべてのファイルに即座に効果が表れます。

11.14 インタラクティブなハイライト

Highlight Changes モードは、最近変更されたバッファー部分のテキストに、異なるフェイスを与えることによりハイライトするマイナーモードです。Highlight Changes モードを有効または無効にするには、`M-x highlight-changes-mode`を使います。

Hi Lock モードは、指定した正規表現にマッチするテキストをハイライトする、マイナーモードです。たとえば、プログラムのソースファイルで、特定の変数へのすべての参照をハイライトしたり、何らかのプログラムの大量の出力の一部をハイライトしたり、記事中の特定の名前をハイライトするために使用できます。Hi Lock モードを有効または無効にするには、コマンド`M-x hi-lock-mode`を使います。すべてのバッファーで Hi Lock モードを有効にするには、`M-x global-hi-lock-mode`を使うか、`.emacs`ファイルに`(global-hi-lock-mode 1)`と記述してください。

Hi Lock モードは Font Lock モード (Section 11.13 [Font Lock], page 89 を参照してください) と同じように動作しますが、ハイライトするパターンを明示的に正規表現で指定します。これらは以下のコマンドで制御できます (`C-x w`で始まるキーバインドは、`M-s h`で始まるグローバルなバインドが優先されるため推奨されておらず、将来の Emacs のバージョンで廃止されるでしょう)。

`M-s h r` *regexp* `RET` *face* `RET`
`C-x w h` *regexp* `RET` *face* `RET`
*regexp*にマッチするテキストを、フェイス *face*を使ってハイライトします (`highlight-regexp`)。ハイライトはバッファーがロードされている限り残ります。たとえば単語 “whim” をデフォルトのフェイス (黄色いバックグラウンドカラー) でハ

イライトするには、M-s h r whim RET RETとタイプします。ハイライトには任意のフェイスを使うことができますが、Hi Lock モードはモード自身でいくつかのフェイスを提供しており、それらはデフォルト値のリストに事前ロードされています。フェイスの入力プロンプトで M-nと M-pを使うことにより、それらを巡回することができます。プレフィクス数引数は対応する部分式にハイライトを制限します。

オプション hi-lock-auto-select-faceに非 nil値をセットすることにより、このコマンド (およびその他のフェイスを読みとる Hi Lock コマンド) は、入力を求めることなく、デフォルト値のリストから次のフェイスを自動的に選択します。

このコマンドを複数回使用して、さまざまな正規表現を指定し、それぞれを異なる方法でハイライトできます。

M-s h u *regexp* RET
C-x w r *regexp* RET
*regexp*のハイライトを解除します (unhighlight-regexp)。メニューから呼び出した場合、ハイライト解除する正規表現をリストから選択します。キーボードから呼び出した場合は、ミニバッファーを使います。一番最近追加された正規表現を表示し、M-nを使って次に古い正規表現、M-pで次に新しい正規表現を表示できます (手入力もでき、その場合は補完機能つきです)。ハイライト解除したい正規表現がミニバッファーに表示されたら、RETを押してミニバッファーを抜けだし、ハイライトを解除できます。

M-s h l *regexp* RET *face* RET
C-x w l *regexp* RET *face* RET
*regexp*とのマッチを含む行全体を、フェイス *face*を使ってハイライトします (highlight-lines-matching-regexp)。

M-s h p *phrase* RET *face* RET
C-x w p *phrase* RET *face* RET
*phrase*にマッチするフレーズを、フェイス *face*でハイライトします (highlight-phrase)。*phrase*には正規表現を指定できますが、スペースは空白文字にマッチする正規表現に置き換えられます。また、先頭に小文字を使用することにより、大文字小文字を区別しなくなります。

M-s h .
C-x w . ポイントの近くで見つかったシンボルを、次に利用可能なフェイスでハイライトします (highlight-symbol-at-point)。

M-s h w
C-x w b 現在ハイライトを行っている正規表現/フェイスのペアを、バッファーのポイント位置に挿入します。挿入はプログラムを変更してしまわないように、コメント文字列でコメント化されます (このキーバインドは hi-lock-write-interactive-patternsコマンドを実行します)。

これらのパターンは、コメントからも逆抽出されます。それは、コメントに記述されたテキストが適正で、M-x hi-lock-find-patternsを呼び出した、あるいは Hi Lock モードが有効なときファイルを visit(これは hi-lock-find-patternsを実行します) したときです。

M-s h f
C-x w i 正規表現/フェイスのペアを、現在のバッファーのコメントから抽出します (hi-lock-find-patterns)。これらのコマンドを使えば、highlight-regexpでパターンを対話的に入力、hi-lock-write-interactive-patternsでそれをファイルに保存、そ

れらを編集 (あるマッチのフェイスを別のフェイスにしたり)、そして最後にこのコマンド (`hi-lock-find-patterns`) で編集済みのパターンを、Hi Lock のハイライトに適用することができます。

変数 `hi-lock-file-patterns-policy`はファイルを visit したとき、Hi Lock モードがパターンを探して、それを自動的に抽出するべきかを制御します。値には`nil`(ハイライトしない)、`ask`(ユーザーに尋ねる)、または関数を指定します。関数の場合、`hi-lock-find-patterns`はパターンを引数としてその関数を呼び出します。関数が非`nil`を返した場合、パターンを使用します。デフォルトは`ask`です。直接`hi-lock-find-patterns`を呼び出した場合、この変数の値に関係なく、常にパターンはハイライトされることに注意してください。

現在のメジャーモードのシンボルが、リスト `hi-lock-exclude-modes`のメンバーの場合、`hi-lock-find-patterns`は何もしません。

11.15 ウィンドウのフリンジ

グラフィカルなディスプレイでは、通常 Emacs の各ウィンドウの左右の端に、狭いフリンジ *(fringes:* 縁、へり*)* があります。フリンジは、ウィンドウのテキストに関する情報を提供するシンボルの表示に使用されます。`M-x fringe-mode`とタイプしてフリンジ表示を切り替えたり、幅を変更できます。このコマンドは全フレームのフリンジに影響します。選択されたフレームのフリンジだけを変更するには、`M-x set-fringe-style`を使います。変数 `fringe-mode`をカスタマイズして、フリンジへの変更を永続化できます。

フリンジのもっとも一般的な使われかたは、継続行の表示です (Section 4.8 [Continuation Lines], page 23 を参照してください)。テキストの 1 行が複数のスクリーン行に分割されるとき、最初の行を除いた各行の左フリンジには曲矢印が表示され、その行の先頭が実際の行頭ではないことを示します。そして、最後の行を除いた各行の右フリンジにも曲矢印が表示され、その行の最後が実際の行末ではないことを示します。行の方向が右から左 (Section 19.20 [Bidirectional Editing], page 238 を参照してください) の場合、フリンジの曲矢印の意味は逆になります。

行が切り詰められているとき (Section 11.22 [Line Truncation], page 100 を参照してください) は水平方向の直矢印を表示して、この行には水平スクロールしなければ見ることのできないテキストがあることを示します。矢印の上でマウスをクリックすれば、矢印の指す方向に水平スクロールします。

フリンジはバッファーの境界 (Section 11.16 [Displaying Boundaries], page 94 を参照) やウィンドウ下端付近の使用されていない行、デバッグ (Section 24.6 [Debuggers], page 315 を参照) しているプログラムが実行中であることを示すためにも使われます。

現在の行がウィンドウの幅と正確に一致して、ポイントがその行の行末にある場合、フリンジにはカーソルが描画されます。これを無効にするには、変数 `overflow-newline-into-fringe`を `nil` に変更します。これにより Emacs はウィンドウ幅と同じ長さの行にたいしても、継続または切り詰めを行います。

表示されているウィンドウの片側、または両側のフリンジを削除するために`fringe-mode`をカスタマイズする場合、フリンジ上にに表示する機能は利用できなくなりますが、行の継続と切り詰めの標識は例外です。フリンジが利用できない場合、Emacs は特別な ASCII 文字 (Section 4.8 [Continuation Lines], page 23、および Section 11.22 [Line Truncation], page 100 を参照してください) により、行の継続と切り詰めを示すために、最左および最右の文字セルを使用します。行の継続と切り詰めの標識に使用される文字セルは、この目的のために予約されるので、各行に表示するテキストのための列数は減少します。バッファーのテキストには双方向のテキスト、および left-to-right(左から右) と right-to-left(右から左) の両方のパラグラフ (Section 19.20 [Bidirectional Editing], page 238 を参照してください) が含まれるかもしれないので、、片側のフリンジを削除しただけでは依然として

2 つの文字セルが予約されます。つまり行の継続と切り詰めの標識のために、ウィンドウの両側にそれぞれ 1 つの文字セルが予約されます。なぜなら、right-to-left のパラグラフでは、これらの標識はウィンドウの反対側に表示されるからです。

11.16 バウンダリーの表示

Emacs に `fill-column`の位置 (Section 22.6.2 [Fill Commands], page 256 を参照) を表示するインジケーターを追加できます。フィル列インジケーターは特に `prog-mode`とその派生モード (Section 20.1 [Major Modes], page 240 を参照) においてプログラムのソースコードのフォーマットにたいして特別な意味をもつ特定の列位置を示すために有用な機能です。これは等幅フォント (fixed-pitch font) のすべての文字を想定した機能です。ここで等幅フォントとはすべての文字の幅が等しいフォントのことです (全角文字は例外かもしれない)。可変幅フォント (variable-pitch font) を使用バッファーでは、行が異なればフィル列インジケーターは不揃いに表示されるかもしれません。

フィル列インジケーターの表示を有効にするには、インジケーターをローカルまたはグローバルに有効にする、`display-fill-column-indicator-mode`または`global-display-fill-column-indicator-mode`のマイナーモードを使用します。

かわりにインジケーターの有効化とインジケートに使用する文字の制御に、`display-fill-column-indicator`と `display-fill-column-indicator-character`という 2 つのバッファーローカル変数セットできます。インジケーターを表示するためには、これらの変数がどちらも非 `nil`でなければならないことに注意してください (マイナーモードをオンにすることによりこれらの変数がセットされる)。

このモードをカスタマイズするために 2 つのバッファーローカル変数と 1 つのフェイスがあります:

`display-fill-column-indicator-column`
: インジケーターをセットするべき列数を指定します。列にたいする正の数値、あるいは変数 `fill-column`の値の使用を意味する `t`を指定できます。

  それ以外の値ではインジケーターは無効になります。デフォルト値は `t`です。

`display-fill-column-indicator-character`
: インジケーターに使用する文字を指定します。フォントがサポートすれば、Unicode 文字を含む任意の文字を指定できます。値 `nil`はインジケーターを無効にします。`display-fill-column-indicator-mode`や `global-display-fill-column-indicator-mode`でモードを有効にしたときは、値が非 `nil`ならばこの変数が指定する文字が使用されます。それ以外なら Emacs は文字 U+2502 BOX DRAWINGS LIGHT VERTICAL、U+2502 が表示できなければフォールバックとして '|'を使用します。

`fill-column-indicator`
: インジケーターの表示に使用するフェイスを指定します。これはバックグラウンドカラーを除くデフォルト値をフェイス `shadow`から継承します。インジケーターのカラーを変更するために必要なのは、このフェイスのフォアグラウンドカラーのセットだけです。

グラフィカルなディスプレイでは、Emacs はバッファーのバウンダリー (boundary: 境界) を、フリンジに表示することもできます。この機能を有効にすると、最初の行と最後の行ではフリンジに、かぎカッコが表示されます。上矢印または下矢印の場合、それはウィンドウをその方向に、もっとスクロールできることを示します。

バッファーローカルな変数 `indicate-buffer-boundaries`は、バッファーのバウンダリーととウィンドウのスクロールが、フリンジでどのように表示されるかを制御します。値が `left`(または `right`) の場合、かぎカッコと矢印のビットマップは、左フリンジ (または右フリンジ) に表示されます。

値が alist(association list: 連想リスト。Section “Association Lists” in *the Emacs Lisp Reference Manual* を参照してください) の場合、各要素の (*indicator* . *position*)で、標識 (indicator) の位置 (position) を指定します。*indicator*には `top`、`bottom`、`up`、`down`、または `t`(指定されていない標識のデフォルト位置) を指定します。*position*には `left`、`right`、または `nil`(標識を表示しない) を指定します。

たとえば `((top . left) (t . right))`は、最上行の左フリンジにかぎカッコを表示し、右フリンジには最下行のかぎカッコとスクロール矢印を表示します。左フリンジにかぎカッコだけを表示させる場合は、`((top . left) (bottom . left))`を使います。

11.17 不要なスペース

意識せずに不必要なスペースを行末に残してしまったり、バッファーの最後に空行を残してしまうことはよくあります。ほとんどの場合、そのような行末の空白文字 *(trailing whitespace)* は何の影響も及ぼしませんが、厄介物になる場合もあります。

バッファーローカルな変数 `show-trailing-whitespace`を `t`にセットすることにより、行末の空白文字を可視化できます。これにより Emacs はフェイス `trailing-whitespace`で、行末の空白文字を表示します。

この機能は行末に空白文字を含む行の、行末にポイントがあるときは適用されません。厳密にいえば、これも行末の空白文字なのですが、それを特別に表示してしまうと、新しいテキストをタイプするとき面倒です。このような特別なケースでは、ポイントの位置に表示されるカーソルより、空白文字があることが自明だからです。

`M-x delete-trailing-whitespace`とタイプすると、すべての行末の空白文字を削除します。このコマンドは、バッファー内の各行の行末にあるすべての余分なスペースと、バッファーの最後にある空行を削除します。バッファー内の空行を削除しない場合は、変数 `delete-trailing-lines`を `nil`に変更してください。リージョンがアクティブのときは、リージョン内の各行の行末の余分なスペースを削除します。

グラフィカルなディスプレイでは、Emacs はウィンドウの最後の使われていない行の左フリンジに小さなイメージを表示して、それを示すことができます。このイメージはバッファーのテキストが何も含まれていないスクリーン行に表示されるので、バッファーの最後にある空行は、このイメージが表示されないことで見分けることができます。この機能を有効にするにはバッファーローカルな変数 `indicate-empty-lines`に非 `nil`値をセットします。すべての新しいバッファーでこの機能を有効または無効にするには、この変数のデフォルト値をセットします (例 `(setq-default indicate-empty-lines t)`)

Whitespace モードはバッファーローカルなマイナーモードで、バッファー内にある多くの種類の空白文字を視覚化します。これは空白文字を特別なフェイスで描画するか、特別なグリフで表示することにより行われます。このモードを切り替えるには、`M-x whitespace-mode`とタイプします。視覚化される空白文字の種類は、リスト変数 `whitespace-style`により決定されます。`M-x whitespace-toggle-options` とタイプして、カレントバッファーでこのリスト内の個々の要素のオンとオフを切り替えることができます。以下はリストに指定できる要素の一部です (完全なリストは変数のドキュメントを参照してください)。

`face` 特別なフェイスを使った視覚化をすべて有効にします。この要素には特別な意味があります。もしこれがリストに含まれていない場合、`space-mark`、`tab-mark`、`newline-mark`を除く他の視覚化は効果がなくなります。

`trailing` 行末の空白文字をハイライトします。

`tabs` タブ文字をハイライトします。

`spaces` スペースおよび non-breaking space 文字をハイライトします。

`lines` 80 列以上の行をハイライトします。列の上限を変更するには、変数 `whitespace-line-column`をカスタマイズします。

`newline` 改行をハイライトします。

`missing-newline-at-eof`
バッファーが改行で終端されていなければ最後の文字をハイライトします。

`empty` バッファーの先頭、および/または終端の空行をハイライトします。

`big-indent`
非常に深いインデントをハイライトします。デフォルトでは、少なくとも 4 個の連続する TAB 文字と、32 個の連続するスペースからなる、任意のシーケンスがハイライトされます。これを変更するには、正規表現 `whitespace-big-indent-regexp`をカスタマイズしてください。

`space-mark`
スペースと non-breaking 文字を特別なグリフで描画します。

`tab-mark` タブ文字を特別なグリフで描画します。

`newline-mark`
改行文字を特別なグリフで描画します。

Global Whitespace モードは、すべてのバッファーで空白文字を視覚化する、グローバルなマイナーモードです。この機能を個別に切り替えるには、`M-x global-whitespace-toggle-options`を使用してください。

11.18 選択的な表示

Emacs には、与えられたレベルより多くインデントされた行を隠す機能があります。これをプログラムの概要を理解するのに使うことができます。

現在のバッファーの行を隠すには、数引数 *n*を指定して `C-x $` (`set-selective-display`) をタイプします。すると少なくとも *n*列のインデントをもつ行は、スクリーンに表示されなくなります。隠された行の存在を示すのは、表示されている行末に表示された 3 つのドット (‘...’) だけで、これは 1 行以上の行が後に隠されていることを意味します。

コマンド `C-n`および `C-p`は、隠された行が存在しないかのように、隠された行をスキップして移動します。

隠された行は依然としてバッファーに存在し、ほとんどの編集コマンドはそれらを見ることができるので、隠されな行にポイントを移動することもありえます。これが起こるとカーソルは前の行の最後、つまり 3 つのドットの後ろに表示されます。ポイントが表示されている行の行末、つまり改行の前にある場合、カーソルは 3 つのドットの前に表示されます。

隠された行のすべてを再び表示するには、引数を指定せずに `C-x $`とタイプしてください。

変数 `selective-display-ellipses`に `nil`をセットすると、隠された行があることを示す 3 つのドットは表示されなくなり、隠された行があることを示す視覚的な表示はなくなります。変数がセットされると、それは自動的にローカルになります。

バッファーのテキストの一部を隠す他の方法については、Section 22.9 [Outline Mode], page 261 を参照してください。

11.19 モードラインのオプション

バッファーのパーセント表示 *pos*は、ウィンドウの上端がバッファーのどの場所にあるかを 100 分率で示します。`M-x size-indication-mode`とタイプして Size Indication モードをオンにすることにより、バッファーのサイズを追加で表示できます。サイズは以下のようにパーセント表示のすぐ後に表示されます:

pos `of` *size*

*size*は、バッファーの文字数を人間が理解しやすい形式 (‘`k`’は 10^3、‘`M`’は 10^6、‘`G`’は 10^9 などの短縮形が使用されます) で表示します。

Line Number モードが有効なとき、ポイント位置の現在の行番号はモードラインに表示されます。`M-x line-number-mode`コマンドを使って、Line Number モードのオンとオフを切り替えることができます (通常はオンです)。行番号はそれが何であるかを示す文字 ‘`L`’とともに、バッファーのパーセント表示 *pos*の後ろに表示されます。

同様に、`M-x column-number-mode`で Column Number モードをオンにすることにより、現在の列番号を表示できます。列番号は文字 ‘`C`’で示されます。しかし両方のモードが有効になっているときは、行番号と列番号は ‘`L`’や ‘`C`’ではなく、‘`(561,2)`’のようにカッコつきで表示されます。マイナーモードとこれらのコマンドの使い方については、Section 20.2 [Minor Modes], page 241 を参照してください。

Column Number モードでは、列番号はそのウィンドウの左マージンより 0 からカウントされます。1 からカウントした列番号を表示したい場合は、`column-number-indicator-zero-based`を`nil`にセットしてください。

ナローイング (Section 11.5 [Narrowing], page 81 を参照してください) によりバッファーを制限している場合、アクセスできる部分にもとづいた行番号が表示されます。そのため、これは`goto-line`の引数として使用するには適しません (コマンド `what-line`はファイル全体にたいする相対行番号を表示する)。ナローされたバッファーのアクセス可能範囲に相対的な行へポイントを移動するには`goto-line-relative`コマンドを使用できます。

バッファーが非常に大きい場合 (`line-number-display-limit`の値より大)、速度が遅くなるので Emacs は行番号を計算しません。そのためモードラインに行番号は表示されません。この制限を取り除くには、`line-number-display-limit`に `nil`をセットします。

バッファーの行が長いときも、行番号の計算が遅くなります。この理由により、Emacs はポイントの近くの行の幅の平均文字数が、`line-number-display-limit-width`より大きいときは、行番号を表示しません。デフォルト値は 200 文字です。

Emacs はオプションで、時刻とシステムロードを、すべてのモードラインで表示できます。この機能を有効にするには、`M-x display-time`とタイプするか、オプション `display-time-mode`をカスタマイズします。モードラインに追加される情報は以下のような形式です:

hh`:`*mm*`PM` *l.ll*

ここで *hh*と *mm*は時間と分で、後ろに ‘`AM`’と ‘`PM`’がつきます。*l.ll*は、過去数分間における、システム全体で実行中または実行準備ができている (例: 利用可能なプロセッサー待ち) プロセスの平均数です (オペレーティングシステムがサポートしないフィールドは表示されません)。時刻を 24 時間表示にしたいときは、変数 `display-time-24hr-format`に `t`をセットしてください。

もし未読メールがある場合、ロードレベルの後ろに ‘`Mail`’という単語が表示されます。グラフィカルなディスプレイでは、`display-time-use-mail-icon`をカスタマイズすることにより、‘`Mail`’のかわりにアイコンを使うことができます。これによりモードラインのスペースが多少節約できます。`display-time-mail-face`をカスタマイズして、メールの表示を目立たせることができます。

display-time-mail-fileを使ってチェックするメールファイルを指定したり、display-time-mail-directoryで受信メールのディレクトリーを指定できます (ディレクトリー内の空でない普通のファイルは、新しい受信メールと判断されます)。

Emacs をラップトップコンピューターで実行している場合、コマンド display-battery-mode を使うか、変数display-battery-modeをカスタマイズすることにより、モードラインにバッテリー充電状況を表示できます。変数 battery-mode-line-formatは、バッテリーの充電状況の表示方法を決定します。モードラインに表示されるメッセージの正確さはオペレーティングシステムに依存しており、通常はバッテリーの充電トータルにたいする現在のバッテリー充電率が表示されます。battery-update-functionsの関数の実行はモードラインの更新後であり、バッテリー状況にもとづいたアクションをトリガーするために使用できます。

グラフィカルなディスプレイでは、モードラインは立体的に描画されます。この効果が気に入らない場合は、mode-lineフェイスをカスタマイズして、box属性に nilをセットすることにより無効にできます。Section 33.1.5 [Face Customization], page 503 を参照してください。

デフォルトでは、選択されていないウィンドウのモードラインは、mode-line-inactiveと呼ばれる、別のフェイスで表示されます。選択されたウィンドウのモードラインだけが、mode-lineフェイスで表示されます。これにより、どのウィンドウが選択されているかがわかりやすくなります。モードラインがないミニバッファーが選択されているときは、ミニバッファーをアクティブにしたウィンドウのモードラインが、mode-lineで表示されます。結果として通常のミニバッファーの使用では、モードラインは変化しません。

変数 mode-line-in-non-selected-windowsを nilにセットすることにより、mode-line-inactiveの使用を無効にできます。これによりすべてのモードラインが、mode-lineフェイスで表示されます。

モードラインに表示される改行フォーマットは、変数 eol-mnemonic-unix、eol-mnemonic-dos、eol-mnemonic-mac、および eol-mnemonic-undecidedをセットすることにより、カスタマイズできます。

11.20 テキストが表示される方法

ほとんどの文字は、印字文字 *(printing characters)* です。これらの文字がバッファーに存在すると、スクリーンにそのまま表示されます。印字文字には ASCIIの数字、文字、区切り文字、同様に多くの非 ASCII文字が含まれます。

ASCII文字セットには、印字されない制御文字 *(control characters)* が含まれます。その中でも特別に表示されるものが 2 つあります。1 つ目は改行文字 (Unicode のコードポイント U+000A) で、新しい行を開始するのに表示されます。2 つ目はタブ文字 (U+0009) で、次のタブストップ (通常は 8 文字ごと) までをスペースで表示します。タブを何文字のスペースで表示するかは、バッファーローカルな変数 tab-widthで制御され、1 から 1000 の整数で指定しなければなりません。バッファーのタブ文字が表示される方法は、コマンドとしての TABの定義には関係ないことに注意してください。

他の ASCII制御文字としては、U+0020(8 進の 40、10 進の 32) より下のコードがあり、それらはカレット (‘^’) と、その後ろに非制御文字バージョンの文字を続けて、escape-glyphフェイスで表示されます。たとえば文字 ‘control-A’(U+0001) は、‘^A’と表示されます。

コード U+0080(8 進の 200) から U+009F(8 進の 237) までの raw バイトは、escape-glyphフェイスにより *8* 進エスケープシーケンス *(octal escape sequences)* で表示されます。たとえば文字コード U+0098(8 進の 230) は ‘\230’と表示されます。バッファーローカルな変数 ctl-arrowを nilに変更すると、ASCII制御文字もカレットエスケープシーケンスではなく 8 進エスケープシーケンスで表示されます (raw バイトを 16 進表示するように要求することも可能。Section 11.24 [Display Custom], page 102 を参照)。

非 ASCII文字の中には、ASCIIのスペースやハイフン (マイナス記号) と同じ外観を持つものがあります。そのような文字は、意識せずにバッファーに入力されたとき (たとえば yank などで)、問題となることがあります。たとえばソースコードコンパイラーは通常、非 ASCIIのスペースを、空白文字として扱いません。この問題に対処するため、Emacs はそのような文字を特別な方法 (U+00A0 NO-BREAK SPACE および他の Unicode horizontal space class 由来の文字は `nobreak-space`フェイス、U+00AD SOFT HYPHEN、U+2010 HYPHEN、および U+2011 NON-BREAKING HYPHEN は `nobreak-hyphen`フェイス) で表示します。これを無効にするには、変数 `nobreak-char-display` を `nil`に変更します。この変数に非 `nil`かつ非 `t`の値を与えると、Emacs はハイライトされたバックスラッシュの後に、スペースまたはハイフンを表示します。

特定の文字コードの表示のカスタマイズは、ディスプレイテーブル (display table) によって行われます。Section “Display Tables” in *The Emacs Lisp Reference Manual* を参照してください。

グラフィカルなディスプレイでは、Emacs が利用可能なフォントにグリフがない文字がいくつかあります。これらのグリフがない文字 *(glyphless characters)* は、通常 16 進文字を含むボックスで表示されます。テキスト端末では、端末エンコーディング (Section 19.13 [Terminal Coding], page 231 を参照してください) で表示できない文字は、通常クエスチョン記号で表示されます。表示方法は、変数 `glyphless-char-display-control`で制御できます。これらの文字の表示がより目立つように、`glyphless-char`フェイスをカスタマイズすることもできます。詳細は、Section “Glyphless Character Display” in *The Emacs Lisp Reference Manual* を参照してください。

マイナーモード `glyphless-display-mode`を使えば、カレントバッファーにおけるグリフなし文字の表示を切り替えることができます。グリフのない文字は、中にその文字の名前の頭文字が入ったボックスとして表示されます。

Emacs はカレントのディスプレイで curved quotes(‘ と ’) が表示可能か判断を試みます。デフォルトでは、表示可能なら Emacs はメッセージやヘルプテキスト内の ASCIIクォート ((‘`’と ‘'’)) を curved quotes に変換します。ユーザーオプション `text-quoting-style`をカスタマイズすることにより、この変換を有効または無効にできます (Section “Keys in Documentation” in *The Emacs Lisp Reference Manual* を参照)。

curved quotes(‘ 、 ’、“ 、 ”) を ASCII文字と同様な外観で見ることができる場合、それらは `homoglyph`フェイスで表示されます。表示できないことが既知の curved quotes は、それらの ASCII による代替である ‘`’、‘'’、‘"’が `homoglyph`で表示されます。

11.21 カーソルの表示

テキスト端末では、カーソルの外見は端末により制御され、大部分は Emacs の制御が及びません。いくつかの端末は、普通の固定的なカーソルと、目立つ点滅カーソルの 2 種類を提供します。デフォルトでは Emacs は目立つカーソルを使い、Emacs を開始または再開したときは、そのカーソルに切り替えます。変数 `visible-cursor`が `nil`の場合、Emacs を開始または再開したとき、普通のカーソルを使います。

グラフィカルなディスプレイでは、より多くのテキストカーソルのプロパティを変更できます。カラーを変えるには、フェイス `cursor`の、属性`:background`を変更します (このフェイスの他の属性には、何を指定しても効果はありません。カーソルの下にあるテキストはフレームのバックグラウンドカラーを使って描画されます)。外見を変更するには、バッファーローカルな変数 `cursor-type`をカスタマイズします。有効な値は `box` (デフォルト)、`(box . size)` (いずれかの次元において *size* ピクセルより大なマスクされたイメージ下で中抜きのボックスになるボックスカーソル)、`hollow` (中抜きのボックス)、`bar` (垂直のバー)、`(bar . n)` (幅が *n*ピクセルの垂直バー), `hbar` (水平バー)、`(hbar . n)` (高さが *n*ピクセルの水平バー)、または `nil` (カーソルなし) です。

デフォルトでは、カーソルは 10 回点滅する間に Emacs に何も入力がないと、点滅をストップします。そして何らかの入力イベントがあると、また 0 からカウントを再開します。変数 `blink-cursor-blinks`をカスタマイズして、これを制御できます。変数の値には、何の入力もないとき点滅をストップする点滅回数を指定します。変数に 0 または負の値をセットすると、カーソルはずっと点滅したままになります。カーソルの点滅を無効にするには、変数 `blink-cursor-mode`を `nil`に変更するか (Section 33.1 [Easy Customization], page 499 を参照してください)、init ファイルに以下の行を追加します:

```
(blink-cursor-mode 0)
```

リスト変数 `blink-cursor-alist`をカスタマイズして、カーソルが点滅をストップしたとき、どのように見えるかを変更できます。リストの各要素は、`(`*`on-type`* `.` *`off-type`*`)`という形式を指定します。*on-type*には、点滅しているときのカーソルを指定します (*on-type*には、上で説明したカーソルタイプを指定します)。そして *off-type*には、点滅していないときのカーソルを指定します。

タブ文字のように、特別に幅が広い文字もあります。そのような文字上にカーソルがあるとき、通常はデフォルトの文字幅で描画されます。カーソルを文字幅に伸ばすには、変数 `x-stretch-cursor`を非 `nil`値に変更してください。

選択されていないウィンドウのカーソルは、通常点滅していない中抜きのボックスで表示されます (カーソルにバーを使っている場合、より細いバーで表示されます)。選択されていないウィンドウでカーソルを非表示にするには、変数 `cursor-in-non-selected-windows`を `nil`に変更してください。

カーソルをよりはっきりと表示させるために、HL Line モードを使用できます。このモードでは、ポイントを含む行がハイライトされます。現在のバッファーで有効または無効にするには、`M-x hl-line-mode`を使います。このモードをグローバルに有効または無効にするには、`M-x global-hl-line-mode`を使用してください。

11.22 行の切り詰め

Emacs は行を継続 (Section 4.8 [Continuation Lines], page 23 を参照してください) するかわりに、長い行を切り詰めて表示できます。これは、スクリーンやウィンドウの幅より長い行は、全体が表示されないことを意味します。グラフィカルなディスプレイでは、行が切り詰められている場合、フリンジに小さな直矢印が表示されます。テキスト端末では、右端および/または左端の列に '`$`'が表示されます。

水平スクロールは、自動的に行の切り詰めを引き起こします (Section 11.4 [Horizontal Scrolling], page 80 を参照してください)。特定のバッファーにたいして行の切り詰めを明示的に有効にするには、コマンド `C-x x t` (`toggle-truncate-lines`) を使います。これは変数 `truncate-lines`をローカルに変更することで機能します。値が非 `nil`のときは、長い行は切り詰められ、`nil`のときは複数のスクリーン行に分けられます。変数 `truncate-lines`をセットすると、現在のバッファーにローカルに適用されます。値を変更するまでは、デフォルト値 (`nil`) が使われます。

行切り詰めと単語折り返し (次セクションで説明) は排他なので、`toggle-truncate-lines`で行切り詰めをオンにすると単語折り返しが無効になります。

ウィンドウを分割して狭くなりすぎたとき、Emacs は自動的に行の切り詰めを有効にします。これを制御する変数 `truncate-partial-width-windows`については、Section 17.2 [Split Window], page 186 を参照してください。

11.23 Visual Line モード

このモードでは、通常の行継続の代わりに、単語での折り返しが使われます。通常の行継続のように、長い論理行は 2 行以上のスクリーン行に分割されます。しかし Emacs はウィンドウの右端 (RTL 言語:Right-To-Leftlanguage では左端) の近くの、単語の境界で折り返すよう試みます。これは単語の途中で折り返さないことにより、可読性を高めるためです。

単語での折り返しは、オプションのマイナーモードである、Visual Line モードで有効になります。現在のバッファーで Visual Line モードの有効と無効を切り替えるには、`M-x visual-line-mode`とタイプします。メニューバーから Visual Line モードを有効にすることもできます (Options メニューから、サブメニュー '`Line Wrapping in this Buffer`'の、メニューアイテム '`Word Wrap (Visual Line Mode)`'を選択します)。Visual Line モードが有効なときは、モードラインのモード表示に '`wrap`' という文字が表示されます。コマンド `M-x global-visual-line-mode`は、全バッファーの Visual Line モードを切り替えます。

単語折り返し (前セクションで説明) と行切り詰めは排他なので、`visual-line-mode`をオンにすると行切り詰めが無効になります。

Visual Line モードでは、いくつかのコマンドは論理行ではなくスクリーン行に作用します。`C-a` (`beginning-of-visual-line`) はスクリーン行の先頭に移動し、`C-e` (`end-of-visual-line`) はスクリーン行の最後に移動、`C-k` (`kill-visual-line`) はテキストをスクリーン行の最後まで kill します。

論理行単位で移動するには、コマンド `M-x next-logical-line`または `M-x previous-logical-line`を使います。これらのコマンドは Visual Line モードが有効であるかにかかわらず、次または前の論理行に移動します。これらのコマンドを頻繁に使う場合は、キーを割り当てると便利でしょう。Section 33.3.6 [Init Rebinding], page 522 を参照してください。

デフォルトでは、単語の折り返し表示はフリンジに表示されません。Visual Line モードは、長い論理行を含むファイルを編集するときに使われる場合があり、折り返し行すべてにフリンジの表示をすると見にくくなるためです。これを変更するには、変数 `visual-line-fringe-indicators`をカスタマイズしてください。

デフォルトでは Emacs は `SPC`や `TAB`のような空白文字の後でのみ行ブレークを行い、`EN QUAD`のような空白文字の後での行ブレークは行いません。Emacs はカレントモードでの単語折り返しを切り替える `word-wrap-whitespace-mode`と呼ばれるマイナーモードを提供しています。このモードはどの文字で行の折り返しを行うかをユーザーオプション `word-wrap-whitespace-characters`にもとづいてセットアップします。

CJK と Latin が混ざったテキストでは、空白文字の後でのみブレークすると正しくない結果を生成するかもしれません (CJK は単語区切りに空白文字を使用しないため)。CJK 文字にたいしてより良いサポートを提供するために任意の '`|`'カテゴリー (Section "Categories" in *the Emacs Lisp Reference Manual* を参照) の文字の後で Emacs が行ブレークできるように、オプション `word-wrap-by-category`をカスタマイズできます。この変数が Customize を使用してセットされていると、Emacs は自動的に `kinsoku.el`をロードします。`kinsoku.el`のロード時には、Emacs は行ブレークする際に禁足規則にしたがいます。これは '`>`'カテゴリーの文字 (たとえば U+FF0C FULLWIDTH COMMA) が行頭、'`<`'カテゴリーの文字 (たとえば U+300A LEFT DOUBLE ANGLE BRACKET) が行末に出現しないことを意味します。コマンド `char-category-set`か `category-set-mnemonics`を使用してカテゴリーを閲覧したり、あるいはポイント上で `C-u C-x =`とタイプして結果の "category" セクションを調べることができます。コマンド `modify-category-entry`を使用して、文字にカテゴリーを追加できます。

11.24 ディスプレイのカスタマイズ

このセクションでは、Emacs スクリーンの外観を制御するさまざまな変数を説明します。初心者はスキップして構いません。

Emacs にバッファー内の各行にたいして行番号を表示させたい場合は、バッファーローカルな変数 `display-line-numbers`(デフォルトは `nil`) をカスタマイズします。この変数は行番号表示のさまざまなモードをサポートするために、いくつかの異なる値をもつことができます。

`t` バッファーテキストを表示する継続行ではないスクリーン各行の前に、、(絶対) 行番号を表示します。その行が継続行の場合、またはスクリーン行全体がディスプレイ文字列 (display string)、またはオーバーレイ文字列 (overlay string) の場合、その行に番号は振られません。

`relative` バッファーテキストを表示する非継続行の前に、相対行番号 (relative line number) を表示します。行番号はポイントを表示する行にたいして相対的なので、カレント行から遠ざかるにしたがって、行番号は増加または減少します。

`visual` この値により、Emacs にビジュアル的に行をカウントさせます。実際にディスプレイに表示されている行だけがカウントされ、ラップして複数行を占めるスクリーン行は、複数回番号付けされます。表示される番号は、上述の値 `relative`のように相対的です。これは Outline モード (Section 22.9 [Outline Mode], page 261 を参照) のような、テキストをフォールド (fold: 折り畳む) するモードで、正確なスクリーン行の番号により移動するときに役立つでしょう。

その他 その他の非 `nil`値は、`t`として扱われます。

コマンド `M-x display-line-numbers-mode`は、行番号表示を切り替える便利な方法を提供します。このモードのグローバル版は、`global-display-line-numbers-mode`です。ユーザーオプション `display-line-numbers-type`は、上述した行番号表示のサブモードのどのモードをアクティブにするかを制御します。

たとえグローバルに `display-line-numbers-mode`をオンにしていても、ミニバッファーやツールチップ内に行番号は表示されないことに注意してください。

Emacs が相対行番号を表示しているとき、カレント行 (ポイントを表示している行) の前に表示される番号を制御できます。デフォルトでは、Emacs はカレント行にたいしては、他のすべての行が相対行番号であっても、絶対行番号を表示します。変数 `display-line-numbers-current-absolute` を `nil`値にカスタマイズした場合、カレント行に表示される番号は 0 になります。これはカレント行の番号が重要ではなく、大きなバッファー内のテキストにたいして、より多くの水平方向の空きを残したいとき便利でしょう。

ナロー (Section 11.5 [Narrowing], page 81 を参照) されたバッファーでは通常、ナローイングの先頭から番号が開始されます。しかし、変数 `display-line-numbers-widen`を非 `nil`値にカスタマイズした場合、ナローイングは無視されて、そのバッファーの最初の文字から行番号が開始されます。

`display-line-numbers-offset`の値が非 0 なら、各絶対行番号に追加されて、行はあたかも `display-line-numbers-widen`が非 `nil`であるかのようにバッファー先頭から計数されます。0 にセットされた場合、あるいは行番号が絶対でない場合には影響はありません。

Selective Display(選択的表示) モード (Section 11.18 [Selective Display], page 96 を参照)、およびその他の (Outline モードや Org モードのような) ディスプレイから多くの行を隠すモードでは、行番号のために予約済みのスペースにたいする法則性のない計算ミスを避けるために、変数

`display-line-numbers-width-start`および`display-line-numbers-grow-only`のカスタマイズ、または`display-line-numbers-width`に十分大きな値をセットしたいと思うかもしれません。

行番号は特別なフェイス `line-number`で表示されます。カレント行番号は異なるフェイス`line-number-current-line`で表示されるので、ポイントを表示する行を見つける助けとなるようにカレント行番号に異なる外観を与えることができます。特定の数値の倍数であるような行番号をハイライトするために、追加フェイスの `line-number-major-tick`と `line-number-minor-tick`を使用できます。それらの数値に `display-line-numbers-major-tick`と`display-line-numbers-minor-tick`をカスタマイズしてください。

変数`visible-bell`が非`nil`の場合、Emacs は通常ベルサウンドを鳴らす場面で、スクリーン全体を点滅するよう試みます。端末がスクリーンを点滅させる方法を持たないとき、この変数は効果がありません。

変数`echo-keystrokes`は、複数文字キーのエコー表示を制御します。値にはエコーが開始されるまでの秒数を指定します。0 の場合、エコーされません。何かエコーされるべきものがあるときは、この変数の値が効果をもちます。Section 1.2 [Echo Area], page 8 を参照してください。

グラフィカルなディスプレイでは、Emacs はビジーのときにマウスポインターを砂時計で表示します。この機能を無効にするには、変数 `display-hourglass`に `nil`をセットします。変数`hourglass-delay`は、砂時計が表示されるまでのビジーな時間を、秒数で指定します。デフォルトは 1 です。

マウスポインターが Emacs のフレーム内にある場合、文字をタイプしてテキストを挿入するまでの間、テキストを隠さないよう、Emacs はマウスポインターを非表示にします (正確に言うとマウスポインターの非表示は、自己挿入的 (self-inserting) な文字をタイプしたときです。Section 4.1 [Inserting Text], page 17 を参照してください)。マウスポインターを動かすと、再び表示されます。この機能を無効にするには、変数 `make-pointer-invisible`に `nil`をセットしてください。

グラフィカルなディスプレイでは、変数`underline-minimum-offset`は、アンダーラインされたテキストの、アンダーラインから文字の基底線までの最小距離を、ピクセルで決定します。デフォルトでは値は 1 です。この変数を増加させると、特定のフォントにおいて、アンダーラインされたテキストの可読性が向上します (しかし Emacs は、カレント行にはアンダーラインを描画しません)。変数`x-underline-at-descent-line`は、テキストにアンダーラインを引く方法を決定します。デフォルトは `nil`で、これはフォントの基底線と同じレベルに描画されることを意味します。これを `t`に変更すると、Emacs はフォントが同じ大きさになるよう、アンダーラインを少し下に描画します。(アンダーラインされるテキストにたいして非デフォルトの行間が指定された場合、Emacs は追加される行間の下にアンダーラインを描画する。Section “Line Height” in *The Emacs Lisp Reference Manual* を参照されたい。)

変数`overline-margin`は、テキストの上のオーバーラインの垂直位置を、オーバーライン自身の高さも含めて、ピクセルで指定します。デフォルトは 2 です。

テキスト端末には、bold(太字) かつ反転されたテキストが読みにくいものがあります。関数`tty-suppress-bold-inverse-default-colors`に、引数非 `nil`を与えて呼び出すと、このような場合の bold-face の効果を抑制します。

デフォルトでは raw バイトは、たとえば 10 進で 128 という値をもつバイトは`\200`といったように 8 進形式で表示されます。これを`\x80`のように 16 進形式で表示するよう変更するには、変数`display-raw-bytes-as-hex`に `t`をセットしてください。Emacs セッションを含む端末からテキストをコピーしたり、端末の `escape-glyph`Care フェイスの外観がデフォルトフェイスに似ている際には、raw バイトの解釈に注意が必要かもしれません。たとえば Emacs が ‘\’、‘2’、‘0’、‘0’という 4 つの文字を表示するのは、値が 10 進で 128 であるようなバイトを表示する際と同じ文字です。更に悪いことに 16 進の表示では raw バイトの 128 の後に文字 ‘7’がある場合には、Emacs Lisp はこれを

U+0807、すなわち SAMARITAN LETTER IT という単一の文字として読み取ってしまいます。8 進エスケープでは最大で 3 桁なので、これに相当する 8 進の\2007という表示でも混乱は生じません。

12 検索と置換

他のエディターと同様、Emacs には文字列を検索 (search) するコマンドがあります。Emacs には、文字列を違う文字列で置き換える (replace) コマンドもあります。また、同じことを行いますが、固定文字列ではなくパターンを検索するコマンドもあります。

`xref`の制御下にある複数ファイルにたいして検索したり ((Section 25.4.1.3 [Identifier Search], page 360 を参照してください)、Dired の `A`コマンドを通じて検索したり (Section 27.7 [Operating on Files], page 387 を参照してください)、`grep`コマンドを使った検索 (Section 24.4 [Grep Searching], page 313 を参照してください) も可能です。

12.1 インクリメンタル検索

Emacs における重要な検索コマンドは、インクリメンタル検索 *(isearch: incremental search)* です。これは検索する文字列の最初の文字をタイプすると、すぐに検索が開始されます。検索文字列をタイプしていくにつれて、Emacs はその文字列 (それまでに入力した文字列) がどこにあるかを表示します。望む場所を特定するのに充分な文字列をタイプしたところで、検索をストップできます。次に何をするかによって、明示的な `RET`により検索を終えたり、続けることができます。

`C-s`　　前方にインクリメンタル検索します (`isearch-forward`)。

`C-r`　　後方にインクリメンタル検索します (`isearch-backward`)。

メニューバーの '`Edit->Search`'メニューからインクリメンタル検索を呼び出すこともできます。

12.1.1 インクリメンタル検索の基本

`C-s`　　インクリメンタル検索を開始します (`isearch-forward`)。

`C-r`　　逆向きのインクリメンタル検索を開始します (`isearch-backward`)。

`C-s` (`isearch-forward`) は、前方へのインクリメンタル検索を開始します。これはキーボードから文字を読み取り、タイプした文字が最初に出現するバッファーの位置に、ポイントを移動します。

たとえば `C-s`とタイプした後に `F`をタイプすると、検索を開始したバッファーのポイント位置より前方にある、最初の `F`にカーソルを移動します。つぎに `O`をタイプすると、前方にある最初の '`FO`'にカーソルが移動します。この場合、'`FO`'の '`F`'は、前に見つかった '`F`'と同じ位置である必要はありません。もう一度 `O`をタイプすると、カーソルは最初の '`FOO`'に移動します。

各ステップで Emacs はカレントマッチ *(current match:* 現在のマッチ*)*(検索文字列にマッチしたバッファーのテキスト) を、`isearch`フェイス (Section 11.8 [Faces], page 82 を参照してください) でハイライトします。このハイライトをカスタマイズするさまざまなオプションについては、Section 12.12 [Search Customizations], page 129 を参照してください。その時点での検索文字列はエコーエリアにも表示されます。

検索文字列のタイプ中に間違ったら、`DEL`(`isearch-delete-char`) とタイプしてください。`DEL` をタイプするたびに、検索の間にエンターした最後の入力がキャンセルされます。検索文字列、ポイント位置、検索の成否、検索の方向、カレント検索結果の他端位置、検索の "ラッピング" を変更するコマンドのタイプ時、Emacs は常に新たな入力アイテム *(input item)* を記録します。失敗した検索を処理するための詳細は、Section 12.1.4 [Error in Isearch], page 108 を参照してください。

検索により到達した位置に満足したら、`RET` (`isearch-exit`) をタイプします。これは検索をストップして、検索により移動した位置にカーソルを残します。検索とは関係ないコマンドも、検索をストップして、そのコマンドが実行されます。つまり `C-a`は検索を exit して行の先頭に移動し、矢

印キーをタイプすると検索を exit して対応する移動コマンドが処理される、などとなります。RETで検索を終える必要があるのは、次のコマンドが印字文字をタイプするコマンドのときだけです。DEL、RETおよび他のいくつかの文字 (C-q、C-w、C-r、C-s、C-y、M-y、M-r、M-c、M-e、および以下で説明する文字) は、検索で特別な意味を持っています。検索を exit するコマンドを微調整できます。Section 12.1.6 [Not Exiting Isearch], page 110 を参照してください。

特別な例外として、検索文字列が空のときに RETを入力すると、非インクリメンタル検索 (Section 12.2 [Nonincremental Search], page 112 を開始します (これは"カスタマイズ可能"です。Section 12.12 [Search Customizations], page 129 を参照してください)。

検索を中止して検索を開始した位置に戻るには、ESC ESC ESC (isearch-cancel)、または C-g C-g (isearch-abort) とタイプしてください。

インクリメンタル検索を終了するとき、ポイントの元の位置をマークをアクティブにせず (ただしマークがすでにアクティブではなかったときだけ) に、マークリングに追加します。これにより、C-u C-SPC、または C-x C-xを使って、検索を開始する前の位置に戻ることができます。Section 8.4 [Mark Ring], page 56 を参照してください (Emacs はマークがすでにアクティブでないときだけ、これを行ないます。検索を開始したときにマークがアクティブな場合は、C-u C-SPCと C-x C-xの両方とも、マークに戻ります)。

後方に検索するには、C-sで検索を開始するかわりに、C-r (isearch-backward) を使います。前方検索が検索を開始した位置より前方にある最初のマッチを探すように、後方検索は検索を開始した位置より後方にある最後のマッチを探します。

12.1.2 インクリメンタル検索の繰り返し

前方に 'FOO'を検索してマッチしたが探しているマッチではなく、探しているのはバッファーのもっと前方に出現する 'FOO'だとしましょう。ここでもう 1 度 C-s (isearch-repeat-forward) をタイプすることにより、検索文字列が次に出現する場所、C-r (isearch-repeat-backward) とタイプすれば検索文字列が前に出現した場所に移動します。これは何回でも繰り返すことができます。次、または前へ *n*個目の出現場所を見つけるために C-sと C-rにプレフィクス数引数 *n*をかわりに供給することもできます。もしタイプしすぎたときは、C-sコマンドを DELで取り消すことができます。同様に後方へのインクリメンタル検索では、C-r (isearch-repeat-backward) により後方検索が繰り返されます。

インクリメンタル検索中に手を止めて眺めてみると、検索文字列にたいするカレントマッチ以外に、スクリーンに表示されている他のマッチもハイライトされていることがわかります。これは検索するために、何回 C-sまたは C-rを繰り返せばよいのか予測しやすくするためです。その他のマッチは、カスタマイズ可能なフェイス lazy-highlight (Section 11.8 [Faces], page 82 を参照してください) を使って、現在のマッチとは異なってハイライトされます。この機能を無効にするには、isearch-lazy-highlightに nilをセットしてください。マッチのハイライトに関連するその他のカスタマイズについては、Section 12.12 [Search Customizations], page 129 を参照してください。

検索を終了した後、同じ文字列を再度検索するには、C-s C-sとタイプします。最初の C-sはインクリメンタル検索を呼び出し、2 回目の C-sは最後に検索した文字列の再検索を意味します。同様に、C-r C-rは最後に検索した文字列を後方に検索します。最後に検索された文字列の決定では、その文字列が C-sで検索されたのか、C-rで検索されたのかは問題ではありません。

前方に検索していて、検索している対象が検索開始点より後方にあるのに気付いたときは、検索文字列を変更すること無く、C-rで後方検索に切り替えることができます。同様に後方検索で C-sをタイプすると、前方検索に切り替わります。

検索方向を変更すると、最初にコマンドをタイプした時点ではデフォルトでは同じマッチ上に留まって、カーソルはそのマッチの他端へと移動するでしょう。別のマッチへ即座に移動するには、変数 isearch-repeat-on-direction-changeを tにカスタマイズしてください。

検索が失敗したとき、バッファーの先頭から検索を再開するには、もう一度 C-sをタイプします。逆向きの繰り返し検索が失敗したときは、C-rでバッファーの最後から検索を再開します。これは *wrapping around*(巻き直し) と呼ばれ、これが発生すると、'Wrapped'という単語が検索プロンプトに表示されます。検索を続けて元の検索開始ポイントを通過すると、'Overwrapped'に変化します。これはすでに見たマッチを再訪していることを意味します。

それ以上マッチが存在しない際に何が起こるかは、ユーザーオプション isearch-wrap-pauseのカスタマイズによって制御できます。t (デフォルト) ならエラーをシグナル (繰り返し検索すると最初に戻り検索)、noなら dingを発して最後のマッチ到達後即座に最初に戻り検索、no-dingなら dingなしで即座に最初に戻り検索を行います。値が noか no-dingなら、文字のタイプ中にも最初に戻って検索を試みます。最後に nilなら最初へは戻らず最後のマッチで単に停止します。

以前に検索した文字列を再利用するには、サーチリング *(search ring)* を使います。コマンド M-p (isearch-ring-retreat) または M-n (isearch-ring-advance) で、リングを移動して再使用したい文字列を取り出します。これらのコマンドは、選択されたリング要素の文字列をミニバッファーに残すので、それを編集することができます。C-sと C-r、または RETとタイプすると、その文字列を受け入れて、その文字列にたいする検索を開始します。サーチリングの中に保存されている、最近使用された検索文字列の数は、変数 search-ring-maxで指定され、デフォルトは 16 です。

ミニバッファーの現在の検索文字列を、サーチリングのアイテムで置き換えずに編集するときは、M-e (isearch-edit-string) とタイプするか、ミニバッファーを mouse-1でクリックします。RET、C-s、C-rとタイプすれば、編集を終了してそれを検索できます。バッファーの検索を開始したポイントの後に続く文字を検索文字列に追加するには、C-fまたは RIGHTとタイプしてください。

12.1.3 インクリメンタル検索での yank

多くのケースで、ポイントの近くにあるテキストを検索文字列として使いたいことがあるでしょう。このサブセクションで説明されているコマンドにより、これを便利に行なえるようになります。

C-w (isearch-yank-word-or-char) は検索された文字列のポイントの次の文字または単語を検索文字列に追加します。これはポイント位置にあるテキストを検索する簡単な方法です (コピーする対象を文字または単語のどちらにするかの決定は発見的に行われます)。プレフィクス数引数 *n*を与えると次の *n*個の文字または単語を追加します。

C-M-w (isearch-yank-symbol-or-char) は、検索された文字列のポイントの次の文字またはシンボルを検索文字列に追加します。これはポイント位置にあるシンボルを検索する簡単な方法です (コピーする対象を文字またはシンボルのどちらにするかの決定は発見的に行われます)。プレフィクス数引数 *n*を与えると次の *n*個の文字またはシンボルを追加します。

M-s C-e (isearch-yank-line) は、検索文字列にカレント行の残りの部分を追加します。ポイントが既に行末にある場合には次の行が追加されます。プレフィクス引数 *n*を指定すると次の *n*行を追加します。

同様に C-M-z (isearch-yank-until-char) はポイントから指定した文字の次の出現場所 (その文字を含まず) までのすべてを検索文字列に追加します。これはたとえばプログラミング言語やマークアップ言語でその文字が字句境界をマークするようなときのキーボードマクロにとって有用です。プレフィクス数引数 *n*を与えると、このコマンドはポイントから *n*個目の指定した文字までのすべてを追加します。

インクリメンタル検索での C-y (isearch-yank-kill) は、カレント kill を検索文字列に追加します。C-yの後のインクリメンタル検索中に M-y (isearch-yank-pop) が呼び出されると、追加す

るテキストをもっと前に kill されたものに置き換えます。これは通常の M-y (yank-pop) コマンドと似ています ((Section 9.2 [Yanking], page 62) を参照)。エコーエリアで mouse-2をクリックすることにより、はカレントの X 選択 (Section 9.3.2 [Primary Selection], page 66 を参照) を検索文字列に追加します (isearch-yank-x-selection)。

C-M-d (isearch-del-char) は検索文字列の最後の文字を削除し、C-M-y (isearch-yank-char) は検索されたポイントの後の文字を追加します。ポイントの後ろの文字を追加する他の方法は、M-e(Section 12.1.2 [Repeat Isearch], page 106 を参照してください) でミニバッファーに移動してから、検索文字列の最後で C-fか RIGHTをタイプします。C-fか RIGHTをタイプするたびに、ポイントの後の文字が検索文字列に順次追加されます。

検索が大文字小文字を区別しない場合は通常、検索文字列に yank されるテキストは小文字に変換されるので、検索は大文字小文字を区別しないままです (Section 12.9 [Lax Search], page 120 を参照してください)。しかし、変数 search-upper-case (Section 12.9 [Lax Search], page 120 を参照してください) の値が not-yanks以外の場合には、この小文字への変換は無効になります。

ポイント近傍のテキストを最初の検索文字列に yank してインクリメンタル検索を新たに開始するには、コマンド isearch-forward-thing-at-pointを実行するために M-s M-.をタイプしてください。これはリージョンがアクティブなら、リージョンからテキストを検索文字列に yank、それ以外の場合にはポイントの近くで見つかった URL、シンボル、または式の yank を試みます。何を yank するかはユーザーオプション isearch-forward-thing-at-pointで定義されます。

12.1.4 インクリメンタル検索でのエラー

文字列が見つからなかった場合、エコーエリアに‘Failing I-Search’と表示されて、文字列とできるかぎりマッチした位置に、カーソルが移動します。つまり‘FOOT’を検索して‘FOOT’がない場合、カーソルは‘FOOL’という文字列の‘FOO’の後ろに移動します。エコーエリアではマッチに失敗した検索文字列の一部が、フェイス isearch-failを使ってハイライトされます。

その時に行うことができる選択肢がいくつかあります。もし文字列が間違っている場合には、文字列を訂正するために、DELで前の入力アイテム (Section 12.1.1 [Basic Isearch], page 105 を参照) を削除、一度に一文字削除するなら C-M-d、編集する場合は M-eとタイプします。もし見つかった位置が望む位置なら、RETをタイプしてその位置に留まることができます。または C-gをタイプして、検索文字列から検索できなかった文字 (‘FOOT’の中の‘T’) を取り除き、検索された部分の文字列 (‘FOOT’の中の‘FOO’) を残します。その位置でもう 1 回 C-gをタイプすると、検索全体を取り消し、ポイントは検索を開始した位置に戻ります。

終了コマンドの C-gは、検索において特別な処理を行います。このコマンドの動作は、検索の状況に依存します。もし指定した文字列の検索が成功して、さらに検索文字の入力を待っているとき、C-gは検索全体を取り消して、カーソルを検索を開始したときの位置に移動します。検索文字列に検索に失敗した文字が含まれているときに、C-gがタイプされたときは、検索文字列から検索に失敗した文字が取り除かれます。後に残るのは検索に成功した文字列で、さらに検索文字の入力を待っているので、先のケースと同様、2 回目の C-gで検索全体が取り消されます。

12.1.5 インクリメンタル検索の特別な入力

前のサブセクションで説明した文字に加えて、インクリメンタル検索のときにタイプする文字列の中には、特別な効果をもつものがあります。ここではそれらについて説明します。

lax space matching(「だらしない、ゆるんだ、緩慢な、締まりのない」スペースのマッチング。Section 12.9 [Lax Search], page 120 を参照してください) を切り替えるには、M-s SPCとタイプします。

検索で case sensitivity(大文字小文字を区別するか) を切り替えるには、M-cまたはM-s cとタイプします。Section 12.9 [Lax Search], page 120 を参照してください。検索文字列が大文字を含む場合、デフォルトではその検索は case-sensitive(大文字小文字を区別) します。

検索が似ている文字、または等価な文字を考慮するかどうかを切り替えるには、M-s 'とタイプします。Section 12.9 [Lax Search], page 120 を参照してください。検索文字列にアクセント付きの文字が含まれる場合、その検索の間、character folding は無効になります。

非表示のテキストを検索するかしないかは、M-s i (isearch-toggle-invisible) とタイプして切り替えることができます。[Outline Search], page 265 を参照してください。

インクリメンタル検索で、非正規表現による検索と、正規表現による検索を切り替えるには、M-r、またはM-s r (isearch-toggle-regexp) とタイプします。Section 12.5 [Regexp Search], page 114 を参照してください。

シンボルモードを切り替えるには、M-s _とタイプします。Section 12.4 [Symbol Search], page 113 を参照してください。

改行文字を検索するには、検索文字列の途中で C-jとタイプします。

インクリメンタル検索の間に非 ASCII文字を検索するには、以下の方法のいずれかを使用してください:

- C-q (isearch-quote-char) に続けて、非グラフィック文字か 8 進数字をタイプします。これはC-qを使ってバッファーに文字を挿入するのと同様に、検索文字列にタイプする文字を追加します (Section 4.1 [Inserting Text], page 17 を参照してください)。たとえばインクリメンタル検索で C-q C-sをタイプすると、検索文字列に文字 'control-S'が追加されます。
- 入力メソッド (IM: input method) を使います (Section 19.3 [Input Methods], page 220 を参照してください)。検索を開始したとき、カレントバッファーで入力メソッドが有効の場合、ミニバッファーで検索も L 字列をタイプするときにも、同じメソッドがアクティブになるでしょう。検索文字列をタイプするとき、C-\\ (isearch-toggle-input-method) で、入力メソッドを切り替えることができます。非デフォルトの入力メソッドに切り替えるには、C-^ (isearch-toggle-specified-input-method) を使います。これは入力メソッドの名前を尋ねます。インクリメンタル検索で入力メソッドがアクティブのとき、検索プロンプトには以下のようなニーモニックが含まれます。

  I-search [*im*]:

  ここで *im*はアクティブな入力メソッドのニーモニックです。インクリメンタル検索の間に有効にした入力メソッドは、その後もカレントバッファーで有効のままです。最後に検索文字列に入力メソッドを使用して 1 文字を挿入してから、その後自動的に入力メソッドを無効にするために、C-x \ (isearch-transient-input-method) で一時的に一時入力メソッドを有効にできます ([transient input method], page 223 を参照)。
- C-x 8 RET (isearch-char-by-name) に続けて、Unicode 名か 16 進のコードポイントをタイプします。これは通常の insert-charコマンドと同様に、検索文字列に指定した文字を追加します (Section 4.1 [Inserting Text], page 17 を参照してください)。

サーチリングには Emoji シーケンス (絵文字シーケンス) を含めることもできます。C-x 8 e RET の後に Emoji の Unicode 名 (たとえば smiling faceや heart with arrow) をタイプしてください。これによって指定された Emoji がサーチリングに追加されます。検索したい emoji の名前が判らなければ C-x 8 e l (emoji-list) と C-x 8 e d (emoji-describe) を使うことができます (Section 19.3 [Input Methods], page 220 を参照)。

インクリメンタル検索の中で M-s oとタイプすることにより、カレントの検索文字列で occurを実行する、isearch-occurが呼び出されます。Section 12.11 [Other Repeating Search], page 126 を参照してください。

インクリメンタル検索で M-% (isearch-query-replace) をタイプすると、query-replace または query-replace-regexpが呼び出され (検索モードに依存します)、現在の検索文字が置換対象になります。負のプレフィクス引数は、後方への置換を意味します。Section 12.10.4 [Query Replace], page 124 を参照してください。C-M-% (isearch-query-replace-regexp) とタイプすることにより、カレント検索文字列を置換すべき正規表現として query-replace-regexpが呼び出されます。

インクリメンタル検索で M-TABをタイプすると、isearch-completeが呼び出され、サーチリング (以前に使用された検索文字列) を補完リストとして使って、検索文字列の補完を試みます。Section 5.4 [Completion], page 31 を参照してください。多くのオペレーティングシステムでは、キーシーケンス M-TABはウィンドウマネージャーに捕えられます。その場合、これを使うには isearch-complete を、他のキーシーケンスに再バインドする必要があります (Section 33.3.5 [Rebinding], page 521 を参照してください)。

M-s h r (isearch-highlight-regexp) とタイプすればマッチをハイライトしたまま検索を exit できます。これはハイライトに使用するフェイスの入力を求める highlight-regexp (Section 11.14 [Highlight Interactively], page 91 を参照) に、検索文字列から継承した regexp をに渡して実行します。(マッチだけではなく) マッチを含む行全体をハイライトするには、M-s h l (isearch-highlight-lines-matching-regexp) とタイプします。いずれの場合でも、ハイライトを除去するには M-s h u (unhighlight-regexp) とタイプしてください。

インクリメンタル検索がアクティブのとき、C-h C-h (isearch-help-map) とタイプすると、特別なキーバインドのリストを含む、対話的なヘルプにアクセスできます。これらのキーバインドは、キーマップ isearch-mode-mapの一部です (Section 33.3.1 [Keymaps], page 519 を参照してください)。

インクリメンタル検索がアクティブな際に M-s M->とタイプすると検索文字列の最後の出現位置、M-s M-<とタイプすると最初の出現位置に移動します。プレフィクス数引数 *n*を与えると、これらのコマンドはそれぞれバッファーの先頭または最後から数えて *n*個目の検索文字列の出現位置へ移動します。

12.1.6 インクリメンタル検索を終了させない

このサブセクションでは検索で特別な意味をもたないコマンドがコマンドを実行する前に、検索を exit するかどうかを制御する方法を説明します。また、(たとえそれらがインクリメンタル検索の一部ではなくても) カレントのインクリメンタル検索を exit せずにタイプできる、3 つのカテゴリーに属するコマンドを説明します。

インクリメンタル検索によりバインドされていないコマンドをタイプすると通常、そのコマンドを実行する前に検索を exit します。したがって、そのコマンドは検索を呼び出したときのバッファーにたいして処理を行なうことになります。しかし、変数 search-exit-optionを appendにカスタマイズした場合、(インクリメンタル検索により解釈されないような) タイプした文字は、単に検索文字列に追加されます。これにより、通常は検索を exit して、その文字にバインドされているコマンドをそのバッファーにたいして呼び出す、C-aのような制御文字を検索文字列に含めることができるようになります。

プレフィクス引数

インクリメンタル検索でプレフィクス引数を指定したコマンドをタイプすると、デフォルトではその引数は次の検索アクションに適用されるか、検索を exit するコマンドに渡されます。他の言い方をすると、プレフィクス引数の入力自体は、検索を終了させません。

以前のバージョンの Emacs では、プレフィクス引数の入力は常に検索を終了させていました。この振る舞いに戻すには、変数 `isearch-allow-prefix`に `nil`をセットしてください。

`isearch-allow-scroll`が非 `nil`のとき (以下を参照)、プレフィクス引数は上で説明したようなデフォルト動作をします。つまり、たとえ `isearch-allow-prefix`が `nil`でも、プレフィクス引数は検索を終了させません。

スクロールコマンド

通常スクロールコマンドは、インクリメンタル検索を終了させます。しかし変数 `isearch-allow-scroll`を非 `nil`値に変更すると、非 `nil`の `scroll-command`プロパティをもったスクロールバーや `C-v`、`M-v`、`C-l`のようなスクロールコマンド (Section 11.1 [Scrolling], page 77 を参照) が利用可能になります。これは、これらのコマンドをバウンドされたキーシーケンスで呼び出したときだけ適用されます。つまり `M-x`は依然として検索を終了させます。これらのコマンドには、通常の方法でプレフィクス引数を与えることができます。この機能では通常はカレントマッチが表示されない位置にスクロールすることはできません。しかし `isearch-allow-scroll`を特別な値 `unlimited`にカスタマイズすることにより、この制限は解除されます。

`isearch-allow-scroll`の機能は、正確にはスクロールではないが、テキストが表示されるスクリーン位置に影響する `C-x 2` (`split-window-below`) や `C-x ^` (`enlarge-window`) のようなコマンドにも効果を及ぼします。実際のところこれはコマンドの `isearch-scroll`プロパティが非 `nil`のコマンドに適用されます。そのためどのコマンドが影響を受けるかは、それらのプロパティを変更して制御できます。

たとえば将来の Emacs セッションも含めて、インクリメンタル検索中に `C-h l`を使えれば便利だと思ったら、まず `C-h c`で何のコマンドが実行されるか調べて (Section 7.1 [Key Help], page 45 を参照してください)、それが `view-lossage`だとわかります。その後は init ファイルに以下を追加します (Section 33.4 [Init File], page 528 を参照してください)。

```
(put 'view-lossage 'isearch-scroll t)
```

この機能はポイント、バッファー内容、マッチデータ、カレントバッファーや選択されているウィンドウ・フレームを変更しない任意のコマンドに適用できます。そして、そのコマンド自体がインクリメンタル検索を行ってはなりません。この機能は、`isearch-allow-scroll`が `nil`(デフォルト) の場合は無効です。

同様に変数 `isearch-allow-motion`を非 `nil`値に変更するとキーボード移動コマンド `M-<`、`M->`、`C-v`、`M-v`でカレント検索文字列のバッファー内での最初、最後、カレントウィンドウの後にある最初、カレントウィンドウの前の最後の出現箇所へと移動します。これらの移動コマンド使用時は検索方向は変化しません。変数 `isearch-motion-changes-direction`を非 `nil`値に変更した場合には、検索方向は `M-<`と `C-v`の後は前方、`M->`と `M-v`の後は後方になります。

移動コマンド

`isearch-yank-on-move`を `shift`にカスタマイズしている際には、カーソル移動コマンドをタイプ中にシフトキーを押下することにより検索文字列を拡張できます。これはポイント移動後にカレントバッファーの新たな位置で終了するテキストを yank します。

isearch-yank-on-moveが tの際にはカーソル移動コマンドにシフトキーを使用せずに検索文字列を拡張できますが、それはコマンドのシンボルに isearch-moveプロパティをもつ特定の移動コマンドだけに適用されます。

12.1.7 ミニバッファーの検索

ミニバッファーがアクティブのときインクリメンタル検索を開始すると、Emacs はミニバッファーの内容を検索します。通常のバッファーにたいする検索とは異なり、エコーエリアはミニバッファーの表示に使われているので、検索文字列はエコーエリアには表示されません。

ミニバッファーでのインクリメンタル検索が失敗すると、ミニバッファーヒストリーを検索します。Section 5.5 [Minibuffer History], page 36 を参照してください。ミニバッファーとミニバッファーのヒストリーは、一番古いヒストリー要素が最初で、カレントのミニバッファーが最後にある、一連のページとして視覚化することができます。前方検索の C-sは前方、つまり新しいページを検索し、後方検索の C-rは後方、つまり古いページを検索します。普通のバッファーの検索と同様、検索が失敗すると、最後から最初のページ、またはその逆に巻き直して検索します。

カレントマッチがヒストリーの要素にあった場合、ヒストリーの要素はミニバッファーに取り出されます。インクリメンタル検索を正常に終了 (たとえば RETをタイプ) すれば、それはミニバッファーに残ります。検索を取り消すのは C-gで、これによりミニバッファーの内容は検索を開始したときのものに復元されます。

12.2 非インクリメンタル検索

Emacs には、従来式の非インクリメンタル検索もあります。これは検索を開始する前に、検索文字列全体を入力する必要があります。

C-s RET *string* RET
: *string*を検索します。

C-r RET *string* RET
: 後方に *string*を検索します。

非インクリメンタル検索を開始するには、最初に C-s RETをタイプします。これにより、検索文字列を読みとるために、ミニバッファーに移動します。検索文字列の入力を終了して検索を開始するには、RETをタイプします。文字列が見つからなかったとき、検索コマンドはエラーをシグナルします。

C-s RETとタイプすると、C-sは通常どおりインクリメンタル検索を呼び出します。しかし、このコマンドは指定した文字列が空のとき、非インクリメンタル検索を行うコマンドを呼び出すよう、プログラムされています (そのような用途以外に空の引数は無意味です)。C-r RETも同様に、後方に非インクリメンタル検索を行なうコマンドを呼び出します。

メニューバーの 'Edit->Search'メニューから非インクリメンタル検索を呼び出すこともできます。

よりシンプルな 2 つのコマンド、M-x search-forwardと M-x search-backwardを使うこともできます。これらのコマンドは指定した文字をリテラルとして検索し、case folding(検索で大文字小文字を区別するか) を除く lax-search 機能 (Section 12.9 [Lax Search], page 120 を参照してください) をサポートしません。

12.3 単語検索

単語検索 *(word search)* は、単語の並びを、その間にある区切り文字の種類とは無関係に検索します。たとえば検索文字列に、1 つのスペースで区切られた 2 つの単語を入力すると、2 つの単語を区切るのが、1 つまたはそれ以上のスペース、改行文字、およびそれ以外の区切り文字の場合にもマッチし

ます。これはテキスト文書を検索するとき特に有用です。なぜなら検索する単語が改行で区切られているのか、スペースで区切られているのか考慮しなくてもよいからです。プログラミング言語のためのメジャーモード、およびその他の特別なメジャーモード、そのモードの構文的なニーズに適合するように、単語の定義が変更されているかもしれないことに注意してください。

`M-s w` インクリメンタル検索がアクティブのとき、単語検索モードに切り替えます (`isearch-toggle-word`)。非アクティブのときは、前方へのインクリメンタルな単語検索を開始します (`isearch-forward-word`)。

`M-s w RET` *`words`* `RET`
非インクリメンタルな単語検索を使って、*words*を前方検索します。

`M-s w C-r RET` *`words`* `RET`
非インクリメンタルな単語検索を使って、*words*を後方検索します。

`M-s M-w` リージョン内のテキストにたいして、Web を検索します。

前方へのインクリメンタルな単語検索を開始するには、`M-s w`とタイプします。インクリメンタル検索が非アクティブの場合、これはコマンド `isearch-forward-word`を実行します。インクリメンタル検索がすでにアクティブの場合 (前方か後方かにかかわらず)、`M-s w`は検索の方向と現在の検索文字列は変更せずに、単語検索に切り替えるコマンド `isearch-toggle-word`を実行します。単語検索をオフに切り替えるには、再度 `M-s w`をタイプしてください。

非インクリメンタルな単語検索を開始する場合、前方検索は `M-s w RET`、後方検索は `M-s w C-r RET`をタイプします。これらはコマンド `word-search-forward`、または `word-search-backward`を実行します。

インクリメンタルな単語検索と、非インクリメンタルな単語検索では、マッチを見つける方法に若干の違いがあります。非インクリメンタルな単語検索では、検索文字列の各単語は、単語全体に厳密に一致しなければなりません。インクリメンタルな単語検索では、マッチの規則は緩くなります。検索文字列をタイプするとき、最初と最後の単語は、単語全体にマッチする必要はありません。これはタイプする度にマッチを処理するためです。これは (カーソルがある) カレントマッチ以外のマッチ (lazy matches。Section 12.1 [Incremental Search], page 105 を参照してください) には適用されません。それらは単語全体がマッチしなければハイライトされません。検索文字列をタイプしている間は、`C-s`のような検索繰り返しキーを使用するまで、検索プロンプトに '`Pending`'が表示されます。

単語検索コマンドは、character folding を処理せず、lax whitespace matching(Section 12.9 [Lax Search], page 120 を参照してください) が効果をもたないように切り替えます。

リージョン内のテキストにたいして Web を検索するには、`M-s M-w`とタイプします。このコマンドは、変数 `eww-search-prefix`により指定された URLの検索エンジンを使用して、リージョン内の単語にたいするインターネット検索を行ないます (Section "Basics" in *The Emacs Web Wowser Manual* を参照)。リージョンがアクティブでない場合、このコマンドはユーザーに検索する URL、またはキーワードの入力を求めます。

12.4 シンボル検索

シンボル検索 *(symbol search)* は、通常の検索と似ていますが、検索の境界がシンボルの境界にマッチしていなければなりません。シンボルの意味は、メジャーモードのコンテキストに依存しており、Emacs Lisp モードでの Lisp シンボルのように、通常はソースコードのトークンを参照します。たとえば Lisp シンボル `forward-word`をインクリメンタルなシンボル検索すると、これは `isearch-forward-word`にはマッチしません。そのため、この機能は主にソースコードの検索に有用です。

M-s _ インクリメンタル検索がアクティブのとき、シンボル検索モードに切り替えます (isearch-toggle-symbol)。非アクティブのときは、前方へのインクリメンタルなシンボル検索を開始します (isearch-forward-symbol)。

M-s . ポイントの近くにあるシンボルを、検索文字列の初期値に追加して、前方へのインクリメンタルなシンボル検索を開始します。

M-s _ RET *symbol* RET
*symbol*にたいして、前方への非インクリメンタルな検索をします。

M-s _ C-r RET *symbol* RET
*symbol*にたいして、後方への非インクリメンタルな検索をします。

前方へのインクリメンタルなシンボル検索を開始するには、M-s _ (ポイントの近くにシンボルがあるときは M-s .) をタイプします。インクリメンタル検索がまだアクティブでなければ M-s _はコマンド isearch-forward-symbol、M-s .はコマンド isearch-forward-symbol-at-pointを実行します。プレフィクス数引数 *n*を与えると M-s .はポイント位置のシンボルにたいして次の *n*番目の出現位置を検索します。負の *n*では後方に検索します。すでにインクリメンタル検索がアクティブのとき、M-s _は検索方向と現在の検索文字列を維持した状態で、シンボル検索に切り替えます (もう一度 M-s _をタイプすると、シンボル検索を無効にできます)。インクリメンタルなシンボル検索では、検索み文字列をタイプする間は、検索文字列の先頭がシンボルの先頭にマッチだけが必要であり、C-sのような検索繰り返しキーを使用するまでは検索プロンプトに 'Pending'が表示されます。

非インクリメンタルなシンボル検索は、前方への検索は M-s _ RET、後方への検索は M-s _ C-r RETをタイプします。非インクリメンタル検索では、文字列の先頭と最後が、シンボルの先頭と最後にマッチする必要があります。

シンボル検索コマンドは、character folding を処理せず、lax whitespace matching(Section 12.9 [Lax Search], page 120 を参照してください) が効果をもたないように切り替えます。

12.5 正規表現検索

正規表現 (regular expression: *regexp* と略します) とは、文字列にマッチさせるための文字列候補のクラスを示すパターンです。Emacs は regexp にマッチする検索をインクリメンタル、非インクリメンタルの両方で提供します。正規表現の構文は、次のセクションで説明します。

C-M-s インクリメンタルな regexp 検索を開始します (isearch-forward-regexp)。

C-M-r 逆方向のインクリメンタルな regexp 検索を開始します (isearch-backward-regexp)。

regexp にたいするインクリメンタル検索は、C-M-s (isearch-forward-regexpをタイプするか、プレフィクス引数 (引数の値は何でもよい) を指定して、C-sを呼び出します。前方へのインクリメンタル検索中は、M-rをタイプします。このコマンドは C-sと同様に、検索文字列をインクリメンタルに読みとりますが、検索文字列に正確に一致するバッファーのテキストを検索するのではなく、検索文字列を regexp として扱います。検索文字列にテキストを追加する度に、regexp は長くなり、新しい regexp を検索します。後方への regexp 検索には、C-M-r(isearch-backward-regexp) またはプレフィクス引数を指定した C-rを使います。後方へのインクリメンタル検索中は M-rを使います。

通常のインクリメンタル検索の特別なキーシーケンス (Section 12.1.5 [Special Isearch], page 108 を参照) は、インクリメンタルな regexp 検索でも同じようなことを行います。たとえば検索開始直後に C-sをタイプすると、最後に行ったインクリメンタル検索で使った regexp で、前方検索を行います。インクリメンタルな regexp と非 regexp 検索は、独立したデフォルトを持ちます。これらは別の

サーチリングも持っており、これには M-pと M-nでアクセスできます。サーチリングに保存される検索 regexp の最大数は、regexp-search-ring-maxの値により決定され、デフォルトは 16 です。

通常のインクリメンタル検索とは異なり、インクリメンタル regexp 検索は、デフォルトでは lax space matching を使いません。この機能を切り替えるには、M-s SPC (isearch-toggle-lax-whitespace) を使います。そうするとインクリメンタル regexp 検索での SPCは、1 つ以上の空白文字の並びにマッチするようになります。変数 search-whitespace-regexpは、lax space matching にたいする regexp を指定します。Section 12.1.5 [Special Isearch], page 108 を参照してください。

通常のインクリメンタル検索とは異なり、インクリメンタルな regexp 検索では character folding(Section 12.9 [Lax Search], page 120 を参照してください) を使用できません (インクリメンタル regexp 検索の途中で、M-s 'により character folding を切り替えた場合、検索は非 regexp 検索となり、タイプした検索パターンはリテラル文字列として解釈されます)。

インクリメンタル regexp 検索では検索文字列の追加によりカーソルが前に戻されて、最初から検索しなおされることがありえます。たとえば検索文字列 'foo'に '\|bar'を追加すると、カーソルは 'foo' より前にある最初の 'bar'に戻ります (この再計算の発生をユーザーに通知するために "Pending" を示すプロンプトに変更される)。Section 12.6 [Regexps], page 115 を参照してください。

前方および後方への regexp 検索は、対照的ではありません。なぜなら Emacs での regexp に対するマッチは常に前方へ処理され、regexp の先頭から開始されるからです。したがって前方への regexp 検索は前方にスキャンし、可能性のある開始位置から前方へとマッチを試みます。後方への regexp 検索は後方へスキャンし、可能性のある開始位置から前方へとマッチを試みます。これらの検索手法はミラーイメージではありません。

regexp にたいする非インクリメンタルな検索は、コマンド re-search-forwardおよび re-search-backwardで行われます。これらのコマンドは M-xから呼び出すか、インクリメンタル regexp 検索からの C-M-s RETおよび C-M-r RETで呼び出します。M-xを使用してこれらのコマンドを呼び出した場合は、指定した regexp を厳密に検索するので、case folding を除く lax-search 機能 (Section 12.9 [Lax Search], page 120 を参照してください) はサポートされません

プレフィクス引数を指定したインクリメンタル regexp 検索は、isearch-forwardや isearch-backwardのような、通常の文字列を検索します。Section 12.1 [Incremental Search], page 105 を参照してください。

12.6 正規表現の構文

このセクション (および、マニュアル全般) では、ユーザーが通常使う正規表現の機能を説明します。主に Lisp プログラムで使用される追加の機能については、Section "Regular Expressions" in *The Emacs Lisp Reference Manual* を参照してください。

正規表現は、いくつかの特殊文字 *(special constructs)* と、それ以外の普通の文字からなる構文を持ちます。通常の文字はそれと同じ文字にマッチし、それ以外の文字にはマッチしません。特殊文字は、'$^.*+?[\'です。文字 ']'は、文字候補を終了させる場合は特殊文字です (以下参照)。文字 '-' は、文字候補の中では特殊文字です。正規表現の中に現れるその他の文字は、前に '\'がついてない限り普通の文字です (Lisp プログラム内で正規表現を使う場合、'\'は 2 つ記述しなければなりません。このセクションの最後にある例を参照してください)。

たとえば 'f'は特殊文字ではなく普通の文字なので、正規表現中の 'f'は文字列 'f'にマッチし、他の文字列にはマッチしません (文字列 'ff'にはマッチしません)。同様に正規表現中の 'o'は、'o'だけにマッチします (大文字小文字を区別しない場合、これらの regexp は 'F'や 'O'にもマッチしますが、これを例外としてではなく、"同じ文字列" を一般化したものと考えます)。

2 つの正規表現 *a*と *b*を結合できます。結合した結果は、*a*が先頭の適当な部分に一致して、*b*が残りの部分に一致する正規表現となります。些細な例としては、'f'と 'o'という正規表現を結合すると、

正規表現 ‘fo’となり、これは文字列 ‘fo’だけにマッチします。ここまでは普通です。これより複雑ことを行うには、特殊文字を使う必要があります。以下にリストを示します。

. (ピリオド)
: これは改行文字以外の任意の 1 文字にマッチする特殊文字です。たとえば正規表現 ‘a.b’は、最初が ‘a’で最後が ‘b’の 3 文字の文字列にマッチします。

*
: 単独では使用されません。これは接尾演算子で、前の正規表現の任意の回数、可能な限り多くの回数を含めた繰り返しを意味します。したがって ‘o*’は任意の個数の ‘o’にマッチし、‘o’がない場合 (0 個の場合) も含めてマッチします。

  ‘*’は常に可能な限り小さな、前置表現に適用されます。したがって ‘fo*’は ‘fo’の繰り返しではなく ‘o’の繰り返しです。この正規表現は ‘f’、‘fo’、‘foo’、... にマッチします。

  マッチングにより構成される ‘*’は、見つけられるだけの反復回数へと、直ちに処理されます。その後でパターンの残りの部分の処理を続けます。これが失敗すると、バックトラッキングが発生します。‘*’の反復回数をいくつか捨てて、パターンの残りの部分がマッチするように構成を変更します。たとえば文字列 ‘caaar’にたいして ‘ca*ar’をマッチさせる場合、まず最初に ‘a*’を、3 つすべての ‘a’にマッチさせます。しかしパターンの残りの部分は ‘ar’ですが、マッチさせるために残っているのは ‘r’だけなので、このマッチは失敗します。かわりに ‘a*’を 2 つの ‘a*’だけにマッチさせます。この選択により、regexp の残りの部分のマッチが成功します。

+
: これは ‘*’と同様に接尾演算子ですが、前置表現に最低 1 回マッチしなければならない点が異なります。したがって ‘ca+r’は ‘car’、‘caaaar’にマッチしますが、‘cr’にはマッチしません。一方 ‘ca*r’は、これら 3 つすべての文字列にマッチします。

?
: これは ‘*’と同様に接尾演算子ですが、前置表現が 1 回出現するか、出現しないかいずれかという点が異なります。したがって ‘ca?r’は、‘car’または ‘cr’のいずれかになります。

*?, +?, ??
: これらは上述した演算子の、非貪欲 *(greedy)* なタイプの演算子です。通常の演算子 ‘*’、‘+’、‘?’は、regexp がマッチする、できる限り長いマッチを行います。しかしこれらの演算子に ‘?’を後置すると、できる限り短いマッチを行います。

  したがって ‘ab*’と ‘ab*?’は、両方とも文字列 ‘a’と文字列 ‘abbbb’にマッチしますが、文字列 ‘abbb’にマッチさせたとき、‘ab*’は文字列全体 (有効な最長マッチ) にマッチしますが、‘ab*?’は ‘a’(有効な最短マッチ) だけにマッチします。

  非貪欲な演算子は、与えられた開始位置から開始される、できるだけ短い文字列にマッチします。前方検索では、利用できる一番最初の開始位置は、常にカーソルの位置となります。したがって末尾が改行のテキスト ‘abbab’にたいして ‘a.*?$’を検索すると、文字列全体にマッチします。これはマッチが最初の ‘a’から開始されので、マッチさせることができるのです。

[...]
: これは ‘[’で始まり ‘]’で終わる文字候補集合 *(set of alternative characters)*、あるいは文字集合 *(character set)* です。

  もっとも簡単なケースでは、2 つのカッコの間に指定された文字が、マッチできる文字集合となります。したがって ‘[ad]’は 1 つの ‘a’か 1 つの ‘d’にマッチし、‘[ad]*’は ‘a’と ‘d’からなる任意の文字列にマッチします (空の文字列を含む)。結果として ‘c[ad]*r’は、‘cr’、‘car’、‘cdr’、‘caddaar’、... にマッチします。

  文字集合には、開始文字と終了文字の間に ‘-’を記述することにより、文字の範囲を含めることもできます。したがって ‘[a-z]’は、ASCII小文字となります。文字の範囲と特定

の文字の指定を混ぜることもできます。‘[a-z$%.]’は任意の ASCII小文字と ‘$’、‘%’、‘.’にマッチします。別の例としては、‘[α-ω^^ce^^af]’はギリシャ文字の小文字すべてにマッチします。

いくつかの特別な文字クラス *(character classes)* を文字集合に含めることもできます。文字候補集合を含む文字クラスは‘[:’と‘:]’で囲んで指定します。たとえば‘[[:alnum:]]’は、任意のアルファベットと数字にマッチします。文字クラスのリストは、Section “Char Classes” in *The Emacs Lisp Reference Manual* を参照してください。

文字集合に‘]’を含めるには、それを最初に記述しなければなりません。たとえば‘[]a]’は、‘]’または‘a’にマッチします。文字集合に‘-’を含めるには、‘-’を文字集合の最後に記述しますが、範囲の先頭や後にも記述できます。したがって‘[]-]’は、‘]’と‘-’の両方にマッチします。

文字集合に‘^’を含めるには、集合の最初以外に記述します (最初に記述した場合、補集合を指定したことになります。以下を参照してください)。

大文字小文字を区別しない検索で文字の範囲を使う場合、範囲の先頭と最後を、大文字だけ、または小文字だけで記述するか、先頭と最後をアルファベット以外で記述するべきです。‘A-z’のような大文字小文字を混成した範囲指定は不正な定義で、Emacs の将来のバージョンで変更されるかもしれません。

`[^ ... ]` ‘[^’は文字の補集合 *complemented character set* を開始します。つまり指定された文字以外がマッチします。したがって‘[^a-z0-9A-Z]’は、ASCII文字と数字以外にマッチします。

‘^’は文字集合で最初以外で使用された場合、特別な意味をもちません。‘^’に続く文字は、先頭にあるものとして扱われます (いいかえると‘-’と‘]’は、ここでは特別な意味をもちません)。

もしマッチしない文字として改行が記述されていなければ、文字の補集合を改行にマッチさせることができます。これは grepのようなプログラムにおける、regexp にたいする処理とは対照的です。

`^` これは空文字列、ただしテキストの行頭だけにマッチする特殊文字です。それ以外ではマッチに失敗します。したがって‘^foo’は行の先頭にある‘foo’にマッチします。

歴史的な互換性により、この意味での‘^’の使用は、正規表現の先頭か、‘\(’または‘\|’の後に記述された場合に限ります。

`$` ‘^’と似ていますが、行末だけにマッチします。したがって‘x+$’は、行末にある 1 つ以上の‘x’にマッチします。

歴史的な互換性により、この意味での‘$’の使用は、正規表現の最後か、‘\)’または‘\|’の後に記述された場合に限ります。

`\` これには 2 つの機能があります。まず特殊文字 (‘\’を含む) をクォートすることと、追加の特別な構成を導入することです。

‘\’は特殊文字をクォートするので、正規表現中の‘\$’は‘$’だけにマッチし、‘\[’は‘[’だけにマッチします。

‘\’で始まる特別な構成については、以下のセクションを参照してください。

注意: 歴史的な互換性から、特殊文字はそれが特殊な意味をもたないようなコンテキストで使用された場合は、通常の文字として扱われます。たとえば‘*foo’は、‘*’が特殊文字として動作するための前置された表現がないので、普通の文字として扱われます。このような振る舞いに依存することは、

よい習慣ではありません。特殊文字を記述する場合、それがどこに記述されようとクォートするべきです。

文字候補の中では '\'は特別ではなく '-'、'^'、']'のもつ特別な意味を除去することはありません。これらの文字が特別な意味をもたないような場所にあるときは、これらの文字もクォートすべきではありません。これでは明確ではないかもしれません。これらの特殊文字が特別な意味をもつ場所にあるとき、バックスラッシュを前置することによりクォートされるのです。たとえば '[^\]'(Lisp の文字記法では"[^\\]") は、バックスラッシュ以外の任意の一文字にマッチします。

12.7 正規表現でのバックスラッシュ

多くの場合、任意の文字を伴う '\'はその文字だけに一致します。しかしいくつか例外があって、'\'で始まる 2 文字のシーケンスが、特別な意味を持つ場合があります。シーケンス内の 2 文字目にくる文字は、単独で使った場合には普通の文字として扱われるものです。以下は '\'の構成の表です。

`\|` 選択肢を指定します。2 つの正規表現 *a*と *b*の間に '\|'を記述すると、それは *a*または *b* のいずれかにマッチする表現を形成します。これはまず *a*とのマッチを試み、失敗した場合に *b*とのマッチを試みます。

したがって 'foo\|bar'は、'foo'または 'bar'のいずれかにマッチし、それ以外の文字列にはマッチしません。

'\|'は、周囲の一番大きな表現に適用されます。'\|'のグループ化の能力に制限をかけることができるのは、周囲の '\(... \)'によるグループ化だけです。

複数の '\|'使用を処理するための、完全なバックトラッキング能力が存在します。

`\( ... \)` 3 つの目的のためのグループ化構成です:

1. 他の操作に使うために、一連の選択肢 '\|'を括ります。したがって '\(foo\|bar\)x'は、'foox'または 'barx'のいずれかにマッチします。
2. 接尾演算子 '*'、'+'、'?'を適用できるように、複雑な正規表現を括ります。したがって 'ba\(na\)*'は、'bananana'のように、(0 個以上の) 文字列 'na'にマッチします。
3. あとで参照できるように、マッチした部分文字列を記録します。

この最後の使い方は、カッコでグループ化することが重要なのではありません。これは '\\(... \\)' 構成の、2 番目の意味とは異なる機能です。実際には、これら 2 つの機能が衝突することは、通常はありません。もし衝突するようなら、以下で説明する、shy(内気) なグループ化を使うことができます。

`\(?: ... \)`
マッチした部分文字列を記録しない、*shy(*内気*)* なグループ化を指定します。マッチした部分文字列は、'*d*'により後方参照できません (以下参照)。この機能は正規表現を機械的にまとめるときに役立ちます。これにより後方参照するためのグループにたいする番号づけに影響することなく、文法的な目的によるグループ化を行うことができます。

`\`*d* *d*番目に '\(... \)'構成にマッチしたテキストと同じテキストにマッチします。これは後方参照 *(back reference)* と呼ばれます。

最後の '\(... \)'構成の後で、マッチ処理はこの構成にマッチしたテキストの最初と最後を記録します。そして正規表現の後の部分で '\'の後に数字 *d*を使うことにより、*d* 番目の '\(... \)'構成にマッチしたテキストと同じテキストにマッチさせることができます。

正規表現に記述された最初の 9 つの ‘\(... \)’にマッチしたテキストは、正規表現で開きカッコが出現した順に、1 から 9 までの数字が割り当てられます。そのため ‘\1’から ‘\9’を使うことにより、‘\(... \)’構成にマッチした、対応するテキストを参照することができます。

たとえば ‘\(.*\)\1’は改行を含まない、前半と後半が同一の文字列にマッチします。‘\(.*\)’は、そのテキストが何であろうと前半にマッチしますが、‘\1’は前半と正確に同じテキストにマッチしなければなりません。

もし特定の ‘\(... \)’構成が 1 回以上マッチする場合 (これは ‘*’が後置されているとき簡単に発生します)、最後のマッチだけが記録されます。

`\{`*m*`\}` これは *m*回の繰り返しを指定する接尾演算子です。つまり前置される正規表現に、正確に *m*回連続でマッチしなければなりません。たとえば ‘x\{4\}’は、文字列 ‘xxxx’だけにマッチします。

`\{`*m*`,`*n*`\}` これは *m*回から *n*回の繰り返しを指定する接尾演算子です。つまり前置される正規表現が最低 *m*回、最大 *n*回マッチしなければなりません。*n*が省略されたときはマッチ回数の上限はありませんが、前置される正規表現は少なくとも *m*回マッチしなければなりません。
‘\{0,1\}’は ‘?’と同じです。
‘\{0,\}’は ‘*’と同じです.
‘\{1,\}’は ‘+’と同じです。

\` 空の文字列にマッチしますが、文字列またはバッファー (またはアクセスできる部分) の先頭に限定されます。

\' 空の文字列にマッチしますが、文字列またはバッファー (またはアクセスできる部分) の最後に限定されます。

\= 空の文字列にマッチしますが、ポイント位置に限定されます。

\b 空の文字列にマッチしますが単語の先頭または最後に限定されます。したがって ‘\bfoo\b’は、区切られた単語 ‘foo’にマッチします。‘\bballs?\b’は、別々の単語 ‘ball’または ‘balls’にマッチします。

‘\b’は、そこにどんなテキストが出現しようと、バッファーの先頭または最後にもマッチします。

\B 空の文字列にマッチしますが、単語の最初と最後以外にマッチします。

\< 空の文字列にマッチしますが、単語の先頭に限定されます。‘\<’は単語の構成文字が続く場合に限り、バッファーの先頭にマッチします。

\> 空の文字列にマッチしますが、単語の最後に限定されます。‘\<’は内容が単語の構成文字で終わる場合に限りバッファーの最後にマッチします。

\w 任意の単語構成文字にマッチします。どの文字が該当するかは、構文テーブル (syntax table) により決定されます。Section “Syntax Tables” in *The Emacs Lisp Reference Manual* を参照してください。

\W 単語構成文字以外の任意の文字にマッチします。

< 空の文字列にマッチしますが、シンボルの先頭に限られます。シンボルは 1 文字以上のシンボル構成文字からなります。シンボル構成文字は、‘w’と ‘’の構文をもつ文字です。‘_<’はシンボル構成文字が続く場合に限り、バッファーの先頭にもマッチします。単語

にたいしては、構文テーブル (syntax table) が、どの文字がシンボル構成文字かを判断します。

`\_>` 空の文字列にマッチしますが、シンボルの最後に限定されます。'`\_>`'は内容がシンボル構成文字で終わる場合に限り、バッファーの最後にマッチします。

`\sc` 構文が *c*である、任意の文字にマッチします。ここで *c*とは、特定の構文クラスを表す文字です。したがって'`w`'は単語構成文字、'`-`'または' 'は空白文字、'`.`'は通常の区切り文字、などとなります。Section "Syntax Class Table" in *The Emacs Lisp Reference Manual* を参照してください。

`\Sc` 構文が *c*以外の、任意の文字にマッチします。

`\cc` カテゴリー *c*に属する、任意の文字にマッチします。たとえば '`\cc`'は中国文字、'`\cg`'はギリシャ文字にマッチします。既知のカテゴリーについての説明は、`M-x describe-categories RET`をタイプしてください。

`\Cc` カテゴリー *c*に属さない、任意の文字にマッチします。

単語と構文に関係する構成は構文テーブルのセッティングにより制御されます。See Section "Syntax Tables" in *The Emacs Lisp Reference Manual*.

12.8 正規表現の例

以下に regexp の例を示します。これは Emacs がセンテンスの最後 (末尾の空白は含まない) を認識するために、デフォルトで使用する regexp(たとえば変数 `sentence-end-base`) と似ています。

```
[.?!][]\"')}]*
```

これには 2 つの連続する部分があります。1 つは '`.`'、'`?`'、'`!`'にマッチする文字です。もう 1 つは閉じカッコ、クォート、カッコの 0 回以上の繰り返しです。

12.9 検索中の Lax マッチング

あなたは通常、タイプした文字と、検索されるテキストの間にある、特定の瑣末な違いを、検索コマンドが無視することを望むでしょう。たとえば長さが異なる空白文字シーケンスは通常、等しいとみなされ、大文字小文字の違いは通常問題にならない、などです。これは等価文字 *(character equivalence)* として知られています。

このセクションでは Emacs の lax search(緩い検索) 機能と、それを必要に応じて調整する方法について説明します。

デフォルトでは、検索コマンドは *lax space matching(*緩いスペースマッチング*)* を行います。この検索ではスペースおよび一連のスペースは、テキスト中の 1 つ以上の空白文字にマッチします。正確に言うと、Emacs は検索文字列内の空白文字のシーケンスそれぞれにたいして、ユーザーオプション `search-whitespace-regexp`で指定されている正規表現をマッチします。このオプションのデフォルトには、スペースとタブのシーケンスすべてを空白とみなす値がセットされています。したがって '`foo bar`' は '`foo bar`'、'`foo  bar`'、'`foo  bar`'、... にマッチします ('`foobar`'にはマッチしない)。スペースやタブと同じように一連の改行をスペースとしてマッチさせるには、このオプションの値を正規表現 '`[[:space:]\\n]+`'をカスタマイズしてください (インクリメンタルな regexp 検索のデフォルトの振る舞いは異なる。Section 12.5 [Regexp Search], page 114 を参照)。

空白文字を正確にマッチさせたい場合は、インクリメンタル検索中に`M-s SPC` (`isearch-toggle-lax-whitespace`) とタイプすることにより、lax space matching をオフに切り替えることができます。もう 1 度 `M-s SPC`とタイプすると、lax space matching ふぁオンに切り替わります。すべて

の検索で lax space matching を無効にするには、`search-whitespace-regexp`を `nil`に変更します。これにより検索文字列の中のスペースは、正確に 1 つのスペースにマッチするようになります。

Emacs での検索では、検索文字列を小文字で指定した場合、デフォルトでは検索するテキストの大文字小文字は区別されません。したがって‘`foo`’を検索すると、‘`Foo`’や‘`fOO`’もマッチします。regexp、特に文字集合でも同様に振る舞います。つまり‘`[ab]`’は、‘`a`’、‘`A`’、‘`b`’、‘`B`’もマッチします。この機能は *case folding* として知られており、これはインクリメンタル検索と非インクリメンタル検索の両方でサポートされています。

検索文字列のどこかに大文字があると、検索は case-sensitive(大文字小文字を区別する) になります。したがって‘`Foo`’を検索すると、‘`foo`’や‘`FOO`’は検索されません。これは正規表現検索でもリテラル文字列検索と同様に適用されます。検索文字列から大文字を削除すると、効果はなくなります。変数 `search-upper-case`がこれを制御します。この変数が非 `nil`の場合、検索文字列の中の大文字は検索を case-sensitive にします。これを `nil`にセットすることにより、大文字によるこの効果は無効になります。この変数のデフォルト値は検索文字列中に大文字があれば検索を case-sensitive にして、検索文字列に yank するテキスト (Section 12.1.3 [Isearch Yank], page 107 を参照) を小文字にする (そうすればデフォルトで検索が case-insensitive になる)`not-yanks`です。

変数 `case-fold-search`に `nil`をセットすると、すべての文字は大文字小文字を含めて、完全にマッチしなければなりません。これはバッファーごとの変数で、変数の変更はデフォルト値を変えない限り、通常はカレントバッファーだけが影響を受けます。Section 33.2.3 [Locals], page 511 を参照してください。これは置換コマンド (Section 12.10 [Replace], page 122 を参照してください) や、ミニバッファーのヒストリー検索 (Section 5.5 [Minibuffer History], page 36 を参照してください) を処理する、非インクリメンタル検索にも適用されます。

インクリメンタル検索で `M-c`、または `M-s c` (`isearch-toggle-case-fold`) とタイプすると、検索が大文字小文字を区別するかが、切り替わります。この効果は、現在の検索を超えて引き継がれませんが、カレントの検索にたいして大文字を追加・削除したときの効果をオーバーライドします.

特定のコマンドや操作にたいする検索やマッチングにおいて、大文字小文字の区別を制御する変数がいくつかあります。たとえば `tags-case-fold-search`は、`find-tag`での大文字小文字の区別を制御します。これらの変数を探すには、`M-x apropos-variable RET case-fold-search RET`とタイプしてください。

case folding では、文字の大文字小文字の違いを無視するので、大文字は小文字にマッチし、その逆もマッチします。case folding を一般化したものが *character folding* で、これは類似した文字間の違いの、より広いクラスを無視します。たとえば character folding では、文字 `a`は `ä`や `á`のようなアクセント付きの類似文字にもマッチし、これらの変種を区別するための特殊記号も無視します。加えて `a`は、`a`に似ている他の文字、それに U+00AA FEMININE ORDINAL INDICATOR (訳注: スペイン語などで女性形の序数を表すために数字の後に加えられる上付きの小さな `a`) や U+24D0 CIRCLED LATIN SMALL LETTER A (丸で囲まれた小さい `a`のような外観) のように文字のグラフィカルな外観の一部に `a`をもつ文字にもマッチします。同様に ASCIIのダブルクォート文字`"`は、ダブルクォートの変種として Unicode 標準で定義されている、他のすべての変種にマッチします。最後に、character folding により、1 つ以上の文字シーケンスは違う長さの他の文字にもマッチするようになります。たとえば 2 つの文字 `ff`は U+FB00 LATIN SMALL LIGATURE FF、`(a)`というシーケンスは U+249C PARENTHESIZED LATIN SMALL LETTER A にマッチします。文字シーケンスはまったく同じではありませんが、character folding でのマッチは *equivalent character sequences(*等価文字シーケンス*)* として知られています。

一般的に Emacs の検索コマンドは、デフォルトでは等価な文字シーケンスのマッチのために character folding を行ないません。変数 `search-default-mode`を `char-fold-to-regexp`にカスタマイズすることにより、この振る舞いを有効にできます。Section 12.12 [Search Customizations],

page 129 を参照してください。インクリメンタル検索では、M-s ' (isearch-toggle-char-fold) とタイプすることにより character folding が切り替わりますが、これはその検索だけです (置換コマンドは別のオプションによる異なるデフォルトをもちます。Section 12.10.3 [Replacement and Lax Matches], page 123 を参照してください)。

デフォルトでは äのような文字の明示的な変種を検索文字列の一部としてタイプしても、それは a のようなベース文字とはマッチしません。しかし変数 char-fold-symmetricを tにカスタマイズすれば、検索コマンドは等価な文字を同一に扱い、検索文字列内での等価な文字集合の使用により、検索するテキスト内の等価な文字を検索するので、アクセントつきの文字 äは文字 aと同じように áのようなすべての変種にもマッチします。

カスタマイズ可能な変数 char-fold-includeを使用して新たな folding を追加したり、カスタマイズ可能な変数 char-fold-excludeを使用して既存の folding を削除できます。char-fold-overrideを tにカスタマイズすれば、char-fold-includeを用いて追加したもの以外の等価文字すべてを無効にすることができます。

12.10 置換コマンド

Emacs は検索と置換を行うコマンドをいくつか提供します。単純な M-x replace-stringコマンドに加えて、出現する検索パターンごとに置換するかを問い合わせる M-% (query-replace) も提供します。

置換コマンドは通常、ポイント位置からバッファーの最後までのテキストにたいして処理を行います。リージョンがアクティブのときは、リージョンにたいして処理を行います (Chapter 8 [Mark], page 52 を参照してください)。基本的な置換コマンドは 1 つの検索文字列 (または regexp) を、1 つの置換文字列で置き換えます。コマンド expand-region-abbrevsを使用して、複数の置換を並行して処理することが可能です (Section 26.3 [Expanding Abbrevs], page 374 を参照してください)。

12.10.1 無条件の置換

M-x replace-string RET *string* RET *newstring* RET
: すべての *string*を *newstring*で置換します。

ポイントの後にある ‘foo’のすべてのインスタンスを ‘bar’に置換するには、コマンド M-x replace-stringに 2 つの引数 ‘foo’と ‘bar’を指定します。置換はポイントの後だけで発生するので、バッファー全体を置換したい場合は、最初にバッファーの先頭に移動しなければなりません。バッファーの最後までのすべてが置換されます。置換をバッファーの一部に制限したいときは、そのリージョン部分をアクティブにします。リージョンがアクティブのときは、置換はそのリージョンに制限されます (Chapter 8 [Mark], page 52 を参照してください)。

replace-stringが終了したとき、ポイントは最後に置換された位置に留まります。以前のポイント位置 (replace-stringコマンドを実行した場所) はマークリングに追加されるので (マークは非アクティブ)、C-u C-SPCで戻ることができます。Section 8.4 [Mark Ring], page 56 を参照してください。

プレフィクス引数を指定すると、置換対象は単語単位に制限されます。

置換コマンドでの case-sensitivity(大文字小文字の区別) と character folding については、Section 12.10.3 [Replacement and Lax Matches], page 123 を参照してください。

12.10.2 正規表現の置換

M-x replace-stringコマンドは、正確にマッチする単一の文字列を置換します。同様なコマンド M-x replace-regexpは、指定した正規表現パターン (Section 12.6 [Regexps], page 115 を参照してください) にマッチするすべてを置換します。

`M-x replace-regexp RET` *regexp* `RET` *newstring* `RET`
*regexp*にマッチするすべてを *newstring*で置換します。

`replace-regexp`では、*newstring*が定数である必要はありません。*regexp*にマッチした全体、または部分を参照することができます。*newstring*での '`\&`'は、置換されるマッチ全体を表します。*newstring*での '`\`*d*'(*d*は 1 から始まる数字) は、*regexp*内でカッコでグループ化されたものの、*d*番目にマッチします (これは "後方参照 (back reference) と呼ばれます")。'`\#`'は、このコマンドですでに置換された件数を 10 進数で参照します。最初の置換では '`\#`'は '`0`'で、2 番目の置換では '`1`'、... のようになります。たとえば、

```
M-x replace-regexp RET c[ad]+r RET \&-safe RET
```

これは '`cadr`'を '`cadr-safe`'に、'`cddr`'を '`cddr-safe`'に置き換えます。

```
M-x replace-regexp RET \(c[ad]+r\)-safe RET \1 RET
```

これは逆向きの置換をします。置換するテキストに '`\`'を含めるときは、'`\\`'と入力しなければなりません。

置換する文字列の一部を毎回手入力したいときは、置換文字列で '`\?`'を使用します。すると置換ごとにミニバッファーで置換文字列を編集できます (ポイント位置は '`\?`'を記述した場所です。)

このサブセクションの残りの部分は、Lisp の知識が必要となる特別な処理を念頭に書かれています。大半の読者はスキップしても構いません。

置換文字列の一部を計算するために、Lisp 式を使うことができます。これを行うには、置換文字列の中で、'`\,`'に続けて Lisp 式を記述します。各置換において式の値が計算され、それをクォートされていない文字列に変換します (もし文字列の場合は、その文字列の内容が使われることを意味します)。そしてそれを置換文字列内で、式が記述された場所に使用します。もし式がシンボルのときは、シンボル名と、置換文字列中のシンボル名の後にあるの間に 1 つスペースは、両方シンボルの値に置換されます。

このような式の中では、いくつかの特別なシーケンスを使うことができます。式の中での '`\&`'や '`\`*d*'は、通常のようにマッチした文字列全体と、部分マッチした文字列を参照します。*d*には複数桁の数字を記述でき、カッコでグループ化された *d*番目の正規表現がマッチしなかったとき、'`\`*d*'は `nil`になります。'`\#&`'と '`\#`*d*'を使って、それらのマッチを数字で参照することもできます (これはマッチまたは部分マッチが数字書式の場合に有効です)。ここでの '`\#`'も、すでに置換された数を意味します。

たとえば '`x`'と '`y`'の入れ替えは、以下の方法で行うことができます:

```
M-x replace-regexp RET \(x\)\|y RET
\,(if \1 "y" "x") RET
```

'`\,`'により置換する文字列を計算するには、`format`関数が便利なときがあります (Section "Formatting Strings" in *The Emacs Lisp Reference Manual* を参照してください)。たとえば 73 列目から 80 列目 (もしそこに何もなければ) に、'`ABC00042`'のような連番を振りたいときは、以下を使うことができます。

```
M-x replace-regexp RET ^.\{0,72\}$ RET
\,(format "%-72sABC%05d" \& \#) RET
```

12.10.3 置換コマンドと Lax マッチ

このサブセクションでは lax マッチに関する置換コマンドの振る舞いと、それをカスタマイズする方法を説明します。一般的には、ほとんどの置換は、それと同等な検索コマンドに比べて、デフォルトではより厳密なマッチを行ないます。

インクリメンタル検索とは異なり、置換コマンドはデフォルトでは lax space matching(緩いスペースマッチング) を行いません (Section 12.9 [Lax Search], page 120 を参照してください)。置

換で lax space matching を有効にするには、変数 replace-lax-whitespaceを非 nilに変更してください (これは Emacs が置換文字列ではなく、置換するテキストを検索する方法だけに影響を与えます)。

query-replace-regexpがパターンを検索するとき lax whitespace matching を使うかどうかを制御するのは、それに対応する変数 replace-regexp-lax-whitespaceです。

置換コマンドの最初の引数がすべて小文字の場合、置換のための検索において大文字小文字の違いを無視します。これは case-fold-searchと search-upper-caseがいずれも非 nilの場合です。search-upper-case (Section 12.9 [Lax Search], page 120 を参照) が nilなら、検索が大文字小文字を無視するかどうかは、コマンドの 1 つ目の引数が小文字かどうかと無関係に case-fold-search単独で決定されます。case-fold-searchを nilにセットすると、すべての検索において常に大文字小文字の違いが有効になります。

さらに置換コマンドの 2 つ目の引数のすべて、または一部が小文字の場合には、置換コマンドは大文字小文字が出現するパターンを維持しようと試みます。したがって以下のコマンド、

```
M-x replace-string RET foo RET bar RET
```

は小文字の 'foo'を小文字の 'bar'で置換し、すべて大文字の 'FOO'を 'BAR'、そして最初が大文字の 'Foo'を 'Bar'に置換します。(これら 3 つの候補、すなわち小文字、すべて大文字、先頭が大文字は、replace-stringが認識できる唯一のパターンです)。

置換文字列に大文字が使われている場合、テキストが挿入されるときは、常に大文字のままとなります。大文字が最初の引数で使用されている場合、大文字小文字の変換なしで、2 番目の引数に与えられたとおりに置換されます。同様に case-replaceと case-fold-searchの両方が nilにセットされている場合、大文字小文字の変換なしで置換されます。

デフォルトでは置換コマンドは、置換するテキストを探すとき、character folding(Section 12.9 [Lax Search], page 120 を参照してください) を使用しません。query-replaceと replace-stringでのマッチングで character folding を有効にするには、変数 replace-char-foldに非 nil値をセットします (このセッティングは、Emacs が置換するテキストを探す方法だけに影響し、置換するテキストには影響を与えません。また、replace-regexpにも影響を与えません)。

12.10.4 問い合わせつき置換

M-% *string* RET *newstring* RET
: 任意の *string*を *newstring*で置換します。

C-M-% *regexp* RET *newstring* RET
: *regexp*にたいする任意のマッチを *newstring*で置換します。

'foo'を 'bar'に置換するとき、すべてではなく、そのうちのいくつかだけを置換したいときは、M-% (query-replace) を使います。このコマンドは 'foo'を 1 つずつ検索して、それを置換するかを毎回尋ねます。この問い合わせを別とすれば、query-replaceは replace-stringと同様に機能します (Section 12.10.1 [Unconditional Replace], page 122 を参照してください)。通常のように、case-replaceが非 nilのときは、大文字小文字を区別します (Section 12.10.3 [Replacement and Lax Matches], page 123 を参照してください)。数引数を指定すると、単語区切り文字で区切られた単語だけを考慮します。負のプレフィクス引数は後方に置換します。

C-M-% (query-replace-regexp) は、regexp の検索と置換を行います。これは query-replaceのように問い合わせを行う以外は、replace-regexpと同様に機能します。

これらのコマンドで行なった以前の置換を再利用できます。query-replaceや query-replace-regexpが検索文字列の入力を求めるプロンプトを表示しているとき、M-pと M-nを使用することにより、'*from* -> *to*'という形式で、以前の置換を表示できます。ここで *from*は検索パターン、*to*はそれ

の置換、これらの間に表示されるセパレータは変数 query-replace-from-to-separatorの値により決定されます。望む置換が表示されたら、RETとタイプしてそれを選択します。この変数の値がnilの場合、置換はコマンドヒストリーに追加されず、再利用できません。

これらのコマンドは、カレントのマッチを、フェイス query-replaceを使ってハイライトします。変数query-replace-highlightをnilにセットすることにより、このハイライトを無効にできます。他のマッチのハイライトにはインクリメンタル検索 (Section 12.1 [Incremental Search], page 105 を参照してください) と同様に、フェイス lazy-highlightが使われます。変数 query-replace-lazy-highlightを nilにセットすることにより、このハイライトを無効にできます。デフォルトでは query-replace-regexpは、カレントマッチを置換する展開後の文字列を、ミニバッファーに表示します。特別なシーケンス '\&'および '\n'を展開せずに維持するには、query-replace-show-replacement変数をカスタマイズしてください。インクリメンタル検索中に search-highlight-submatchesが副式 (subexpression) をハイライトするように (Section 12.12 [Search Customizations], page 129 を参照)、変数 query-replace-highlight-submatchesは regexp の置換コマンド中で副式をハイライトするかどうかを定義します。

変数query-replace-skip-read-onlyに非nilがセットされている場合、置換コマンドはread-only(読み取り専用) のテキスト内のマッチを無視します。デフォルトでは、それらを無視しません。

以下は文字列または regexp にたいするマッチが表示されているときにタイプできる文字です:

SPC
y マッチを *newstring*で置き換えます。

DEL
Delete
BACKSPACE
n カレントマッチを置換せずに次のマッチへスキップします。

, (カンマ) カレントマッチを置換して、結果を表示します。そして次に何をするかを文字入力するよう促します。置換がすでに行われているので、この状況では DELと SPCは等価で、どちらも次のマッチへ移動します。

ここで C-r(以下を参照) をタイプして、置換されたテキストを編集できます。undoコマンド (C-x uとタイプする。Section 13.1 [Undo], page 131 を参照されたい) で置換をアンドゥすることもできます。置換を取り消すこともできます。これは query-replaceを終了させるので、さらに置換を行う場合は、C-x ESC ESC RETで置換を再開しなければなりません (Section 5.6 [Repetition], page 38 を参照)。

RET
q これ以上の置換を行わずに終了します。

. (ピリオド)
カレントマッチを置換してから、これ以上の検索を行わずに終了します。

! これ以上の問い合わせをせずに、残りのマッチをすべて置換します。

^ 前のマッチの位置に戻ります。これは間違えて変更したときや、再検証したい場合に使います。

u 最後の置換をアンドゥ(undo: 取り消し) して、その置換が行われた位置に戻ります。

U すべての置換をアンドゥして、最初の置換が行われた位置に戻ります。

C-r 再帰編集レベル (recursive editing level) に入ります。これはマッチを *newstring*で置換するだけでなく、編集したい場合に使用します。編集を終えたら C-M-cで再帰編集レ

ベルを抜けて、次のマッチを処理します。Section 31.11 [Recursive Edit], page 484 を参照してください。

`C-w` マッチを削除してから、`C-r`と同様に再帰編集レベルに入ります。これは *string*を削除してから、テキストを挿入することにより置換を行う場合に使用します。編集を終えたら `C-M-c`で再帰編集レベルを抜けて、次のマッチを処理します。

`e` 置換文字列をミニバッファーで編集します。`RET`でミニバッファーを抜けると、カレントマッチをミニバッファーの内容で置換します。この新しい置換文字列は、残りのマッチにたいしても適用されます。

`E` これは `e`と似ていますが、次の置換を case(大文字小文字) に厳格に行います。つまり '`foo`'から '`bar`'に `query-replace`していたら、'`Foo`'のようなテキストは通常は '`Bar`' に置き換えられます。カレントの置換を case に厳格に行うには、このコマンドを使用してください。

`C-l` スクリーンを再描画します。その後でカレントマッチにたいして何を行うか、別の文字をタイプして指定しなければなりません。

`Y` (大文字) 複数バッファーの置換で、残りのバッファーの、残りのマッチをすべて置換します (これは選択したファイルにたいして問い合わせつきの置換を行う、Dired の `Q`コマンドと似ています)。これはすべての一連の問い合わせにたいして、これ以上のユーザーとの対話なしに "yes" を答えます。

`N` (大文字) 複数バッファーの置換で、カレントバッファーの残りのマッチを置換せずに、次のバッファーへスキップします。これはカレントバッファーのマッチにたいする問い合わせに "no" を答えて、次のバッファーへと処理を続けます。

`C-h`
`?`
`F1` 上述したオプションの要約を表示します。その後でカレントマッチにたいして何を行うか、別の文字をタイプして指定しなければなりません。

これらのエイリアス文字以外の文字は、`query-replace`を終了してから、キーシーケンスの残りの部分を読みとります。したがって `C-k`とタイプすると、`query-replace`を終了してから、行末までを kill します。特に、`C-g`は単に `query-replace`を exit します。

一度終了した `query-replace`を再開するには、`C-x ESC ESC`を使います。`query-replace`は引数の読み取りにミニバッファーを使っているので、このキーシーケンスで再開させることができます。Section 5.6 [Repetition], page 38 を参照してください。

オプション `search-invisible`は、`query-replace`が非表示のテキストを扱う方法を決定します。[Outline Search], page 265 を参照してください。

選択されたファイルにたいして問い合わせつきの置換を行う、Dired の `Q`コマンドについては、Section 27.7 [Operating on Files], page 387 を参照してください。regexp にマッチするファイル名にたいして、ファイル名の変更、ファイルのコピー、ファイルのリンクを行う Dired のコマンドについては、Section 27.10 [Transforming File Names], page 393 を参照してください。

12.11 その他の検索およびループコマンド

ここでは正規表現にたいするマッチを検索する、その他のコマンドを説明します。これらのコマンドは、パターンに大文字が含まれていないカ、`case-fold-search`が非 `nil`のときは、マッチングで

大文字小文字を区別しません。常にバッファー全体を検索する multi-occurと multi-occur-in-matching-buffersをのぞき、これらのコマンドはすべてポイント位置からバッファーの最後まで、リージョンがアクティブなときはそのリージョンにたいして処理を行います。

M-x multi-isearch-buffers
: １つ以上のバッファー名の入力を求め (RETで終了)、それらのバッファーにたいして複数バッファーのインクリメンタル検索を開始します (あるバッファーでの検索に失敗すると、次の C-sにより、指定された次のバッファーへと検索を試みます)。プレフィクス引数を指定すると、regexp の入力を求め、regexp にマッチするバッファーにたいして、複数バッファーでのインクリメンタル検索を開始します。

M-x multi-isearch-buffers-regexp
: このコマンドは multi-isearch-buffersと同様ですが、インクリメンタルな regexp 検索を行います。

M-x multi-isearch-files
: １つ以上のファイル名の入力を求め (RETで終了)、それらのファイルにたいして複数ファイルのインクリメンタル検索を開始します (あるファイルで検索に失敗すると、次の C-sにより、指定された次のファイルへと検索を試みます)。プレフィクス引数を指定すると、regexp の入力を求め、regexp にマッチするファイルにたいして、複数ファイルでのインクリメンタル検索を開始します。

M-x multi-isearch-files-regexp
: このコマンドは multi-isearch-filesと同様ですが、インクリメンタルな regexp 検索を行います。

  バッファーローカルな変数 multi-isearch-next-buffer-functionをセットするいくつかのモード (たとえば Change Log モード) では、複数ファイルにたいするインクリメンタル検索は自動的にアクティブになります。

M-x occur
M-s o
: regexp の入力を求め、それへのマッチを含むバッファー内の各行をリスト表示します。プロンプトで M-nをタイプした場合は、前のインクリメンタル検索から検索文字列を再利用できます。マッチするテキストは、matchフェイスを使用してハイライトされます。数引数 *n*を指定すると、そのコンテキストでマッチした各行の、前後 *n*行を表示します。

  コンテキスト行のデフォルト行数は、変数 list-matching-lines-default-context-linesにより指定されます。list-matching-lines-jump-to-current-lineが非 nilのとき、カレント行はフェイス list-matching-lines-current-line-faceでハイライト表示され、ポイントはその行の最初のマッチの後にセットされます。

  インクリメンタル検索がアクティブのときは M-s oを実行して、カレントのサーチリングを使うこともできます。

  あなたがタイプした regexp にたいするマッチは完全な行を含むように拡張され、1 つ前のマッチの終了の前から開始するマッチは考慮されないことに注意してください。

  *Occur*バッファーはメジャーモードとして Occur モードを使用します。次または前のマッチに移動するために nおよび pのキーを使うことができます。プレフィックス数引数を指定した場合には、その個数分のマッチを移動します。数字キーは digit-argumentにバインドされているので、5 nなら次の 5 番目のマッチに移動します (C-uのタイプは不要)。SPCおよび DELはそれぞれ*Occur*バッファーを上または下へとスクロールします。マッチをクリックするかそこにポイントを移動して RETをタイプすれば、検索した元

バッファーの対応する位置を visit します。oおよびC-oはマッチを別ウィンドウに表示しますが、C-oはソンウィンドウを選択しません。マッチを 1 つずつ visit するために、かわりに M-g M-n (next-error) を使用できます (Section 24.2 [Compilation Mode], page 310 を参照)。最後に qは*Occur*バッファーを表示しているウィンドウを quit するとともに、そのバッファーをバリーします。

*Occur*バッファーで eをタイプすることにより、バッファーを編集可能にするとともに Occur Edit モードに切り替えます。このモードではマッチした行を編集することができ、更に編集結果は元のバッファーのテキストに適用されます。C-c C-cとタイプすることに Occur Edit モードを抜けて Occur モードに戻ります。

コマンド M-x list-matching-linesは、M-x occurの別名です。

M-x multi-occur
: このコマンドは occurと同じですが、複数のバッファーを検索する点が異なります。このコマンドは、1 つずつバッファー名の入力を求めます。

M-x multi-occur-in-matching-buffers
: このコマンドは multi-occurと似ていますが、visit しているファイル名にマッチする正規表現を指定することにより、検索するバッファーを指定する点が異なります。プレフィクス引数を指定すると、正規表現にマッチするバッファー名となります。

M-x how-many
: regexp の入力を求め、バッファーのポイント位置以降に、何個のマッチがあるか表示します。リージョンがアクティブのときは、リージョンにたいして操作を行います。

M-x flush-lines
: regexp の入力を求めて、ポイントの後のテキストでマッチを含む各行を削除します。コマンドの終了時には削除したマッチ行の行数をプリントします。

  カレント行のポイント以降にマッチする文字列が含まれているときはカレント行を削除します。リージョンがアクティブのときはリージョンにたいして操作を行います。行の一部ににリージョンが含まれていて、さらにマッチの全体がリージョンに含まれているときには行は削除されます。

  マッチが行をまたいでいる場合、flush-linesはそれらの行すべてを削除します。このコマンドは行を削除してから、次のマッチを検索します。したがって、1 行にマッチ全体と、次の行にまたがるマッチが両方含まれるとき、次の行にまたがるマッチは無視されます。

M-x keep-lines
: regexp の入力を求め、ポイント以降にあるテキストにたいして、マッチを含まない行を削除します。ポイントが行の先頭でない場合、このコマンドは常にカレント行を維持します。リージョンがアクティブのときは、リージョンにたいして操作を行います。このコマンドは一部がリージョンに含まれるだけの行は削除しません (行を終端する改行は、その行の一部と判断します)。

  マッチが行をまたぐ場合、このコマンドはそれらすべての行を維持します。

M-x kill-matching-lines
: flush-linesと同様ですがマッチした行は kill リングへも追加されます。このコマンドは行区切りの改行を含めて、マッチした行を単一の文字列として kill リングに追加します。

`M-x copy-matching-lines`
: `kill-matching-lines`と同様ですが、マッチした行をバッファーから削除しません。

12.12 必要に応じて検索を調整する

このセクションでは、他では説明されていない、その他の検索に関連した機能を説明します。

インクリメンタル検索にたいするデフォルトの検索モードは、変数 `search-default-mode`により指定されます。この変数には、`nil`、`t`、または関数を指定できます。`nil`の場合、デフォルトのモードは character folding なしのリテラル検索です。しかし、case folding にたいしては `case-fold-search`、lax-whitespace match にたいしては `search-whitespace-regexp`により決定されます (Section 12.9 [Lax Search], page 120 を参照してください)。値が`t`の場合、インクリメンタル検索のデフォルトは regexp 検索になります。デフォルト値には、case folding と lax-whitespace matching だけを行なう関数が指定されています。

継続的なインクリメンタル検索でのカレントマッチは、`isearch`フェイスでハイライトされます。変数 `search-highlight`を `nil`にセットすることにより、このハイライトを無効にできます。

(たとえば `C-M-s`による) 正規表現の検索時には、`search-highlight-submatches`変数に応じて部分式は特別にハイライトされます。この変数の値が `nil`なら特別なハイライトは行われませんが、値が非 `nil`なら正規表現内の '`\( ... \)`'構文にマッチするテキスト (いわゆる "部分式") は異なるフェイスでハイライトされます。デフォルトでは `isearch-group-1`および `isearch-group-2`という名前の 2 つの異なるフェイスが定義されています。これら 2 つのフェイスでは奇数番目の部分式は `isearch-group-1`フェイス、偶数番目の部分式は `isearch-group-2`フェイスを使用してハイライトされます。たとえば '`foo-\([0-9]+\)\([a-z]+\)`'の検索時には '`[0-9]+`'がマッチする部分が `isearch-group-1`フェイス、'`[a-z]+`'がマッチする部分は `isearch-group-2`フェイスを使用してハイライトされます。`isearch-group-3`、`isearch-group-4`、...のようにこの同じ番号付けスキームを使用してフェイスを追加定義する場合には M、*N*+*M*、2*N*+*M*、...番目の部分式のハイライトにフェイス `isearch-group-`*M*が使用されます。ここで Nは `isearch-group-`*M*形式のフェイス総数です。

ディスプレイに表示されている、検索文字列にたいするその他のマッチは、`lazy-highlight`フェイスを使用してハイライトされます。変数 `isearch-lazy-highlight`をセットして、このハイライトを無効にできます。以下は lazy-highlight をカスタマイズする、その他の変数です:

`lazy-highlight-initial-delay`
: 可視なマッチをハイライトする前に待機する秒数です。検索文字列が `lazy-highlight-no-delay-length`文字より短い場合のみ適用されます。

`lazy-highlight-no-delay-length`
: 検索文字列の長さが少なくともこの変数の値であれば、マッチの遅延ハイライトは即座に開始されます。

`lazy-highlight-interval`
: マッチをハイライトする秒数です。

`lazy-highlight-max-at-a-time`
: 入力をチェックする前にハイライトする、マッチの最大数です。大きな数を指定するとハイライトに時間を要するかもしれず、その間に検索を継続するために `C-s`や `C-r`をタイプしても、それらすべてのマッチのハイライトを終了するまで、Emacs は反応しないでしょう。したがって小さな数を指定することにより、Emacs の反応を改善できます。

`isearch-lazy-count`
: 検索プロンプトにカレントマッチ数とマッチ総数を表示します。

`lazy-count-prefix-format`
`lazy-count-suffix-format`
: これらの 2 つの変数は `isearch-lazy-count`の表示にたいするカレントマッチ数とマッチ総数のフォーマットを決定します。

インクリメンタル検索で検索文字列が空のときに `RET`を入力すると、通常これは非インクリメンタル検索を開始します (実際には、これにより検索文字列の編集が開始され、次の `RET`で検索を行ないます)。しかし、変数 `search-nonincremental-instead`を `nil`にセットした場合、検索文字列が空でも、`RET`のタイプは常にインクリメンタル検索を exit します。

デフォルトではインクリメンタル検索および問い合わせ付き置換コマンドは不可視のテキストにもマッチしますが、そのようなマッチは隠され、不可視のテキストの外にできるだけ早くカレントマッチを移動させます。変数 `isearch-hide-immediately`を `nil`にカスタマイズした場合、マッチが見つかった任意の不可視テキストは、検索または置換コマンドが exit するまで表示され続けます。

遅い回線で接続されたリモートマシンのディスプレイなど、遅い端末でインクリメンタルな検索をすると、検索によりディスプレイの大きな範囲を再描画しなければならないことが、煩わしくなるかもしれません。Emacs は、遅い端末にたいして特別なディスプレイモードを提供します。これは検索が別の小さなウィンドウをポップアップして、マッチの周辺テキストをそのウィンドウに表示します。小さいウィンドウは早く表示できるので、遅いスピードの影響による煩わしさは軽減されます。変数 `search-slow-speed`は、Emacs がこのディスプレイモードを使用する、ボーレートの下限値を決定します。変数 `search-slow-window-lines`は、Emacs がポップアップして検索結果を表示するウィンドウの行数を制御します (デフォルトは 1)。このウィンドウは通常、検索を開始したバッファーを表示するウィンドウの下部にポップアップされますが、`search-slow-window-lines`の値が負の場合、ウィンドウは上部に配され、`search-slow-window-lines`の絶対値が表示される行数になります。

13 タイプミスを訂正するコマンド

このチャプターでは、編集中に間違いに気付いたときに便利なコマンドを説明します。これらのコマンドの中でもっとも基本的なのは、undo コマンド C-/です (これは C-x uと C-_にもバインドされています)。これは 1 つのコマンドやコマンドの一部 (query-replaceの場合など)、または連続するいくつかの文字の挿入を取り消します。連続して C-/を繰り返すと、前へ前へと変更を取り消し、undo 情報が利用できる限界までさかのぼって、変更を取り消します。

ここで説明するコマンドとは別に、DEL (delete-backward-char) のような削除コマンドでもテキストを消去できます。これらのコマンドは、このマニュアルの前のチャプターで説明しています。Section 4.3 [Erasing], page 21 を参照してください。

13.1 Undo(取り消し)

undo(取り消し) コマンドは、バッファーのテキストにたいする最後の変更を無効にします。バッファーはそれぞれ変更を個別に記録しており、undo コマンドは常にカレントバッファーに適用されます。バッファーのレコードにより、バッファーにたいするすべての変更を undo できます。通常、個々の変種コマンドは、undo レコードに個別のエントリーを作成しますが、query-replaceのようないくつかのコマンドは、undo 操作に柔軟性をもたせるために、コマンドによる変更を複数のエントリーに分割します。連続した文字の挿入コマンドは、undo 操作の冗長性をなくすために、通常 1 つの undo レコードにまとめられます。

C-/
C-x u
C-_ カレントバッファーの undo レコードの、1 エントリーで undo します (undo)。

undo を開始するには、C-/(またはエイリアスの C-_か C-x u) をタイプします [1]。これは一番最近のバッファーにたいする変更を取り消して、バッファーが変更される前の位置にポイントを戻します。連続して C-/(またはそれのエイリアス) を繰り返すと、現在のバッファーにたいする変更を、前へ前へとさかのぼって取り消します。すでに記録されている変更がすべて取り消されているとき、undo コマンドはエラーをシグナルします。

undo 以外のコマンドは、undo コマンドの順序性を損ないます。undo 以外のコマンドを開始した時点から、undo してきた一連の undo コマンド全体が、undo レコードとして記録されます。したがって undo した変更を再適用するには、undo の順序性を損なわないような C-fのようなコマンドをタイプしてから、C-/を 1 回以上タイプして、undo コマンドを undo していきます。

以前の undo コマンドを再実行せずに undo を再開したいときは、かわりに M-x undo-onlyを使います。これは undoと同様ですが、すでに undo した変更を redo(再実行) しません。これを補うために、M-x undo-redoは前の undo コマンドを undo します (そしてそのコマンド自身をアンドゥ可能なコマンドとして記録しない)。

バッファーをうっかり変更してしまったのに気付いたら、モードラインのアスタリスクが表示されなくなるまで C-/を繰り返しタイプするのが、もっとも簡単な復旧方法です。undo コマンドによりモードラインのアスタリスクが消えたとき、それはバッファーの内容がファイルを最後に読み込んだとき、または保存したときと同じ内容だということを意味します。バッファーを意図して変更したか覚えていないときは、C-/を 1 回タイプします。最後の変更が undo されたのを確認して、それが意

[1] C-/以外に、undoコマンドは C-x uにもバインドされています。なぜならこれは初心者が記憶するのが簡単だからです。'u'は "undo" に由来しています。このコマンドは C-_にもバインドされています。なぜならいくつかのテキスト端末では、C-/とタイプすることにより、C-_が入力されるからです。

図した変更なのか確かめます。もしそれが意図しない変更だったなら undo したままにします。意図した変更だった場合、上で説明した方法で変更を再実行します。

かわりに `M-x revert-buffer`を使用して、そのバッファーが最後に visit されたとき、または最後に保存されたとき以降のすべての変更を破棄できます (Section 15.4 [Reverting], page 157 を参照)。

アクティブなリージョンがあるとき、`undo`は選択的な *undo(selective undo)* を行います。これはバッファー全体ではなく、リージョンにたいして一番最近の変更を undo します。しかし Transient Mark モードがオフのとき (Section 8.7 [Disabled Transient Mark], page 57 を参照してください)、`C-/`は、リージョンではなく、常にバッファー全体を操作します。この場合 `undo`コマンドにプレフィクス引数を指定 (`C-u C-/`) することにより、選択的な undo ができます。同じリージョンにたいしてさらに変更を undo したいときは、`undo`コマンドを繰り返します (プレフィクス引数は必要ありません)。

undo レコードを作らない、特別なバッファーがいくつかあります。それは名前がスペースで始まるバッファーです。これらのバッファーは Emacs により内部的に使用されており、通常ユーザーが閲覧したり編集しないテキストを保持します。

バッファーにたいする undo 情報が大きくなりすぎたとき、Emacs は一番古いレコードを、時々 (ガベージコレクション *(garbage collection)* の間) 廃棄します。変数 `undo-limit`、`undo-strong-limit`、`undo-outer-limit`をセットすることにより、どれだけの undo 情報を保持するか指定することができます。これらの値はバイト数で指定します。

変数 `undo-limit`は、ソフトリミットをセットします。Emacs はこのサイズに達するまでのコマンドの undo データを保持します。制限を超えることもあり得ますが、これを超える古いデータは保持しません。デフォルト値は 160000 です。変数 `undo-strong-limit`は、厳密なリミット (stricter limit) をセットします。この制限を超えるような以前のコマンド (一番最近のコマンドではない) は記憶されません。`undo-strong-limit`のデフォルト値は 240000 です。

これらの変数の値にかかわらず、一番最近の変更は undo 情報が `undo-outer-limit`(通常は 24,000,000) より大きくならない限り、廃棄されることはありません。もしこれを超えるような場合、Emacs は undo データを廃棄して、それにたいする警告を発します。これは一番最近のコマンドを undo できない唯一の状況です。これが発生した場合、将来同じことが起こらないように `undo-outer-limit`の値を増やすことができます。しかしコマンドがそのような大きな undo データを作るとは考えられない場合、それは多分バグなので、それを報告すべきです。Section 34.3 [Reporting Bugs], page 542 を参照してください。

13.2 テキストの入れ替え

`C-t` 2 つの文字を入れ替えます (`transpose-chars`)。

`M-t` 2 つの単語を入れ替えます (`transpose-words`)。

`C-M-t` バランスのとれた 2 つの式を入れ替えます (`transpose-sexps`)。

`C-x C-t` 2 つの行を入れ替えます (`transpose-lines`)。

`M-x transpose-sentences`
2 つのセンテンスを入れ替えます (`transpose-sentences`)。

`M-x transpose-paragraphs`
2 つのパラグラフを入れ替えます (`transpose-paragraphs`)。

`M-x transpose-regions`
2 つのリージョンを入れ替えます。

2 つの文字を入れ違いに入力してしまう間違いはよくありますが、それらが隣接しているときは C-t(transpose-chars) コマンドで訂正できます。通常 C-tはポイントの左右にある文字を入れ替えます。行末でこのコマンドを使う場合、最後の文字と改行を入れ替えるのはおそらく無意味なので、C-tは行末の 2 文字を入れ替えます。このような入れ違いミスにすぐ気付いたときは、C-tだけで訂正できます。すぐに間違いに気付かなかった場合、C-tをタイプする前に、入れ替えて入力してしまった文字の間にカーソルを移動しなければなりません。単語の最後の文字とスペースを入れ替えて入力してしまった場合、そこに戻るには単語移動コマンド (M-f、M-bなど) が良い方法です。それ以外では、逆向きの検索 (C-r) が最善な方法の場合があります。Chapter 12 [Search], page 105 を参照してください。

M-tは、ポイントの前にある単語と、後にある単語を入れ替えます (transpose-words)。このコマンドはポイントより後方、またはポイントがある単語を前方に移動して、ポイントをその後に移動します。文字の間にある区切り文字は移動しません。たとえば 'FOO, BAR' は、'BAR FOO,' ではなく、'BAR, FOO' に入れ替わります。

C-M-t (transpose-sexps) も似ていますが、これは 2 つの式 (Section 23.4.1 [Expressions], page 292 を参照してください)、C-x C-t (transpose-lines) は行を入れ替えます。M-x transpose-sentencesと M-x transpose-paragraphsはそれぞれセンテンスとパラグラフを入れ替えます。これらのコマンドは M-tと同様に動作しますが、入れ替えを行うテキスト単位が異なります。

入れ替えコマンドに数引数を指定すると、反復回数になります。これは入れ替えコマンドに、ポイントより前 (またはポイントがある) の移動させる文字 (または単語、式、行) を、何文字 (または単語、式、行) 後に移動させるか指示します。たとえば C-u 3 C-t は、ポイントの前にある文字を 3 文字後ろに移動します。つまり 'f⋆oobar'は 'oobf⋆ar'に変更されます。これは C-tを 3 回繰り返したのと同じです C-u - 4 M-tはポイントの前にある単語を、後方に単語 4 つ分移動します。C-u - C-M-t は、引数を指定しない C-M-tの効果を取り消します。

数引数 0 には特別な意味が割り当てられています (繰り返し回数としては、0 に意味がありません)。これはポイントの後ろで終わる文字 (または単語、式、行) を、マーク位置で終わる文字 (または単語、式、行) と入れ替えます。

M-x transpose-regionsはポイントとマークの間のテキストを、マークリングに最後に push された 2 つのマークの間のテキストと入れ替えます (Section 8.1 [Setting Mark], page 52 を参照)。プレフィクス数引数を与えるとポイントとマークの間のテキストを、その個数分マークリングを戻ってから連続する 2 つのマークの間のテキストと入れ替えます。このコマンドは複数の文字 (や単語、センテンス、パラグラフ) を一度に行うために最適です。

13.3 大文字小文字の変換

M-- M-l 最後に入力した単語を小文字に変換します。Meta--はメタとマイナスであることに注意して下さい。

M-- M-u 最後に入力した単語をすべて大文字に変換します。

M-- M-c 最後に入力した単語の先頭を大文字、それ以外を小文字に変換します。

単語の大文字小文字を間違えてタイプするのは、とても一般的な間違えです。そのため、単語の大文字小文字の変換コマンド M-l、M-u、M-cに負の数引数を指定すると、カーソルを移動しないという特性があります。最後に入力した単語のタイプミスに気付いたら、単に大文字小文字を変換して、タイプを続けることができます。Section 22.7 [Case], page 260 を参照してください。

13.4 スペルのチェックと訂正

このセクションでは、1 つの単語、またはバッファーの一部のスペルをチェックするコマンドを説明します。これらのコマンドは Hunspell、Aspell、Ispell、または Enchant のうちのいずれかのスペルチェッカープログラムがインストールされている場合だけ機能します。これらのプログラムは Emacs の一部ではありませんが、GNU/Linux または他のフリーなオペレーティングシステムには、通常 1 つはインストールされています。

インストールされたスペルチェッカーが 1 つだけなら、ここで説明するコマンドのいずれかを最初に呼び出したときに、Emacs はそれを見つけるでしょう。インストールされているのが複数なら、変数 `ispell-program-name`をカスタマイズしてどれを使用するか制御できます。

`M-$`
: ポイントがある単語スペルの、チェックと訂正をします (`ispell-word`)。リージョンがアクティブのときは、リージョンに含まれるすべての単語にたいして行います。

`M-x ispell`
: バッファーにある単語スペルの、チェックと訂正をします。リージョンがアクティブのときは、リージョンに含まれるすべての単語にたいして行います。

`M-x ispell-buffer`
: バッファーにある単語スペルの、チェックと訂正をします。

`M-x ispell-region`
: リージョンにある単語スペルの、チェックと訂正をします。

`M-x ispell-message`
: メールメッセージのドラフト (引用部分を除く) にたいして、単語スペルのチェックと訂正をします。

`M-x ispell-comments-and-strings`
: バッファーやリージョンにあるコメントと文字列のスペルチェックと訂正をします。

`M-x ispell-comment-or-string-at-point`
: ポイント位置のコメントと文字列をチェックします。

`M-x ispell-change-dictionary RET` *dict* `RET`
: スペルチェッカーのプロセスを再起動して、*dict*を辞書として使用させます。

`M-x ispell-kill-ispell`
: スペルチェッカーのサブプロセスを kill します。

`M-TAB`
`ESC TAB`
`C-M-i`
: ポイントの前にある単語を、スペル辞書をもとに補完します (`ispell-complete-word`)。

`M-x flyspell-mode`
: スペルミスした単語をハイライトする、Flyspell モードを有効にします。

`M-x flyspell-prog-mode`
: コメントと文字列にたいして、Flyspell モードを有効にします。

ポイントの前または周囲にある単語スペルをチェックしたり訂正するには、`M-$` (`ispell-word`) をタイプします。リージョンがアクティブのときは、リージョン内のすべての単語スペルをチェックします。Chapter 8 [Mark], page 52 を参照してください (Transient Mark モードがオフのとき、`M-$`

はリージョンを無視して、常にポイントの前または周囲の単語にたいして動作します。Section 8.7 [Disabled Transient Mark], page 57 を参照してください)。

同様に、コマンド `M-x ispell` はリージョンがアクティブのときはリージョン、それ以外のときはバッファー全体にたいしてスペルチェックを行います。コマンド `M-x ispell-buffer` および `M-x ispell-region` は、スペルチェックをバッファー全体にたいして行うのか、リージョンにたいして行うかを明示します。このコマンドはバッファー全体をチェックしますが、インデントされているテキストと、以前のメッセージの引用と思われるテキストはチェックしません。Chapter 29 [Sending Mail], page 420 を参照してください。ソースコードを扱う際には、コメントや文字列リテラルだけをチェックするために `M-x ispell-comments-and-strings`や `M-x ispell-comment-or-string-at-point` を使用できます。

これらのコマンドは、正しくないと思われる単語に出会うと、それをどうするかを尋ねます。通常は番号が振られた近い単語 *(near-misses:* 正しくないと思われる単語に似た単語*)* のリストを表示します。これにたいして 1 文字をタイプして応答しなければなりません。以下は有効な応答文字です。

数字
: 今回だけは近い単語の一覧で単語を置き換えます。近い単語には番号が振られており、選択するにはその番号をタイプします。

`SPC`
: その単語を間違いと判断することは変更しませんが、その単語をここでは変更せずスキップします。

`r` *new* `RET`
: 今回は単語を *new*で置き換えます (置換文字列にスペルエラーがないか再スキャンされます)。

`R` *new* `RET`
: 単語を *new*で置き換え、`query-replace`を実行します。これによりバッファーの他の箇所の間違えを置換できます (置換にスペルエラーがないか再スキャンされます)。

`a`
: 正しくないと思われる単語を正しいものとして受け入れますが、それは今回の編集セッションに限られます。

`A`
: 正しくないと思われる単語を正しいものとして受け入れますが、それは今回の編集セッションとそのバッファーに限られます。

`i`
: この単語をプライベートな辞書ファイルに挿入するので、今後 (将来のセッションを含めて) はその単語を正しいと認識します。

`m`
: `i`と同様ですが、辞書の補完情報も指定できます。

`u`
: この単語の小文字バージョンを、プライベートな辞書ファイルに挿入します。

`l` *word* `RET`
: *word*にマッチする単語を辞書から探します。これらの単語は近い単語の新しいリストになるので、置換で数字をタイプして選択できるようになります。*word*にはワイルドカードとして '`*`'を使うことができます。

`C-g`
`X`
: 対話的なスペルチェックを終了します (ポイントはチェックされた単語の位置に残されます)。`C-u M-$`でチェックを再開できます。

`x`
: 対話的なスペルチェックを終了します (ポイントはチェックを開始したときの位置に戻ります)。

`q`
: 対話的なスペルチェックを終了して、スペルチェッカーのサブプロセスを kill します。

`?`
: オプションの一覧を表示します。

Text モードおよびそれに関連するモードでは、M-TAB (ispell-complete-word) で、スペル訂正にもとづくバッファー内補完を行います。単語の最初の部分を入力してから M-TABをタイプすると、補完一覧が表示されます (ウィンドウマネージャーが M-TABを横取りしてしまう場合は、ESC TAB または C-M-iをタイプします)。補完対象には番号か文字が振られているので、その番号か文字をタイプして選択します。

1 度プロセスが開始されれば、スペルチェッカーのサブプロセスは待機状態で実行を続けるので、一連のスペルチェックコマンドは素早く完了します。プロセスを終了させたいときは M-x ispell-kill-ispell を使います。スペルの訂正を行うとき以外プロセスが CPU 時間を使うことはないので、これは通常必要ありません。

スペルチェッカーは、スペルを 2 つの辞書から探します。それは標準辞書と個人用辞書です。標準辞書は変数 ispell-local-dictionaryで指定されます。これが nilのときは、変数 ispell-dictionaryで指定されます。両方が nilのとき、スペルプログラムは既定の辞書を使います。コマンド M-x ispell-change-dictionary は、バッファーにたいする標準辞書をセットしてからサブプロセスを再起動することにより、これにより異なる標準辞書を使うことができます。個人用の辞書は、変数 ispell-personal-dictionaryで指定します。これが nilのとき、スペルプログラムは個人辞書を、各スペルチェッカーが指定する既定の場所から探します。

単語の補完には別の辞書が使われます。変数 ispell-complete-word-dictは、この辞書のファイル名を指定します。補完辞書は、単語の変化形を検知するためにスペルチェックが使用する、単語にたいする root 語 (接辞を含まない語) と affix 語 (接辞語) に関する情報を使用できないので、、別の辞書でなければなりません。補完辞書はもたないが、スペルチェック辞書はあもつ言語もいくつかあります。

Flyspell モードは、タイプしたテキストのスペルをタイプ時に自動的にチェックするマイナーモードです。認識できない単語を見つけると、その単語をハイライトします。M-x flyspell-mode とタイプすると、カレントバッファーの Flyspell モードを切り替えます。すべての Text バッファーで Flyspell モードを有効にするには、text-mode-hookに、flyspell-modeを追加します。Section 33.2.2 [Hooks], page 509 を参照してください。Flyspell モードは、ユーザーの移動にしたがって各単語をチェックする必要があるので、カーソル移動やスクロールコマンドにより速度低下するかもしれないことに注意してください。タイプしていないテキストや移動と関係ないテキストの自動チェックも行いません。それを行うためには、flyspell-regionや flyspell-bufferを使用してください。

Flyspell モードが間違ったスペルの単語をハイライトしているとき、それを mouse-2 (flyspell-correct-word) でクリックして、可能な定性とアクションを表示するメニューを表示できます。このメニューをかわりに mouse-3にしたければ、context-menu-modeを有効にしてください。加えて C-.、または ESC TAB (flyspell-auto-correct-word) はポイント位置の単語にたいする可能な訂正を提案し、C-c $ (flyspell-correct-word-before-point) は可能な訂正のメニューをポップアップします。もちろん、間違ったスペルの単語を、常にお好みの方法で手修正できます。

Flyspell Prog モードは通常の Flyspell モードと同じように機能しますが、コメントと文字列に含まれた単語だけをチェックします。この機能はプログラムを編集する場合に便利です。M-x flyspell-prog-mode をタイプして、現在のバッファーにたいするこのモードの有効と無効を切り替えます。すべてのプログラミングに関連したモードでこのモードを有効にするには、prog-mode-hookに flyspell-prog-modeを追加します (Section 33.2.2 [Hooks], page 509 を参照してください)。

14 キーボードマクロ

このチャプターでは一連の編集コマンドを記録して、後で簡単に繰り返す方法を説明します。

キーボードマクロ *(keyboard macro)* とは、Emacs のユーザーにより定義される、一連のキー入力からなるコマンドです。たとえば C-n M-d C-dを 40 回繰り返しタイプしていることに気付いたとしましょう。C-n M-d C-dを行うキーボードマクロを定義して、それを 39 回以上繰り返すことにより、作業スピードをあげることができます。

キーボードマクロは、コマンドを実行・記録することにより定義します。違う言い方をすると、キーボードマクロの定義では、初回はマクロの定義が実行されるということです。この方法により、頭だけで考えるのではなく、コマンドの影響を目で見ることができます。コマンド列の入力を終了して定義を終了するときは、キーボードマクロが定義されるとともに、入力したコマンド列の影響としてマクロが 1 回実行されたことになります。その後はマクロを呼び出すことにより、コマンド列全体を実行することができます。

キーボードマクロは、Lisp ではなく Emacs コマンド言語 (Emacs command language) で記述されている点が、通常の Emacs コマンドと異なります。しかし Emacs コマンド言語は、高度なことや一般的なことを記述するプログラム言語として、充分にパワフルとはいえません。そのような事項には、Lisp を使わなければなりません。

14.1 基本的な使い方

F3 キーボードマクロの定義を開始します (kmacro-start-macro-or-insert-counter)。

F4 キーボードマクロを定義しているときは定義を終了します。それ以外の場合は一番最近のキーボードマクロを実行します (kmacro-end-or-call-macro)。

C-u F3 最後のキーボードマクロを再実行してから、キーをマクロ定義に追加します。

C-u C-u F3
最後のキーボードマクロを再実行せずに、キーをマクロ定義に追加します。

C-x C-k r リージョンの中の各行の行頭にたいして、最後のキーボードマクロを実行します (apply-macro-to-region-lines)。

C-x ((旧スタイル) キーボードマクロの定義を開始します (kmacro-start-macro)。プレフィクスキーを与えると最後のマクロにキーを追加します。

C-x) (旧スタイル) マクロ定義を終了します (kmacro-end-macro)。プレフィクス引数はそのマクロ自体を除外した繰り返し回数として機能します。

C-x e もっとも最後に定義したキーボードマクロを実行します (kmacro-end-and-call-macro)。プレフィクス引数は繰り返し回数として機能します。

キーボードマクロの定義を開始するには、F3をタイプします。それからはキーを入力して実行を続けますが、それは同時にマクロ定義の一部になります。その間は、モードラインに 'Def'が表示されて、マクロの定義中であることを示します。終了するときは F4 (kmacro-end-or-call-macro) をタイプして、定義を終了します。たとえば、

```
F3 M-f foo F4
```

これは 1 単語前方に移動してから、'foo'を挿入するマクロを定義します。F3と F4は、マクロの一部とはならないことに注意してください。

マクロを定義した後は、F4でそれを呼び出すことができます。上記の例では、それは M-f foo again をタイプしたのと同じ効果をもちます (F4コマンドの 2 つの役割に注意してください。これはマクロを定義しているときはマクロの定義を終了し、そうでないときは最後のマクロを呼び出します)。F4に数引数 'n'を与えることもできます。これはマクロを 'n'回呼び出すことを意味します。引数に 0 を与えると、エラーになるか、C-g(MS-DOS では C-Break) をタイプするまで、マクロを永久に繰り返します。

上記の例は、キーボードマクロを使った便利なトリックをデモンストレイトする例です。テキストの一定間隔の位置にたいして繰り返し操作を行いたいときは、マクロの一部に移動コマンドを含めます。この例ではマクロの繰り返しにより、連続する単語の後ろに文字列 'foo'を挿入していきます。

キーボードマクロの定義を終了した後でも、C-u F3をタイプすることにより、マクロの定義にキーストロークを追加できます。これは F3に続けてマクロの定義を再タイプするのと同じです。結果として、そのマクロの以前の定義が再実行されることになります。変数 kmacro-execute-before-appendを nilに変更すると、既存のマクロにキーストロークが追加されるまでは実行されません (デフォルトは t)。最後に実行したキーボードマクロを再実行することなく、定義の最後にキーストロークを追加するには、C-u C-u F3をタイプしてください。

コマンドがミニバッファーから引数を読みとる場合、ミニバッファーにたいする入力は、コマンドと一緒にマクロの一部となります。したがってマクロを再生すると、そのコマンドの引数は入力されたのと同じになります。たとえば、

```
F3 C-a C-k C-x b foo RET C-y C-x b RET F4
```

これはカレント行を kill して、バッファー 'foo'にそれを yank した後、元のバッファーに戻ります。

ほとんどのキーボードコマンドは、キーボードマクロの定義で普通に機能しますが、いくつか例外があります。C-g (keyboard-quit) をタイプすると、キーボードマクロの定義が終了します。C-M-c (exit-recursive-edit) は信頼できません。これはマクロの中で再帰編集 (recursive edit) を開始したときは期待通りに再帰編集から抜け出しますが、キーボードマクロの呼び出し前に開始された再帰編集を抜け出すには、キーボードマクロからも抜け出す必要があります。同様に、マウスイベントもキーボードマクロで使用できますが。信頼はできません。マクロによりマウスイベントが再生されるときは、マクロを定義したときのマウス位置が使用されます。この効果は予測が困難です。

コマンド C-x C-k r (apply-macro-to-region-lines) は、リージョン内の各行のにたいして、最後に定義されたキーボードマクロを繰り返します。これは 1 行ずつポイントをリージョン内の行頭に移動してからマクロを実行します。

上記で説明した F3と F4に加えて、Emacs はキーボードマクロを定義したり実行するための、古いキーバインドもサポートします。F3と同様にマクロ定義を開始するには、C-x ((kmacro-start-macro) とタイプします。プレフィクス引数を指定すると、最後のキーボードマクロの定義に追加します。マクロ定義を終了するには C-x) (kmacro-end-macro) とタイプします。一番最近のマクロを実行するには、C-x e (kmacro-end-and-call-macro) とタイプします。マクロ定義中に C-x e を入力すると、マクロ定義を終了してからすぐに実行されます。C-x eをタイプした後すぐに eをタイプすることにより、そのマクロを 1 回以上繰り返すことができます。(マクロの実行に使用されるときの)F4と同様、C-x eには繰り返し回数を引数指定できます。

C-x)に、繰り返し回数を引数として与えることができます。これはマクロを定義した後、すぐにマクロが繰り返されることを意味します。マクロの定義は、定義することによりマクロが実行されるので、最初の 1 回として数えられます。したがって C-u 4 C-x)は、マクロを 3 回すぐに追加実行します。

長時間キーボードマクロを実行中に、(どれほど実行されたかを示すために) 再表示をトリガーできれば便利なときがあるかもしれません。このために C-x C-k dを使用できます。とても便利とは言

えない例ですが、C-x (M-f C-x C-k d C-x)はC-u 42 C-x eと実行した際に、繰り返しごとに再表示を行うマクロを作成します。

14.2 キーボードマクロリング

すべての定義されたキーボードマクロは、キーボードマクロリング *(keyboard macro ring)* に記録されます。キーボードマクロリングはすべてのバッファーで共有され、1 つだけしかありません。

C-x C-k C-k
: リングの先頭にあるキーボードマクロを実行します (kmacro-end-or-call-macro-repeat)。

C-x C-k C-n
: キーボードマクロリングを、次のマクロ (古く定義されたもの) にローテートします (kmacro-cycle-ring-next)。

C-x C-k C-p
: キーボードマクロリングを前のマクロ (新しく定義されたもの) にローテートします (kmacro-cycle-ring-previous)。

キーボードマクロリングを操作するすべてのコマンドは、同じ C-x C-kを使います。これらのコマンドでは、すぐ後にコマンドを実行して繰り返す場合には、互いに C-x C-kプレフィクスを必要としません。たとえば、

```
C-x C-k C-p C-p C-k C-k C-k C-n C-n C-k C-p C-k C-d
```

これは、キーボードマクロリングを 2 つ前のマクロが先頭にくるようにローテートして、3 回実行します。次にキーボードマクロリングをローテートして、元は先頭だったマクロを先頭に戻して 1 回実行します。次にキーボードマクロリングを 1 つ前のマクロが先頭にくるようにローテートして、それを実行します。そして最後にそれを削除しています。

コマンド C-x C-k C-k (kmacro-end-or-call-macro-repeat) は、マクロリングの先頭にあるキーボードマクロを実行します。もう一度すぐに C-kをタイプすると、マクロを繰り返すことができます。すぐに C-nか C-pをタイプすれば、マクロリングをローテートすることができます。

キーボードマクロを定義しているとき、C-x C-k C-kは F4と同様に振る舞いますが、すぐ後にタイプされた場合は異なります。このセクションで説明するほとんどのキーバインドは、C-x C-kプレフィクスが必要ない場合があります。たとえば、すぐに C-kをタイプした場合は、マクロを再実行します。

コマンド C-x C-k C-n (kmacro-cycle-ring-next) および C-x C-k C-p (kmacro-cycle-ring-previous) は、マクロリングをローテートして、次または前のキーボードマクロをリングの先頭に移動させます。新しく先頭となったマクロの定義は、エコーエリアに表示されます。お望みのマクロが先頭にくるまで、すぐに C-nまたは C-pを繰り返しタイプすれば、マクロリングのローテートを続けることができます。新しくマクロリングの先頭にきたマクロを実行するには、単に C-kをタイプします。

Emacs はマクロリングの先頭を、最後に定義されたキーボードマクロとして扱います。たとえば、そのマクロは F4で実行でき、C-x C-k nで名前をつけることができます。

キーボードマクロリングに格納できるマクロの最大数は、カスタマイズ可能な変数 kmacro-ring-maxにより決定されます。

14.3 キーボードマクロカウンター

キーボードマクロには、それぞれカウンターが割り当てられています。これはマクロの定義を開始したとき 0 に初期化されます。このカレントカウンター *(current counter)* の数値をバッファーに挿入することもできます。カレントカウンターの数値は、マクロが呼び出された回数にもとづきます。バッファーにカウンターの値が挿入される度に通常、カウンターは増加します。

カレントカウンターに加えて、前回カレントカウンターが増加またはセットされたときにもっていた値を記録する、前回カウンター *(previous counter)* も保守します。`C-u 0 C-x C-k C-i` により増分値 0 でカレントカウンターを増加させると、カレントカウンターの値も前回カウンターの値として記録されることに注意してください。

`F3` キーボードマクロの定義では、キーボードマクロカウンターの値をバッファーに挿入します (`kmacro-start-macro-or-insert-counter`)。

`C-x C-k C-i`
キーボードマクロカウンターの値をバッファーに挿入します (`kmacro-insert-counter`)。

`C-x C-k C-c`
キーボードマクロカウンターをセットします (`kmacro-set-counter`)。

`C-x C-k C-a`
プレフィクス引数をキーボードマクロカウンターに加えます (`kmacro-add-counter`)。

`C-x C-k C-f`
挿入するキーボードマクロカウンターの書式を指定します (`kmacro-set-format`)。

キーボードマクロを定義しているとき、コマンド `F3` (`kmacro-start-macro-or-insert-counter`) は、キーボードマクロカウンターの現在の値をバッファーに挿入して、カウンターを 1 増加させます (マクロを定義していないとき、`F3`はマクロの定義を開始します。Section 14.1 [Basic Keyboard Macro], page 137 を参照してください)。異なる増分の指定には、数引数を使うことができます。単にプレフィクス `C-u`を指定すると、前回カウンターの値を挿入して、カレントカウンターの値は変化しません。

例として数字が振られたリストを構築するために、キーボードマクロカウンターを使う方法を見てみましょう。以下のキーシーケンスを考えてください:

```
F3 C-a F3 . SPC F4
```

マクロ定義の一部として、現在の行の先頭に文字列 '`0.` 'が挿入されます。バッファーの他の箇所で `F4` でマクロを呼び出すと、その行の先頭に文字列 '`1.` 'が挿入されます。その後に呼び出すと '`2.` '、'`3.` '、... が挿入されます。

コマンド `C-x C-k C-i` (`kmacro-insert-counter`) は、`F3`と同様のことを行いますが、これはキーボードマクロの定義外でも使用できます。キーボードマクロが定義中でなく実行もされていない場合、これはキーボードマクロリングの先頭にあるマクロのカウンター値を挿入および増加します。

コマンド `C-x C-k C-c` (`kmacro-set-counter`) は現在のマクロカウンターを、数引数の値にセットします。マクロ内で使用した場合、マクロ実行ごとに処理します。プレフィクス引数に単に `C-u`を指定した場合、マクロの現在の繰り返し実行おいて、カウンターが最初にもっていた値に、カウンターをリセットします (この繰り返しにおける増加を取り消します)。

コマンド `C-x C-k C-a` (`kmacro-add-counter`) は、プレフィクス引数を現在のマクロカウンターに加えます。単に `C-u`を引数に指定すると、任意のキーボードマクロにより最後に挿入された値に、カウンターをリセットします (通常これを使うときは、最後の挿入は同じマクロによる同じカウンターです)。

コマンド C-x C-k C-f (kmacro-set-format) は、マクロカウンターを挿入するときに使われる書式の入力を求めます。デフォルトの書式は '%d'で、これはパディングなしの 10 進数字が挿入されることを意味します。ミニバッファーに何も入力せずに exit することにより、このデフォルト書式にリセットできます。format関数 (この関数はさらに 1 つの整数の引数をとります) が受け入れる書式文字列を指定できます (Section "Formatting Strings" in *The Emacs Lisp Reference Manual* を参照してください)。ミニバッファーに書式文字列を入力するときは、書式文字列をダブルクォーテーションで括らないでください。

キーボードマクロの定義および実行がされていないときにこのコマンドを使うと、新しい書式はそれ以降のマクロ定義すべてに影響を及ぼします。既存のマクロは、それが定義されたときの書式を使いつづけます。キーボードマクロ定義中に書式をセットすると、そのマクロが定義されている箇所に影響を及ぼしますが、それ以降のマクロには影響を与えません。マクロの実行においては、そのマクロ定義の時点の書式が使われます。マクロの実行中にマクロ書式を変更すると、これは定義中における書式の変更と同様、それ以降のマクロに影響を与えません。

C-x C-k C-fによりセットされた書式は、レジスターに格納された数字の挿入には影響しません。

マクロの繰り返しにおいてレジスターを増加してカウンターとして使う場合、これはキーボードマクロカウンターと同じことです。Section 10.5 [Number Registers], page 74 を参照してください。大抵の用途では、キーボードマクロカウンターを使う方が単純です。

14.4 変化のあるマクロの実行

キーボードマクロで、query-replaceのように変更を行うか応答を求める効果を作ることができます。

C-x q　　マクロ実行中にこの箇所に到達すると確認を求めます (kbd-macro-query)。

マクロ定義中に問い合わせを行いたい箇所で C-x qをタイプします。マクロ定義中は C-x qは何も行いませんが、後でマクロを実行すると C-x qは実行を続けるか対話的に確認を求めます。

以下は C-x qにたいする有効な応答です:

SPC (または y)
: キーボードマクロの実行を続けます。

DEL (または n)
: マクロのこの繰り返しでの残りの部分をスキップして、次の繰り返しを開始します。

RET (または q)
: マクロのこの繰り返しでの残りの部分をスキップして、これ以上の繰り返しを取り消します。

C-r
: マクロの一部ではない編集を行うことができる、再帰編集レベル (recursive editing level) に入ります。C-M-cを使って再帰編集を抜けると、キーボードマクロを続行するか再び確認を求められます。ここで SPCをタイプすると、マクロ定義の残りの部分が実行されます。マクロの残りの部分が期待したように動作するためにポイントとテキストを残すのは、ユーザーの責任です。

C-x qにプレフィクス引数を指定した C-u C-x qは、完全に異なる関数を実行します。これはマクロ定義中およびマクロ実行中の両方で、キーボード入力を読みとる再帰編集に入ります。定義中のときは、再帰編集の中で行った編集はマクロの一部とはなりません。マクロ実行中は、再帰編集により各繰り返しにおいて特別な編集を行う機会が与えられます。Section 31.11 [Recursive Edit], page 484 を参照してください。

14.5 キーボードマクロの命名と保存

`C-x C-k n` 一番最近定義したキーボードマクロに、名前 (持続期間は Emacs セッション中) を与えます (`kmacro-name-last-macro`)。

`C-x C-k b` 一番最近定義したキーボードマクロを、キーにバインド (持続期間は Emacs セッション中) します (`kmacro-bind-to-key`)。

`M-x insert-kbd-macro`
キーボードマクロの定義を、Lisp コードとしてバッファーに挿入します。

キーボードマクロを後で使うために、`C-x C-k n` (`kmacro-name-last-macro`) を使って名前をつけて保存することができます。これはミニバッファーを使って名前を引数として読み取り、最後のキーボードマクロの現在の定義を実行するための名前を定義します (後でこのマクロの定義を追加した場合にはその名前のマクロ定義は変更されない)。マクロ名は Lisp シンボル、`M-x`で呼び出せて、`keymap-global-set`でキーにバインドできる有効な名前をつけます。キーボードマクロ以外に定義されている名前を指定すると、エラーメッセージが表示されて何も変更はされません。

`C-x C-k b` (`kmacro-bind-to-key`) の後に、バインドしたいキーシーケンスを続けることにより、最後のキーボードマクロ (の現在の定義) をキーにバインドすることもできます。グローバルキーマップ (global keymap) の任意のキーシーケンスにバインドできますが、大部分のキーシーケンスはすでに他のバインドをもっているので、キーシーケンスの選択は慎重に行う必要があります。任意のキーマップで既存のバインドをもつキーシーケンスにバインドしようとすると、既存のバインドを置き換える前に確認を求めます。

既存のバインドの上書きに起因する問題を避けるには、キーシーケンス `C-x C-k 0`から `C-x C-k 9`と、`C-x C-k A`から `C-x C-k Z`を使います。これらのキーシーケンスは、キーボードマクロのバインド用に予約されています。これらのキーシーケンスにバインドするには、キーシーケンス全体ではなく数字か文字だけをタイプすればバインドできます。たとえば、

```
C-x C-k b 4
```

これは最後のキーボードマクロをキーシーケンス `C-x C-k 4`にバインドします。

1 度マクロにコマンド名をつければ、その定義をファイルに保存できます。それは他の編集セッションでも使用できます。最初に定義を保存したいファイルを visit します。次に以下のコマンドを使います:

```
M-x insert-kbd-macro RET macroname RET
```

これは後で実行するとき、今と同じ定義のマクロとなる Lisp コードをバッファーに挿入します (これを行うために Lisp コードを理解する必要はありません。なぜならあなたのかわりに `insert-kbd-macro` が Lisp コードを記述するからです)。それからファイルを保存します。後でそのファイルを `load-file` (Section 24.8 [Lisp Libraries], page 327 を参照してください) でロードできます。init ファイル `~/.emacs` (Section 33.4 [Init File], page 528 を参照してください) に保存すれば、そのマクロは Emacs を実行する度に定義されます。

`insert-kbd-macro`にプレフィクス引数を与えると、(もしあれば)*macroname*にバインドしたキーを記録するための Lisp コードが追加されるので、ファイルをロードしたとき同じキーにマクロが割り当てられます。

14.6 キーボードマクロの編集

`C-x C-k C-e`
最後に定義されたキーボードマクロを編集します (`kmacro-edit-macro`)。

C-x C-k e *name* RET
: 以前に定義されたキーボードマクロ *name*を編集します (edit-kbd-macro)。

C-x C-k l
: 過去300回分のキーストロークを、キーボードマクロとして編集します (kmacro-edit-lossage)。

C-x C-k C-eまたは C-x C-k RET (kmacro-edit-macro) をとタイプして、最後のキーボードマクロを編集できます。これはマクロ定義をバッファーに整形出力して、それを編集するために特化したモードに入ります。そのバッファーで C-h mをタイプすると、マクロを編集する方法の詳細が表示されます。編集を終了するには C-c C-cをタイプしてください。

名前をつけたキーボードマクロ、またはキーにバインドしたマクロは、C-x C-k e (edit-kbd-macro) とタイプして編集できます。このコマンドに続けてそのマクロを呼び出すときのキー入力 (C-x e、M-x *name*、またはその他のキーシーケンス) を入力します。

C-x C-k l (kmacro-edit-lossage) とタイプして、最近300回のキーストロークをマクロとして編集できます。

14.7 キーボードマクロのステップ編集

C-x C-k SPC (kmacro-step-edit-macro) とタイプして、最後のキーボードマクロをインタラクティブに1コマンドずつ再生および編集できます。マクロを qまたは C-gで終了しなければ、編集されたマクロでマクロリングの最後のマクロを置き換えます。

このマクロ編集機能は、最初 (または次) に実行されるコマンドと、それにたいする操作を尋ねるプロンプトをミニバッファーに表示します。?を入力すれば、オプションの要約を表示できます。以下のオプションが利用可能です:

- SPCおよび yは、現在のコマンドを実行して、キーボードマクロの次のコマンドに進みます。
- n、d、および DELは、現在のコマンドをスキップして削除します。
- fは、キーボードマクロの実行において現在のコマンドをスキップしますが、マクロから削除はしません。
- TABは、現在のコマンドと、現在のコマンドのすぐ後に続く同じようなコマンドを実行します。たとえば連続する文字の挿入 (self-insert-commandに相当) には、TABが使われます。
- cは、(これ以上の編集は行わずに) キーボードマクロの最後まで実行を続けます。実行が正常に終了した場合、編集されたマクロで元のキーボードマクロを置き換えます。
- C-kは、キーボードマクロの残りの部分をスキップおよび削除して、ステップ編集を終了し、編集されたマクロで元のキーボードマクロを置き換えます。
- qおよび C-gは、キーボードマクロのステップ編集を取り消して、キーボードマクロにたいして行った編集を破棄します。
- i *key*... C-jは、一連のキーシーケンス (最後の C-jは含まれません) を読み取って実行し、キーボードマクロの現在のコマンドの前に挿入します (現在のコマンドはスキップしません)。
- I *key*...は、1つのキーシーケンスを読み取って実行し、キーボードマクロの現在のコマンドの前に挿入します (現在のコマンドはスキップしません)。
- r *key*... C-jは、一連のキーシーケンス (最後の C-jは含まれません) を読み取って実行し、現在のコマンドをそれで置き換えます (実行は挿入されたキーシーケンスの次に移ります)。
- R *key*...は、1つのキーシーケンスを読み取って実行し、キーボードマクロの現在のコマンドを、そのキーシーケンスで置き換えます (実行は挿入されたキーシーケンスの次に移ります)。

- a *key*... C-jは、現在のコマンドを実行してから、一連のキーシーケンス (最後の C-jは含まれません) を読み取って実行してから、それをキーボードマクロの現在のコマンドの後ろに挿入します (実行は現在のコマンドと、その後ろに挿入されたキーシーケンスの次に移ります)。
- A *key*... C-jは、キーボードマクロの残りのコマンドを実行してから、一連のキーシーケンス (最後の C-jは含まれません) を読み取って実行し、それらをキーボードマクロの最後に追加します。それからステップ編集を終了し、編集されたマクロで元の元のキーボードマクロを置き換えます。

15 ファイルの処理

オペレーティングシステムはファイルにデータを永続化するので、Emacs で編集するテキストの大部分はファイルから読み込んで、最終的にファイルに格納します。

ファイルを編集するには、Emacs にファイルを読み込み、ファイルのテキストを含むバッファーを準備するよう、指示しなければなりません。これを、ファイルの *visit(*訪問*)* と呼びます。編集コマンドは直接バッファーのテキスト、つまり Emacs 内部のコピーに適用されます。変更がファイルに反映されるのは、バッファーをファイルに保存 *(save)* したときだけです。

ファイルの visit や保存に加え、Emacs はファイルの削除、コピー、名前の変更、ファイルへの追加、ファイルの複数バージョンの保持、ディレクトリーの操作を行うことができます。

15.1 ファイルの名前

ファイルを操作する Emacs コマンドの多くは、ミニバッファー (Section 5.2 [Minibuffer File], page 28 を) を使って、ファイル名の指定を求めます。

ミニバッファーでは、通常の補完およびヒストリーコマンドを使うことができます (Chapter 5 [Minibuffer], page 28 を参照してください)。ファイル名の補完では、ファイル名の拡張子が変数 `completion-ignored-extensions`に含まれているファイルは無視されます (Section 5.4.5 [Completion Options], page 35 を参照してください)。またほとんどのコマンドは、ファイルの読み込みにおいて、確認をともなう寛大な補完 (permissive completion with confirmation) を使います。この補完では、存在しないファイル名が許されますが、存在しないファイル名の入力を完了するために `RET`をタイプすると、Emacs は‘`[Confirm]`’を表示し、この確認に同意するために 2 番目の `RET`をタイプしなければなりません。詳細については、Section 5.4.3 [Completion Exit], page 33 を参照してください。

ミニバッファーのヒストリーコマンドは、ファイル名を読み取るための特別な機能をいくつか提供します。Section 5.5 [Minibuffer History], page 36 を参照してください。

それぞれのバッファーは、バッファーローカルな変数 `default-directory`に、デフォルトのディレクトリーを格納しています。ミニバッファーを使ってファイル名を読み取るとき、通常 Emacs はミニバッファーの初期内容として、デフォルトディレクトリーを挿入します。変数 `insert-default-directory`を `nil`に変更することにより、この挿入を抑制できます。常に Emacs は任意の相対パスで指定されたファイル名を、デフォルトディレクトリーにたいする相対パスとみなします。たとえばディレクトリーを指定しないファイル名は、デフォルトディレクトリーのファイルを指定します。

ファイルを visit するとき、Emacs は visit しているバッファーの `default-directory`に、そのファイルのディレクトリーをセットします。`C-x b`のようなコマンドを通じて、ファイルを visit していないバッファーを新たに作成すると、通常そのバッファーのデフォルトディレクトリーは、現在のバッファーのデフォルトディレクトリーをコピーします。現在のバッファーの `default-directory`の値を見るために、`M-x pwd`コマンドを使用できます。`M-x cd`コマンドはディレクトリー名の入力を求め、バッファーの `default-directory`もそのディレクトリーをセットします (これを行うことによりバッファーのファイル名は変更されません)。

例として、ファイル`/u/rms/gnu/gnu.tasks`を visit しているとします。このときデフォルトディレクトリーは`/u/rms/gnu/`にセットされます。ファイル名を読み取るコマンドを呼び出して、ミニバッファーでディレクトリー名を省略して単に‘`foo`’と入力すると、これはファイル`/u/rms/gnu/foo`を指定したことになります。‘`../.login`’と入力すると`/u/rms/.login`、‘`new/foo`’と入力すると`/u/rms/gnu/new/foo`を指定したことになります。

ミニバッファーにファイル名をタイプするとき、2 つのショートカットを使うことができます。2 つのスラッシュは、2 番目のスラッシュの前にあるすべてを無視します。そして '~/'は、ユーザーのホームディレクトリーです。Section 5.2 [Minibuffer File], page 28 を参照してください。

文字 '$'は、ファイル名を置き換える環境変数の代用として使われます。環境変数の名前は、'$'の後ろのすべての英数字から構成されます。'$'の後ろの、大カッコ (braces) に囲まれた変数名も使用できます。たとえばシェルコマンド `export FOO=rms/hacks`は、名前が FOOの環境変数をセットするために使われます。すると`/u/$FOO/test.c`と`/u/${FOO}/test.c`はどちらも、`/u/rms/hacks/test.c`の省略形となります。環境変数が定義されていないときは、何の置き換えも発生せず、文字 '$'はそれ自身を意味します。Emacs 外部でセットされた環境変数は、それが Emacs の開始前に適用されたときだけ、Emacs に影響を与えることに注意してください。

'$'により環境変数が展開されるようなとき、名前に '$'を含むファイルにアクセスする場合は、'$$' とタイプします。1 つの '$'が環境変数を展開するのと同時に、2 つのペアは 1 つの '$'に変換されます。かわりにファイル名を '/:'でクォートすることもできます (Section 15.16 [Quoted File Names], page 170 を参照してください)。名前が文字 '~'で始まるファイル名も、'/:'でクォートするべきです。

ファイル名に非 ASCII文字を含めることができます。Section 19.11 [File Name Coding], page 230 を参照してください。

15.2 ファイルの visit(訪問)

`C-x C-f` ファイルを visit します (`find-file`)。

`C-x C-r` 変更を許さない閲覧用として、ファイルを visit します (`find-file-read-only`)。

`C-x C-v` 最後に visit したファイルとは異なるファイルを、かわりに visit します (`find-alternate-file`)。

`C-x 4 f` 別のウィンドウでファイルを visit します (`find-file-other-window`)。選択されたウィンドウに表示されているものは変更しません。

`C-x 5 f` 新しいフレームでファイルを visit します (`find-file-other-frame`)。選択されたフレームに表示されているものは変更しません。

`M-x find-file-literally`
内容を変換せずにファイルを visit します。

ファイルを *Visiting(訪問)* するとは、そのファイル内容を Emacs のバッファーに読み込むことを意味するので、それを編集することができます。Emacs は visit するファイルごとに、新しいバッファーを作成します。

ファイルを visit するには、`C-x C-f` (`find-file`) とタイプして、visit したいファイルの名前をミニバッファーで入力します。ミニバッファーでは、`C-g`をタイプして、コマンドを中止することができます。ミニバッファーでのファイル名の入力についての詳細は、Section 15.1 [File Names], page 145 を参照してください。

ファイルは存在するが、システムが読み込みを許さない場合、エコーエリアにエラーメッセージが表示されます (GNU および Unix システムでは、'su'や 'sudo'のような方法を使用して、そのようなファイルを visit できるかもしれない。Section 15.15 [Remote Files], page 169 を参照されたい。) それ以外の場合、スクリーンに新しいテキストが表示され、モードラインバッファー名が表示されることで、`C-x C-f`が成功したことを知ることができます。通常 Emacs は、ファイル名からディレクトリー名を省いて、バッファー名を作ります。たとえば`/usr/rms/emacs.tex`という名前のファイルは、バッファー名 '`emacs.tex`'となります。その名前のバッファーがすでにある場合、Emacs は一意

な名前を作ります。通常の方法はディレクトリー名にもとづく接尾辞の追加です (たとえば‘<rms>’、‘<tmp>’など) が、違う方法を選択することもできます。Section 16.7.1 [Uniquify], page 184 を参照してください。

新しいファイルを作成するには、同じコマンド C-x C-fを使って visit するだけです。Emacs はエコーエリアに‘(New file)’と表示しますが、他の点では既存の空のファイルを visit したのと同じく振る舞います。

ファイルを visit した後で編集コマンドにより行われた変更は、Emacs のバッファーに反映されます。バッファーを保存 *(save)* するまでは、visit しているファイルに影響はありません。バッファーが保存されていない変更を含むとき、そのバッファーが変更されている *(modified)* といいます。これはバッファーを保存しなければ、その変更が失われることを意味します。モードラインの左余白の近くに 2 つのアスタリスクが表示され、バッファーが変更されていることを示します。

すでに Emacs が visit しているファイルを visit した場合、C-x C-fは他のコピーを作らず既存のバッファーに切り替えます。切り替えを行う前に、最後に visit または保存した後にファイルが変更されているか確認します。もしファイルが変更されているとき、Emacs はそれの再読み込みを提案します。

large-file-warning-threshold (デフォルトは 10000000 で、これは約 10MB) より大きなファイルの visit を試みると、Emacs は最初に確認を求めます。yを応えることによりファイルの visit を続けて、lではファイルをリテラル (以下参照) に visit します。巨大なファイルではリテラルにファイルを visit することにより、種々の潜在的に高価な機能がオフに切り替えられるので、操作や編集が高速化されます。しかし Emacs は、Emacs バッファーの最大サイズ (Emacs が割り当てられるメモリー量の制限と、Emacs が扱える整数により制限されます) を超えるバッファーは、visit できないことに注意してください。この場合、Emacs は最大バッファーサイズを超えた旨を知らせるエラーメッセージを表示します。

tree-sitter パースライブラリーを使うメジャーモード (Section 20.1 [Major Modes], page 240 を参照) をもつようなファイルの visit を試みる場合に、そのファイルのサイズ (バイト) が変数treesit-max-buffer-sizeの値より大きければ Emacs が警告を表示します。デフォルト値は 64 ビットの Emacs では 40MB、32 ビットの Emacs では 15MB です。これはそのような巨大なバッファーで tree-sitter にもとづくメジャーモードをアクティブにせず、Emacs がメモリー不足に陥る危険を回避するためです。tree-sitter の典型的なパーサーは、解析するテキストのおよそ 10 倍のメモリーを必要とするからです。

ファイル名にシェル形式のワイルドカード文字が含まれている場合、Emacs はそれにマッチするすべてのファイルを visit します。(大文字小文字を区別しないファイルシステムでは、Emacs は大文字小文字に関係なくワイルドカードをマッチします)。ワイルドカードには‘?’、‘*’および‘[...]’シーケンスが含まれます。ミニバッファーでワイルドカード‘?’をファイル名に入力するには、C-q ?とタイプする必要があります。ワイルドカード文字を実際に名前に含むファイルを visit する方法についての情報は、Section 15.16 [Quoted File Names], page 170 を参照してください。find-file-wildcards をカスタマイズして、ワイルドカード機能を無効にすることができます。

あるバッファーで visit 済みだが外部から変更されたファイルを visit しようとすると、Emacs は通常はディスク上のファイルを再読み込みするかどうか尋ねます。ただし query-about-changed-fileを nilにセットすると、Emacs は問い合わせを行わずかわりに変更前のバッファー内容を表示して、ファイルからバッファーをリバートする方法をエコーエリアのメッセージで示します。

無意識に間違ったファイル名をタイプして違うファイルを visit した場合、C-x C-v (find-alternate-file) で実際に望むファイルを visit できます。C-x C-vは C-x C-fと似ていますが、これは現在のバッファーを kill します (変更されている場合は最初に保存するか確認を求めます)。C-x C-vが visit するファイルの名前を読み取るときは、ミニバッファーにデフォルトのファイ

ル名全体を挿入して、ポイントをディレクトリー名の後に置きます。これは名前を少し間違えた場合などに便利です。

実際はディレクトリーであるファイルを visit したとき、Emacs は Emacs のディレクトリーブラウザーの Dired を呼び出します。Chapter 27 [Dired], page 380 を参照してください。この振る舞いは、変数 `find-file-run-dired`を `nil`にセットすることにより無効にできます。この場合、ディレクトリーを visit するとエラーになります。

実際には他のファイルの集まりであるようなアーカイブファイル *(file archives)* の場合、アーカイブされたメンバーを操作できる、Dired に似た環境を呼び出す特別なモードで visit します。これらの機能については、Section 15.14 [File Archives], page 168 を参照してください。

オペレーティングシステムが変更を許していない、または読み取り専用にしているファイルを visit した場合、Emacs もバッファーを読み取り専用にするので、保存すると問題を起こすような変更を防ぐことができます。`C-x C-q` (`read-only-mode`) で、バッファーを書き込み可能にできます。Section 16.3 [Misc Buffer], page 179 を参照してください。

予期せぬ入力による変更を防ぐために、読み込み専用でファイルを visit したい場合は、`C-x C-f` のかわりにコマンド `C-x C-r` (`find-file-read-only`) で visit します。

`C-x 4 f` (`find-file-other-window`) は `C-x C-f`と同様ですが、指定したファイルを含むバッファーは、別のウィンドウで選択されます。`C-x 4 f`の前に選択されていたウィンドウは、すでに表示していたのと同じバッファーの表示を続けます。1 つのウィンドウしか表示されていないときにこのコマンドを使うと、これはウィンドウを 2 つに分割し、1 つのウィンドウには前に表示されていたのと同じバッファー、別の 1 つには新しい要求されたファイルを表示します。Chapter 17 [Windows], page 186 を参照してください。

`C-x 5 f` (`find-file-other-frame`) も同様ですが、新しいフレームををオープンするか、指定したファイルをすでに表示している既存のフレームを選択します。Chapter 18 [Frames], page 194 を参照してください。

グラフィカルなディスプレイでは、ファイルを visit する追加の方法が 2 つあります。1 つ目は、Emacs が適した GUI ツールキットによりビルドされているとき、マウスによるコマンドの呼び出し (メニューバーやツールバーのクリックによる) は、ミニバッファーでファイル名の入力を求める代わりに、そのツールキット標準のファイル選択 (file selection) ダイアログを表示します。GNU/Linux および Unix プラットフォームでは、GTK+、LessTif、Motif ツールキットとともに構築されていると、Emacs はこれを行います。MS-Windows と Mac では、GUI バージョンのデフォルトにより行われます。これをカスタマイズする情報については、Section 18.18 [Dialog Boxes], page 213 を参照してください。

2 つ目は、Emacs はのドラッグアンドドロップ (drag and drop) サポートで、通常の Emacs ウィンドウにファイルをドロップすることにより、そのウィンドウでファイルを visit します。例外として Dired バッファーを表示しているウィンドウにファイルをドロップすると、表示されているディレクトリーにファイルを移動またはコピーします。詳細については Section 18.14 [Drag and Drop], page 208 と Section 27.19 [Misc Dired Features], page 399 を参照してください。

テキスト端末、および GUI ツールキットなしでビルドされているグラフィカルなディスプレイでは、アイテム '`Visit New File`'と '`Open File`'をもつメニューバーの '`File`'メニューからファイルを visit できます。

文字エンコーディングと使用されている改行規則を検知するために、Emacs は自動的に内容をスキャンします。そして、それらを Emacs の内部エンコーディングとバッファーの改行規則に変換します。バッファーを保存するとき、Emacs は逆の変換を行い、元のエンコーディングと改行規則でファイルをディスクに書き込みます。Section 19.5 [Coding Systems], page 223 を参照してください。

ファイルにたいして特別なエンコーディングや変換を行わずに、非 ASCII文字のシーケンスとして編集したいときは、`M-x find-file-literally`コマンドを使います。これは `C-x C-f`と同様ですが、フォーマット変換 (Section “Format Conversion” in *the Emacs Lisp Reference Manual* を参照してください)、文字コード変換 (Section 19.5 [Coding Systems], page 223 を参照してください)、自動解凍 (Section 15.13 [Compressed Files], page 168 を参照してください) を行わず、`require-final-newline`による最後の改行も追加しません (Section 15.3.3 [Customize Save], page 154 を参照してください)。同じファイルをすでに通常 (非リテラル) の方法で visit している場合、このコマンドはそれをリテラル (そのままの文字の列) として visit するかを尋ねます。

ファイルが別のファイルに (緩やかに) 結びるけられているときがあります。このようなファイルのことを兄弟ファイル *(sibling files)* と呼ぶことにしましょう。たとえば C ファイルの編集する際に‘`"foo.c"`’というファイルがある場合には‘`"foo.h"`’というファイルもある場合が間々あり、そのようなファイルが兄弟ファイルに該当します。あるいは‘`"src/emacs/emacs-27/lisp/allout.el"`’と‘`"src/emacs/emacs-28/lisp/allout.el"`’という異なるバージョンのファイルについても兄弟とみなすことができるかもしれません。これらの兄弟ファイル間をジャンプするためのコマンド`find-sibling-file`を Emacs は提供しますが、ユーザーがどのファイルを兄弟とみなしたいかを推奨するのは不可能なので、ユーザーオプション `find-sibling-rules`を変更することによってルールを設定できるようにしています。このルールはマッチと展開形のペアーからなる要素です。

たとえば‘`".c"`’を‘`".h"`’にマッピングするには、以下のように記述します:

```
(setq find-sibling-rules
      '(("\\([^/]+\\)\\.c\\'" "\\1.h")))
```

(`ff-find-related-file`は特に C ファイル向けに同等の機能を提供している。Section 23.12.4 [Other C Commands], page 306 を参照のこと)

あるいは‘`"src/emacs/DIR/file-name"`’配下にある別の *dir*にあるすべてのファイルを兄弟としたければ、以下のように記述できます:

```
(setq find-sibling-rules
      '(("src/emacs/[^/]+/\\(.*\\)\\'" "src/emacs/.*/\\1")))
```

見てのとおり、これは *(MATCH EXPANSION...)*という要素のリストです。*match*は visit するファイル名にマッチする正規表現、そして *expansion*はそれぞれ‘`\\1`’、... を使用して参照されるマッチグループです。結果として得られる展開文字列はその後ファイルシステムに適用されて、この展開文字列 (regexp として解釈) とマッチするファイルがあるかどうか確認されます。

2 つの特別なフック変数により、ファイルを visit する操作を変更して拡張することができます。存在しないファイルを visit することにより、`find-file-not-found-functions`の関数が実行されます。この変数は関数のリストを保有し、それらはどれか 1 つが非 `nil`を返すまで、(引数を指定せずに)1 つずつ呼び出されます。これはノーマルフックではないため、その事実を示すために名前の最後が‘`-hook`’ではなく、‘`-functions`’で終わっています。

ファイルが存在するしないにかかわらず、ファイルを visit するのに成功すると、引数なしで関数`find-file-hook`を呼び出します。この変数はノーマルフックです。ファイルが存在しない場合、最初に `find-file-not-found-functions`を実行します。Section 33.2.2 [Hooks], page 509 を参照してください。

ファイルを編集するために自動的にメジャーモードを指定し (Section 20.3 [Choosing Modes], page 244 を参照してください)、そのファイルのために特別なローカル変数を定義する方法がいくつかあります (Section 33.2.4 [File Variables], page 512 を参照してください)。

15.3 ファイルの保存

Emacs でのバッファーの保存 *(Saving)* は、バッファーの内容を、そのバッファーにより visit されているファイルに書き戻すことを意味します。

15.3.1 ファイルを保存するコマンド

ファイルの保存と書き込みに関するコマンドが、いくつかあります。

`C-x C-s` 現在のバッファーを、そのファイルに保存します (`save-buffer`)。

`C-x s` 任意、またはすべてのバッファーを、それらのファイルに保存します (`save-some-buffers`)。

`M-~` 現在のバッファーが変更されたことを忘れます (`not-modified`)。プレフィクス引数 (`C-u`) を指定すると、現在のバッファーを変更済みとマークします。

`C-x C-w` 現在のバッファーを、指定したファイル名で保存します (`write-file`)。

`M-x set-visited-file-name`
現在のバッファーが保存される場所で、ファイル名を変更します。

`M-x rename-visited-file`
`M-x set-visited-file-name`と同様ですが、(もしあれば) そのバッファーが visit しているファイルのリネームも行います。

ファイルを保存して変更を永続化させたいときは、`C-x C-s` (`save-buffer`) とタイプします。保存が完了すると、`C-x C-s`は以下のようなメッセージを表示します:

```
Wrote /u/rms/gnu/gnu.tasks
```

現在のバッファーが変更されていない (新規作成されたとき、または最後に変更されたときから変更していない) 場合、保存しても意味がないので実際の保存は行われません。かわりに `C-x C-s`は、エコーエリアに以下のようなメッセージを表示します:

```
(No changes need to be saved)
```

`C-u C-x C-s`のようにプレフィクス引数を指定すると、Emacs はそのバッファーを次回の保存が行われるときバックアップするようマークします。Section 15.3.2 [Backup], page 152 を参照してください。

コマンド `C-x s` (`save-some-buffers`) は、任意、またはすべての変更されたバッファーの保存を提案します。これはバッファーごとに何を行うか尋ねます。使用できる応答は、`query-replace`と同様です。

`y`
`SPC` このバッファーを保存し、残りのバッファーについて尋ねます。

`n`
`DEL` このバッファーは保存せずに、残りのバッファーについて尋ねます。

`!` このバッファーを保存し、残りのバッファーを尋ねることなくすべて保存します。

`q`
`RET` これ以上の保存をせずに、`save-some-buffers`を終了します。

`.` このバッファーを保存したら、他のバッファーをどうするか尋ねることなく、`save-some-buffers`を終了します。

`C-r` 現在尋ねられているバッファーを閲覧します。View モードから抜けると、再び `save-some-buffers`はどうするか尋ねます。

`C-f` `save-some-buffers`を exit して、カレントで問い合わせ中だったバッファーを visit します。

`d` そのバッファーに対応するファイルと Diff をとり、どのような変更を保存するのか確認できます。これはコマンド `diff-buffer-with-file` (Section 15.9 [Comparing Files], page 163 を参照してください) を呼び出します。

`C-h` これらのオプションについての、ヘルプメッセージを表示します。

`save-some-buffers-default-predicate`の値をカスタマイズして、どのバッファーにたいして Emacs が問い合わせるか制御できます。

Emacs を終了するキーシーケンス `C-x C-c`は、`save-some-buffers`を呼び出すので、同じ質問をします。

バッファーを変更したが変更を保存したくないとき、保存されるのを防ぐためにできることがいくつかあります (それ以外の `C-x s`または `C-x C-c`で間違って保存してしまうのはあなたの責任です)。まずできることは、`M-~` (`not-modified`) とタイプすることで、これはバッファーが変更されているというマークをクリアーします。これを行うと、保存コマンドに、バッファーが保存を必要しないと信じこませることができます (‘~’は数学のシンボルで、“not(否定)” として使われることがあります。したがって `M-~`はメタと “not” になります。)。かわりにファイルからテキストを読み込むことにより、ファイルを visit または保存された後に行った、すべての変更を取り消すことができます。これはリバート *(reverting:* 復元*)* と呼ばれます。Section 15.4 [Reverting], page 157 を参照してください (undo コマンド `C-x u`を繰り返すことにより、変更がすべての変更を取り消すこともできますが、リバートの方が簡単です)。

`M-x set-visited-file-name`は、現在のバッファーが visit しているファイルの名前を変更します。このコマンドはミニバッファーを使って、新しいファイル名を読み取ります。その後に、そのバッファーがそのファイル名のファイルを visit しているとマークし、バッファー名も合わせて変更します。`set-visited-file-name`は、新しく visit するファイルへの保存はしません。これは後で保存するときのために、Emacs 内のレコードを変更するだけです。これはバッファーを変更されている (modified) とマークするので、そのバッファーでの将来の `C-x C-s`で、保存が行われます。

バッファーにたいして、違うファイルを visit しているとマークしてすぐに保存したい場合は、`C-x C-w` (`write-file`) を使います。このコマンドは、`set-visited-file-name`の後すぐに `C-x C-s`をするのと同じですが、`C-x C-w`はファイルが存在するとき確認を求める点が異なります。ファイルを visit していないバッファーでの `C-x C-s`は、`C-x C-w`と同じ効果をもちます。したがってファイル名を読み取り、バッファーがそのファイルを visit しているとマークした後、バッファー内容をそのファイルに保存します。ファイルを visit していないバッファーのデフォルトファイル名は、バッファー名と、バッファーのデフォルトディレクトリーから合成されます (Section 15.1 [File Names], page 145 を参照してください)。

新しいファイル名が何らかのメジャーモードに関連する場合、大抵は `C-x C-w`によりそのメジャーモードへの切り替えが行われます。コマンド `set-visited-file-name`もこれを行います。Section 20.3 [Choosing Modes], page 244 を参照してください。

Emacs がファイルを保存する際、ディスクの最新バージョンのファイル日付を確認して、それが Emacs が最後に読み込み、または書き込みしたときと異なる場合、Emacs はその事実をユーザーに知らせます。なぜならそれはおそらく同時編集による問題を示しており、それをすぐにユーザーに知らせる必要があるからです。Section 15.3.4 [Simultaneous Editing], page 155 を参照してください。

15.3.2 バックアップファイル

ほとんどのオペレーティングシステムでは、ファイルを書き換えるとファイルに入っていたそれまでの記録は、自動的に破棄されます。したがって、Emacs でファイルを保存すると、ファイルの古い内容は捨てられます。しかし実際に保存する前に、Emacs が慎重に古い内容をバックアップ *(backup)* ファイルと呼ばれる別のファイルにコピーすれば、古い内容は失われません。

Emacs は最初にバッファーからファイルに保存されたときだけ、バックアップファイルを作成します。その後ファイルを何回保存しようと、バックアップは変更されません。しかしバッファーを kill してから、そのファイルを再び visit すると、新しいバックアップファイルが作成されます。

ほとんどのファイルにたいして、変数 `make-backup-files`はバックアップファイルを作るかを決定します。ほとんどのオペレーティングシステムでは、デフォルト値は `t`なので、Emacs はバックアップファイルを書き込みます。

バージョンコントロールシステム (version control system: Section 25.1 [Version Control], page 332 を参照してください) で管理されているファイルにたいして、バックアップファイルを作るかは、変数 `vc-make-backup-files`により決定されます。バージョンコントロールシステムに以前のバージョンがある場合、バックアップファイルを作るのは不必要なので、デフォルト値は `nil`です。Section “General VC Options” in *Specialized Emacs Features* を参照してください。

選択できるオプションは、Emacs にファイルごとに 1 つのバックアップを作らせる方法と、編集するファイルごとに番号がついた、一連のファイルを作る方法があります。Section 15.3.2.1 [Backup Names], page 152 を参照してください。

変数 `backup-enable-predicate`のデフォルト値は、一時的なファイルのために使われるディレクトリー (変数 `temporary-file-directory`または `small-temporary-file-directory`で指定されます) のファイルは、バックアップしないような値になっています。

前に保存されたバッファーにたいしても、バッファーから他のバックアップファイルを作るよう、Emacs に明示的に指示することができます。バッファーを `C-u C-x C-s`で保存すると、この保存したバージョンが、次にバックアップするときのバックアップになります。`C-u C-u C-x C-s`はバッファーを保存しますが、最初に元のファイル内容を新しいバックアップファイルとします。`C-u C-u C-u C-x C-s`は両方を行います。まず前の内容でバックアップを作成し、次回に保存したときは、今回保存したものをバックアップにします。

変数 `backup-directory-alist`をカスタマイズして、指定したパターンにマッチする特定のファイルにたいして、指定したディレクトリーにバックアップを作成させることができます。典型的な使い方は、要素 (`"." . dir`)を追加することにより、すべてのバックアップを絶対パス *dir*に作る方法です。異なるディレクトリーにある同じ名前のファイルによる、バックアップファイルの名前の衝突を避けるため、Emacs はバックアップファイルの名前を変更します。(`"." . ".~"`)を追加すると、これは元のファイルがあるディレクトリーに、非表示の`.~`というディレクトリーを作って、そこにバックアップを作成します。Emacs はバックアップを作るため、必要ならディレクトリーを作成します。

15.3.2.1 単一または番号つきバックアップ

Emacs のバックアップファイル作成では、バックアップの名前は通常、編集されるファイル名の後ろに ‘`~`’をつけて作成されます。したがって `eval.c`のバックアップファイルは、`eval.c~`になります。

アクセスコントロールにより Emacs が通常の名前でバックアップファイルを書き込めない場合、`~/.emacs.d/%backup%~`というバックアップファイルに書き込みます。この 1 つのファイルしか存在しないので、一番最近作られたバックアップだけが利用可能です。

Emacs は番号つきバックアップファイル *(numbered backup files)* を作ることもできます。番号つきバックアップファイルの名前は、元のファイル名の後ろに ‘`.~`’と番号と ‘`~`’をつけたものです。し

たがって eval.cのバックアップファイルは、eval.c.~1~、eval.c.~2~、...、eval.c.~259~、... となります。

変数 version-controlは、単一のバックアップファイルを作るか、複数の番号つきバックアップファイルを作るかを決定します。有効な値は以下のとおりです:

`nil` すでに番号つきバックアップのあるファイルにたいしては、番号つきバックアップを作ります。それ以外は単独のバックアップをつくります。これがデフォルトです。

`t` 番号つきバックアップを作ります。

`never` 番号つきバックアップをつくらず、常に単一のバックアップを作ります。

この変数をセットする通常の方法は、init ファイルや customization バッファーを通じて、グローバルにセットする方法です。しかし特定のバッファーにローカルに version-controlをセットして、そのバッファーのバックアップ作成を制御することができます (Section 33.2.3 [Locals], page 511 を参照)。Rmail モードのようないくつかのモードは、この変数をセットします。特定のファイルを visit するとき、常に Emacs に version-controlをローカルにセットさせることもできます (Section 33.2.4 [File Variables], page 512 を参照)。

さまざまな GNU ユーティリティーにたいして、何をすべきか指示する環境変数 VERSION_CONTROL をセットすると、Emacs も開始時にこの環境変数にしたがって、Lisp 変数 version-controlをセットします。環境変数の値が 't'または 'numbered'のときは、version-controlは tになります。値が 'nil'または 'existing'のときは、version-controlは nilになります。もし 'never'または 'simple'のときは、version-controlは neverになります。

変数 make-backup-file-name-functionに適切な Lisp 関数をセットすることにより、Emacs がバックアップファイル名を作る通常の方法をオーバーライドできます。

15.3.2.2 バックアップの自動削除

ディスク容量の過度な消費を防ぐため、Emacs は自動的に番号つきバックアップを削除することができます。一般的に Emacs は一番古いバックアップと、一番新しいバックアップをいくつか保持し、その間にあるバックアップを削除します。これは新しいバックアップが作られる度に行なわれます。

2 つの変数 kept-old-versionsおよび kept-new-versionsが、この削除を制御します。これらの変数の値は順に、削除せずに残す一番古い番号 (小さい番号) のバックアップと、一番新しい番号 (大きい番号) で、新しいバックアップが作られる度に評価されます。中間のバックアップ (一番古いものと一番新しいものを除いたもの) は、余分なバージョンで、これらのバックアップは削除されます。これらの変数の値は余分なバージョンを削除するとき、つまり新しいバックアップが作られた直後に使われます。新しく作られたバックアップは、kept-new-versionsのカウントに含まれます。デフォルトでは、両方の変数の値は 2 です。

delete-old-versionsが tのとき、Emacs は何も尋ねずに余分なバックアップファイルを削除します。nil(デフォルト) のとき、Emacs は余分なバージョンのバックアップを削除するか尋ねます。他の値の場合、Emacs はバックアップの自動削除をしません。

Dired の .(ピリオド) コマンドでも、古いバージョンを削除できます。Section 27.4 [Flagging Many Files], page 383 を参照してください。

15.3.2.3 コピー vs. リネーム

バックアップファイルは、古いファイルをコピーまたはリネームすることで作ることができます。コピーとリネームは、古いファイルが複数の名前をもつ場合 (ハードリンクされている場合) に、異なる効果をもちます。古いファイルがバックアップファイルにリネームされた場合、ハードリンクされた別

の名前で参照されるファイルは、バックアップファイルとなります。かわりに古いファイルをコピーすると、ハードリンクされた別の名前で参照されるファイルは、編集中のファイルのままで、それらの名前でアクセスする内容は新しい内容となります。

バックアップファイルを作る方法は、編集中のファイルの所有者とグループにも影響します。コピーが使われた場合、それらは変化しません。リネームが使われた場合、そのユーザーがファイルの所有者となり、ファイルのグループはデフォルト (オペレーティングシステムごとにグループのデフォルトは異なります) のグループになります。

リネームとコピーの選択は、以下の変数により行われます:

- 変数 `backup-by-copying`が非 `nil`(デフォルトは `nil`) の場合、コピーが使用されます。
- 上記以外の場合、変数 `backup-by-copying-when-linked`が非 `nil`(デフォルトは `nil`) で、ファイルが複数の名前をもつ場合は、コピーが使用されます。
- 上記以外の場合、変数 `backup-by-copying-when-mismatch`が非 `nil`(デフォルトは `t`) で、リネームによりファイルの所有者かグループが変更される場合は、コピーが使用されます。

  `backup-by-copying-when-mismatch`を `nil`に変更すると、Emacs はファイルの所有者のユーザー ID とファイルのグループのグループ ID の数字をチェックします。もしいずれかの数字が `backup-by-copying-when-privileged-mismatch`の値より大きければ、`backup-by-copying-when-mismatch`が非 `nil`であるかのように振る舞います。
- 上記以外の場合、リネームがデフォルトの選択となります。

ファイルがバージョンコントロールシステムで管理されている場合 (Section 25.1 [Version Control], page 332 を参照)、通常、Emacs は普通の方法でそのファイルのバックアップを作りません。しかしコミット (チェックインとも呼ばれる。Section 25.1.1.3 [VCS Concepts], page 334 を参照されたい) は、バックアップを作るのと似たところがあります。これらの操作は通常ハードリンクをこわし、同じファイルにたいする別のファイル名での visit を切断します。Emacs にできることはありません。バージョンコントロールシステムがこれを行うのです。

15.3.3 ファイル保存のカスタマイズ

変数 `require-final-newline`の値が `t`のとき、ファイルの保存または書き込みにより、ファイルの終端に改行がないときは、何も尋ねずに改行を追加します。値が `visit`の場合、Emacs はファイルを visit した直後に、終端に改行がないファイルの改行を追加します (これによりバッファーは変更されたとマークされます。undo はできません)。値が `visit-save`の場合、Emacs はそのような改行を visit と保存のときに追加します。値が `nil`の場合、Emacs はファイルの終端を変更しません。それ以外の非 `nil`値は、改行を追加するかを Emacs が尋ねることを意味します。デフォルトは `nil`です。

ファイルの終端に常に改行があると想定する、特定の種類のファイルのためにデザインされたメジャーモードがいくつかあります。そのようなメジャーモードは、変数 `require-final-newline`に、変数 `mode-require-final-newline`の値 (デフォルトは `t`) をセットします。後者の変数の値をセットすることにより、これらのモードが終端の改行を取り扱う方法を制御できます。

このオプションが非 `nil`でシンボリックリンクを介してファイルを visit すると、バッファー保持時に lEmacs はシンボリックリンクを解除して、`file-precious-flag`の値が非 `nil`なら、そのシンボリックリンクと同じ名前のファイルにバッファーを書き込みます (Section “Saving Buffers” in *The Emacs Lisp Reference Manual* を参照)。シンボリックリンクが指すファイルにバッファーを保存させる (リンクを維持させる) には、変数 `file-preserve-symlinks-on-save`を `t`にカスタマイズしてください。

通常プログラムがファイルに書き込むとき、オペレーティングシステムはデータをディスクにコミットする前に、ファイルのデータをメインメモリーにキャッシュします。これにより大幅にパフォー

マンスを向上できます。たとえばラップトップを使っている場合、ファイルを書き込む度にディスクをスピンアップ (spin-up) しなくて済みます。しかし、キャッシュをディスクにコミットする前にオペレーティングシステムがクラッシュすることにより、データを失うリスクもあります。

このリスクを減少させるため、Emacs はファイルを保存した後に `fsync`システムコールを呼び出すことができます。`fsync`により、データを失うリスクを皆無にすることはできません。その理由の一部は、多くのシステムは `fsync`を正しく実装していないことであり、他の理由の一部は Emacs のファイル保存手段は通常ディレクトリー更新に頼っており、これは `fsync`が正しく実装されていても、クラッシュを生き延びることはできないでしょう。

`write-region-inhibit-fsync`変数は、ファイルを保存した後に Emacs が `fsync`を呼び出すかを制御します。この変数のデフォルト値は、Emacs を対話的に使用しているときは `nil`、バッチモードの時は `t`です (Section C.2 [Initial Options], page 576 を参照)。

Emacs は自動保存ファイルの書き込みに `fsync`を使うことはありません。なぜなら、それらのファイルのデータは、いずれにせよ失われるものだからです。

15.3.4 同時編集からの保護

同時編集 (Simultaneous editing) は、2 人のユーザーが同じファイルを visit して、両者が変更と保存を行ったときに発生します。これが発生していることを誰も知らせなければ、最初に保存したユーザーは、後で自分の変更が失われていることに気付くでしょう。

いくつかのシステムでは、Emacs は 2 番目のユーザーがファイルの変更を開始すると、すぐに警告を発します。また、すべてのシステムにおいて、Emacs はファイルを保存するときにチェックして、他のユーザーの変更を上書きすることを警告します。ファイルを保存するかわりに適切な訂正アクションをとることにより、他のユーザーの変更を失わなわずに済みます。

ファイルを visit している Emacs バッファーで最初の変更を行うとき、Emacs はファイルがロック *(locked)* されていることを記録します (これは同じディレクトリーにある、特別な内容の、特別な名前のシンボリックリンク [1] を作ることにより行われます。詳細は Section “File Locks” in `elisp` を参照のこと)。変更を保存したとき、Emacs はロックを解除します。このアイデアは、ファイルを visit している Emacs バッファーに保存されていない変更があるとき、ファイルはロックされているとするものです。

変数 `create-lockfiles`を `nil`にセットすることにより、ロックファイルの作成を抑制することができます。警告**:** これにより、この機能が提供する利点を失うことになります。`lock-file-name-transforms`変数を使用すればロックファイルの書き込み場所の制御もできます。

他のユーザーによりロックされているファイルを visit しているバッファーの変更を開始すると、衝突 *(collision)* が起こります。Emacs が衝突を検知すると、Lisp 関数 `ask-user-about-lock`を呼び出して、何を行うか尋ねます。カスタマイズのためにこの関数を再定義できます。この関数の標準定義は、ユーザーに質問をして、3 つの有効な応えを受け取ります。

`s` ロックを横取りします。すでにファイルを変更したユーザーはロックを失い、あなたがロックを取得します。

`p` 続行します。他のユーザーがロックしている如何にかかわらずファイルの編集を続けます。

`q` 終了します。これはエラー (`file-locked`) を引き起こし、バッファーの内容は変更されません。あなたが試みた修正は実際には行われません。

[1] システムがシンボリックリンクをサポートしていなければ、通常のファイルが使われます。

Emacs またはオペレーティングシステムがクラッシュすると、偽のロックファイルが残ることがあり、このような偽のロックファイルによる警告を受けることがあります。偽の衝突だと確信できるときは、Emacs にとにかく実行させる `p`を使います。

ロックはファイル名にもとづいて機能するので、ファイルが複数の名前を持っていて、2 人のユーザーがそれぞれ別のファイル名で同時編集を行うことを、Emacs が防ぐことはできないことに注意してください。

ロックファイルに書き込みできない状況がいくつかあります。たとえばシステム権限不足や他の理由により、Emacs がロックファイルが作成できない場合です。このような場合でも保存を試みたときに、ファイルの最終変更日時をチェックすることにより、Emacs は衝突を検知できます。最後に Emacs が visit または保存したときからファイルが変更されているとき、それは他の何らかの手段によりファイルが変更されたことを示し、Emacs が保存を行うことによりそれらが失われることを意味します。そのようなとき Emacs は警告メッセージを表示して、保存する前に確認を求めます。保存するときは `yes`、保存を取り消すときは `no`または `C-g`と応えてください。

すでに同時編集が発生しているとき、バッファーとファイルを比較する方法の 1 つは、`M-x diff-buffer-with-file`コマンドです。Section 15.9 [Comparing Files], page 163 を参照してください。

変数 `remote-file-name-inhibit-locks`を `t`にセットすることによって、リモートロックファイルの作成を抑制することができます。警告**:** これにより、この機能が提供する利点を失うことになります。

インタラクティブに呼び出されるマイナーモード `lock-file-mode`は、カレントバッファーでの `create-lockfiles`のローカル値を切り替えます。

15.3.5 ファイルのシャドーイング

特定のファイルと等しい *shadow* コピーを 1 つ以上の場所、ことによると異なるマシン間で保持するように計画できます。これを行うにはまず、*shadow* ファイルグループをセットアップしなければなりません。これはリストにあるサイト間で共有される同じ名前のファイルのセットです。ファイルグループは永続的で、将来の Emacs セッションでも、現在のセッションと同様に適用されます。一度グループをセットアップすると、Emacs を終了する度に、編集したファイルをグループの他のファイルにコピーします。`M-x shadow-copy-files` をタイプすることにより、Emacs を終了せずにコピーすることもできます。

shadow クラスターは、ディレクトリーを共有するホストのグループなので、それらのコピーは、そのディレクトリーにあるすべてのファイルを更新するのに充分です。shadow クラスターはそれぞれ名前を持ち、プライマリーホスト (コピーを行うホスト) のネットワークアドレスと、プライマリーホスト以外でクラスターに含めるホストを選択するための正規表現を指定します。`M-x shadow-define-cluster` により shadow クラスターを定義できます。

`M-x shadow-initialize`
: ファイルのシャドーイング (shadowing) をセットアップします。

`M-x shadow-define-literal-group`
: サイト間で共有される単一のファイルを定義します。

`M-x shadow-define-regexp-group`
: ファイルのグループがマッチするすべてのファイルを、ホスト間で共有するようにします。

`M-x shadow-define-cluster RET` *name* `RET`
: shadow ファイルのクラスター *name*を定義します。

`M-x shadow-copy-files`
すべての保留中の shadow ファイルをコピーします。

`M-x shadow-cancel`
ファイルにたいする shadow 指示を取り消します。

shadow ファイルグループをセットアップするには、`M-x shadow-define-literal-group` または `M-x shadow-define-regexp-group` を使います。詳細な情報は、これらの関数のドキュメント文字列を参照してください。

ファイルを shadow にコピーする前に、Emacs は確認を求めます。“no” を応えることにより、その時だけはコピーを回避できます。特定のファイルにたいして今後も shadowing を取り消したい場合、`M-x shadow-cancel` を使うことにより、shadow ファイルグループを削除または変更します。

MS Widows では、ファイルのシャドーイングは利用できません。

15.3.6 タイムスタンプの自動更新

ファイルにタイムスタンプを書き込むことができます。これによりファイルを編集・保存する度に、タイムスタンプが自動的に更新されます。タイムスタンプは、ファイルの最初の 8 行になければならず、以下のような形式、

```
Time-stamp: <>
```

または以下のような形式です:

```
Time-stamp: " "
```

その後、フック `before-save-hook`に関数 `time-stamp`を追加します (Section 33.2.2 [Hooks], page 509 を参照してください)。ファイルを保存するとき、この関数は現在の日時で自動的にタイムスタンプを更新します。コマンド `M-x time-stamp`を使って、手動でタイムスタンプを更新することもできます。デフォルトではタイムスタンプの書式は、locale のセッティング (Section C.4 [Environment], page 579 を参照してください) と、タイムゾーン (Section “Time of Day” in *The Emacs Lisp Reference Manual* を参照してください) にしたがいます。カスタマイズに関しては、Custom グループの `time-stamp`を参照してください。

15.4 バッファーのリバート

ファイルを visit しているバッファーにたいして、広範な変更をした後に気が変わったときは、リバート *(revert:* 復元*)* することにより。変更をファイルの保存されたバージョンに戻すことができます。これを行うには `C-x x g`とタイプします。間違えてリバートしてしまうことによって大量の作業結果を失うこともあり得るので、バッファーが変更されていると Emacs は最初に確認を求めます。

`revert-buffer`コマンドは、ファイルが少ししか変更されていないときは、前にポイントがあったテキスト部分とだいたい同じ位置にポイントを置くよう試みます。しかし広範な変更を行っていた場合、ポイントは大きく異なる場所に置かれることになります。

リバートはバッファーを not modified(変更されていない) とマークします。しかし、これはリバートされた変更を単一の変更として、そのバッファーの undo ヒストリーに追加します (Section 13.1 [Undo], page 131 を参照)。したがって、リバート後に気が変わってリバートされた変更を元に戻したくなった場合は、`C-/`またはそのエイリアスをタイプして、それを行うことができます。

より保守的にバッファーをリバートするためには、コマンド `revert-buffer-with-fine-grain` を使用できます。このコマンドは `revert-buffer`と同様に機能しますがバッファー内のすべてのマーカー、プロパティ、オーバーレイを保存するよう努めることにより、可能なかぎり非破壊的なリバートを試みます。この方法によるリバートは多数回の変更を行っている際には非常に低速になり得るので、この

方法によるバッファーコンテンツ置き換えに要する最大秒数を指定するために変数revert-buffer-with-fine-grain-max-secondsを変更できます。これはrevert-buffer-with-fine-grainの実行全体がこれより長くならないことを保証するものではないことに注意してください。

ファイルに関連付けられていない、Diredバッファーのようなバッファーでも、リバートすることができます。それらの場合、リバートはその内容を再計算することを意味します。C-x bで明示的に作成したバッファーは、リバートできません。リバートを試みるとrevert-bufferはエラーを報告します。

自動的かつ頻繁に変更されるファイル、たとえば実行を続けるプロセスのログ出力などを編集しているとき、Emacsが確認なしにリバートできたら便利でしょう。このような振る舞いをさせるには、変数revert-without-queryに正規表現のリストをセットします。ファイル名がそれらの正規表現の1つにマッチしたとき、find-fileおよびrevert-bufferは、バッファーが変更されていないときは、ファイルが変更される度に自動的にリバートします (もしテキストを編集していた場合、変更を放棄するのはおそらく正しくありません)。

キーストローク C-x x gはコマンド revert-buffer-quickにバインドされています。revert-bufferと異なるのは、カレントバッファーがファイルをvisitしていて、そのバッファーが未変更なら問い合わせを行わないことです。ユーザーオプションrevert-buffer-quick-short-answersにもしたがいます。このオプションが非nilなら、長いyes/noより短いy/nで問い合わせを行います。

visit中のファイルがディスク上で変更された際にバッファーを自動的にリバートするようにEmacsに指定できます。Section 15.5 [Auto Revert], page 158 を参照してください。

15.5 自動リバートーバッファーを自動的に最新に保つ

visit中のファイルを別プログラムが変更した場合には、バッファーがvisit中のファイルと同期しなくなる可能性があります。これを最新に保つために、M-x auto-revert-modeをタイプしてAuto Revertモードを有効にできます。これはvisit中のファイルがディスク上で変更された際に、バッファーを自動的にリバートします。すべてのファイルバッファーで同じことを行うには、M-x global-auto-revert-modeをタイプしてGlobal Auto Revertモードを有効にします。

Auto Revertはバッファーに未保存の変更があったり、ディスク上のファイルが削除やリネームされた場合にはバッファーをリバートしません。

Auto Revertモードの1つの使い方は、システムログのようなファイルを “tail” することです。これにより、それらのファイルにたいする、他のプログラムによる変更を、継続的に表示できます。これを行うには、ポイントをバッファーの最後に移動します。そうすればファイル内容が変更されても、ポイントはその位置に留まります。しかし、ファイルがファイルの終端方向に向かって変更されるだけだと確信できるときは、かわりにAuto Revert Tailモード (auto-revert-tail-mode) を使います。このモードは、これをより効果的に行います。Auto Revert Tailモードは、リモートのファイルにたいしても機能します。

バッファーが自動リバートされたとき、メッセージが生成されます。これはauto-revert-verboseを非nilにセットすることにより、抑制できます。

Auto Revertモードはリモートファイルのチェックやリバートは、通常は低速なので行いません。この挙動は変数auto-revert-remote-filesを非nilにセットして変更できます。

デフォルトでは、Auto Revertモードはファイル通知 *(file notifications)* を使用して機能します。これにより、ファイルシステムへの変更が、OSからEmacsに報告されます。変数auto-revert-use-notifyをnilにカスタマイズして、ファイル通知を無効にできます。その場合、Emacsは5秒

ごとにポーリングして、ファイルの変更をチェックするでしょう。変数 auto-revert-intervalを通じて、ポーリングの間隔を変更できます。

すべてのシステムでファイル通知がサポートされている訳ではありません。サポートされないシステムでは、auto-revert-use-notifyはデフォルトで nilです。

デフォルトでは Auto Revert モードは、たとえファイル通知の使用中でもファイル変更の定期的なポーリングは行いません。多くの場合にはポーリングは不要であり、ポーリングをオフにして通知だけを信頼することにより電力が節減できます。これを行うには変数 auto-revert-avoid-polling に非 nilをセットします。しかし別のマシンからファイルを変更可能な Unix 系マシン上の主要なネットワークファイルシステムのように、特定のファイルシステムでは通知は効果がありません。そのようなファイルシステムではポーリングが必要かもしれません。auto-revert-avoid-pollingが非 nilのときにポーリングを強制するには、を通知を使用するファイルから除外すべきファイルにマッチするように auto-revert-notify-exclude-dir-regexpをセットします。

Dired バッファー (Chapter 27 [Dired], page 380 を参照) では、Auto Revert モードはそのバッファーのディレクトリーでファイルが作成、または削除されたとき更新を行います。

バージョンコントロールシステムの下にあるファイルを、以前のバージョンにリバーとするコマンドについては、Section 25.1.8 [VC Undo], page 345 を参照してください。バージョンコントロールシステムの下にあるファイルを visit しているときの自動リバートの特性については、Section 25.1.2 [VC Mode Line], page 336 を参照してください。

15.6 自動保存－災害にたいする防御

Emacs は定期的に、visit しているファイルを、実際に使っているファイルを変更せずに、別のファイルに自動的に保存するときがあります。これは自動保存 *(auto-saving)* と呼ばれます。これはシステムがクラッシュしたとき、失われてしまう作業をある程度以下に制限するためのものです。

Emacs が自動保存するときだと決定すると、各バッファーを判断して、それらのバッファーの自動保存が有効で、最後に自動保存されてから変更されている場合には自動保存します。auto-save-no-message変数が nil (デフォルト) にセットされている際には、もし自動保存されているファイルが実際にあればエコーエリアに ‘Auto-saving...’というメッセージが自動保存の間に表示されます。このメッセージを無効にするには、この変数を非 nilにカスタマイズしてください。自動保存の間のエラーはキャッチされるので、ユーザーがタイプして実行したコマンドに、干渉することはありません。

15.6.1 自動保存ファイル

自動保存は通常、visit しているファイルへの保存はしません。なぜなら永続化したくない変更を保存するのは、好ましくないからです。そのかわりに、自動保存は *auto-save* ファイルと呼ばれる別のファイルに保存し、visit しているファイルへの変更は、保存を明示的に要求したとき (C-x C-sなど) に行います。

auto-save ファイルの名前は通常、visit しているファイル名の前後に ‘#’をつけて作られます。したがって foo.cというファイルを visit しているバッファーは、#foo.c#というファイルに自動保存されます。ファイルを visit していないバッファーのほとんどは、明示的に要求した場合だけ自動保存されます。それらのバッファーが自動保存されるとき、auto-save ファイル名は、バッファー名の前後に ‘#’をつけて、その後ろに数字と文字を付け加えて一意な名前にします。たとえば送信メッセージを作成する*mail*バッファーは、#*mail*#704juuのような名前のファイルに、自動保存されます。Emacs の一部 (関数 make-auto-save-file-nameおよび auto-save-file-name-p) を違った方法で再プログラムしない限り、auto-save ファイル名はこの方法で作成されますバッファーの自動保存に使われるファイル名は、そのバッファーの自動保存をオンにしたときに計算されます。

変数 auto-save-file-name-transformsは、auto-save ファイル名をある程度制御することを許します。これに一連の正規表現を指定して置換することにより、auto-save ファイル名を変更します。デフォルト値は、リモートのファイル (Section 15.15 [Remote Files], page 169 を参照してください) を、ローカルマシンの一時ディレクトリーの auto-save ファイルに変換します。

大きなバッファーから大量のテキストを削除したとき、そのバッファーにたいする自動保存は一時的にオフになります。これは、もしテキストをうっかり削除してしまった場合、それが auto-save ファイルに含まれていて、そこから探せる方が便利だからです。これが発生した後、再び自動保存を有効にするには、バッファーを C-x C-sで保存するか、C-u 1 M-x auto-save-modeを使います。

別の auto-save ファイルではなく、visit しているファイル自体に自動保存したい場合は、グローバルなマイナーモード auto-save-visited-modeを有効にします。このモードでは、自動保存は明示的な保存と等価です。このモードは上述の auto-saveと直交 (orthogonal) するモードであり、両方同時に有効にできることに注意してください。しかし、いくつかのバッファーで auto-saveモードがアクティブで、かつ時代遅れの auto-save-visited-file-name変数が非 nil値の場合、そのバッファーは auto-save-visited-modeの影響を受けないでしょう。

auto-save-visited-modeモードの自動保存処理の間隔は、変数 auto-save-visited-intervalを使用してカスタマイズできます。auto-save-intervalと auto-save-timeoutは、auto-save-visited-modeに影響を与えません。これらの変数の詳細は、Section 15.6.2 [Auto Save Control], page 160 を参照してください。

バッファーの auto-save ファイルは、そのバッファーを visit しているファイルに保存したとき削除されます (変数 delete-auto-save-filesを nilにセットすることにより、これを禁じることができます)。C-x C-wまたは set-visited-file-nameで、visit されているファイル名を変更することにより、あたらしく visit されているファイル名にもとづいて auto-save ファイル名はリネームされます。

バッファーを kill しても、デフォルトではそのバッファーの auto-save ファイルは削除されません。しかし kill-buffer-delete-auto-save-filesが非 nilなら、auto-save をもつファイルの kill において、Emacs がユーザーに auto-save ファイルを削除するかどうか問い合わせるようにできます (これは delete-auto-save-filesが nilなら抑制される)。

15.6.2 自動保存の制御

ファイルを visit する度に、そのファイルバッファーの自動保存は、オンになります (バッチモードでは異なります。Section C.2 [Initial Options], page 576 を参照してください)。この変数のデフォルトは tなので、ファイルを visit しているバッファーの自動保存は通常、常に行われます。現在のバッファーの自動保存を切り替えるには、M-x auto-save-modeとタイプします。Auto Save モードはバッファーローカルに動作するマイナーモードです (Section 20.2 [Minor Modes], page 241 を参照してください)。

Emacs は最後に自動保存されてから何文字タイプしたかにもとづいて、定期的に自動保存を行います。変数 auto-save-intervalは、自動保存と自動保存の間に何文字タイプされたかを指定します。デフォルトは 300 です。Emacs は小さすぎる値は受け付けません。auto-save-intervalを 20 より小さな値にカスタマイズした場合、Emacs は 20 と指定されたかのように振る舞います。

自動保存はタイピングをストップしたときも行われます。デフォルトでは 30 秒アイドル状態が続くと実行されます。(このとき Emacs はガベージコレクションも実行します; Section “Garbage Collection” in *The Emacs Lisp Reference Manual* を参照してください)。この間隔を変更するには、変数 auto-save-timeoutをカスタマイズします。実際の実行間隔は、現在のバッファーの大きさに応じて長くなります。これは、自動保存が実感できるほど時間がかかるような大きなバッファーを編集しているときは、それをなるべく無くすようにする狙いです。アイドル状態のときの自動保存

は、2 つの事を達成します。最初に、端末を少しの間離れるときなどに、すべての作業の保存を保証します。次に、実際にタイプしているときは自動保存を避けます。

`auto-save-visited-mode`が有効なとき、Emacs はアイドル 5 秒後に、ファイルを visit しているバッファーを自動保存するでしょう。アイドル時の間隔は、変数 `auto-save-visited-interval`でカスタマイズできます。

Emacs は、致命的なエラーが発生したときも自動保存を行います。これには‘`kill %emacs`’のようなコマンドによる Emacs ジョブの kill、電話回線やネットワーク回線の切断が含まれます。

コマンド `M-x do-auto-save`により、明示的に自動保存を行うことができます。

15.6.3 自動保存からのデータ復旧

コマンド `M-x recover-file RET` *file* `RET`により、auto-save ファイルの内容を使って、失われたデータを復旧できます。これは *file*を visit して、(確認を求めた後で)auto-save ファイル`#`*file*`#`の内容をリストアします。その後 `C-x C-s`で復旧したテキストを、*file*自身に保存できます。たとえばファイル `foo.c`を、その auto-save ファイル`#foo.c#`で復旧するには、以下のようにします:

```
M-x recover-file RET foo.c RET
yes RET
C-x C-s
```

`M-x recover-file`は確認を求める前に、指定したファイルと auto-save ファイルのあるディレクトリー一覧を表示するので、ファイルのサイズや日付を確認できます。auto-save の方が古い場合、`M-x recover-file`はそれを読み込むように提案しません。

Emacs またはコンピューターがクラッシュしたとき、`M-x recover-session`コマンドで編集していたすべてのファイルを、それらの auto-save ファイルで復旧できます。これは最初に中断されたセッションの記録された一覧を表示します。ポイントを移動して選択してから、`C-c C-c`をタイプします。

`recover-session`は、そのセッションの間に編集されていた各ファイルについて、ファイルを復旧するか尋ねます。`y`と応えると `recover-file`を呼び出し、通常の方法で復旧を行います。これは元のファイルと auto-save ファイルの日付を表示して、ファイルの復旧を行うかもう 1 度尋ねます。

`recover-session`が完了すると、復旧を選択したファイルが Emacs バッファーに表示されます。実際にファイル自体を更新するためには、これらを保存するのが唯一の方法です。

Emacs は中断されたセッションについての情報を、ディレクトリー`~/.emacs.d/auto-save-list/`の、`.saves-`*pid*`-`*hostname*`~`という名前のファイルに記録します。`auto-save-list-file-prefix`を `nil`にセットすると、復旧用にセッションが記録されなくなります。

15.7 ファイルのエイリアス

シンボリックリンクとハードリンクは、同じファイルを参照するためにいくつかの名前を使うことを可能にします。ハードリンクは、ファイルを直接参照する他の名前です。それらすべての名前は同じように有効で、それらの間に優先順位はありません。対照的にシンボリックリンクは、ある種の定義されたエイリアス (別名) です。`foo`が `bar`へのシンボリックリンクの場合、そのファイルをどちらの名前でも参照できますが、`bar`が実際の名前で、`foo`はエイリアスに過ぎません。シンボリックリンクがディレクトリーを指すときは、さらに複雑なケースが発生します。

Emacs がすでに異なる名前で visit している場合、通常はエコーエリアにメッセージを表示して、そのファイルを visit している既存のバッファーを使います。これはハードリンクおよびシンボリックリンクをサポートしているシステム、または長いファイル名を切り詰めるシステムで長い名前のファイルを使っている場合、またはファイル名の大文字小文字を区別しないシステムで発生します。変数

find-file-suppress-same-file-warningsを非nil値にセットすることにより、メッセージを表示しないようにできます。変数find-file-existing-other-nameをnilにセットすれば、この機能全体を無効にできます。その場合、同じファイルを異なる名前で visit すると、それぞれのファイル名で別々のバッファーが使われます。

変数find-file-visit-truenameが非nilの場合、バッファー用に記録されるファイル名は、指定した名前ではなく、ファイルの本当の名前 (これはすべてのシンボリックリンクを対象の名前で置き換えて作られます) が使われます。find-file-visit-truenameをセットするは、find-file-existing-other-nameにも暗に影響します。

シンボリックリンクを通じてアクセスされるようなディレクトリーにたいしては通常、優先的にリンクされた名前を Emacs に表示させたいときがあります。これを行うにはdirectory-abbrev-alistをカスタマイズします。このリストの各要素は(*from* . *to*)という書式です。これはディレクトリー名に *from*が出現したときは常に、*from*を *to*で置き換えることを意味します。文字列 *from*は正規表現です (Section 12.6 [Regexps], page 115 を参照してください)。正規表現はディレクトリー名の最初の文字にマッチさせる必要があるので、‘\`’で始まります (埋め込みの改行をサポートするディレクトリー名の場合は、‘^’で無効にします)。*to*には同じディレクトリーを指す、絶対パスによる普通のディレクトリー名を指定する必要があります。文字列 *to*でホームディレクトリーを指定するのに、‘~’を使用しないでください。Emacs はこれらの変換を個別に行います。以下は通常シンボリックリンク/fsfでアクセスされる、/home/fsfを指定する例です:

```
(("\\`/home/fsf" . "/fsf"))
```

15.8 ファイルディレクトリー

ファイルシステムは、ファイルをディレクトリーにグループ化します。ディレクトリーリストは、ディレクトリーに含まれるファイルのリストです。Emacs はディレクトリーを作成および削除するコマンドを提供し、簡単な形式 (ファイル名のみ)、および詳細な形式 (サイズ、日付、その他の属性を含む) のディレクトリーリストを作成します。Emacs には Dired と呼ばれるディレクトリーブラウザーも含まれています。詳細は Chapter 27 [Dired], page 380(C-x dで呼び出せる) を参照してください。

C-x C-d *dir-or-pattern* RET
: 簡単なディレクトリーリストを表示します (list-directory)。

C-u C-x C-d *dir-or-pattern* RET
: 詳細なディレクトリーリストを表示します。

M-x make-directory RET *dirname* RET
: *dirname*という名前の新しいディレクトリーを作成します。

M-x delete-directory RET *dirname* RET
: *dirname*という名前のディレクトリーを削除します。もし空でない場合、それらを再帰的に削除するか尋ねます。

ディレクトリーリストを表示するコマンドは、C-x C-d (list-directory) です。これはミニバッファーを使って、リストを表示するディレクトリーと、リストするファイルを指定するワイルドカードが含まれたパターンの両方により、ファイル名を読み取ります。たとえば

```
C-x C-d /u2/emacs/etc RET
```

これはディレクトリー/u2/emacs/etcのファイルをリストします。以下はファイル名のパターンを指定する例です。

```
C-x C-d /u2/emacs/src/*.c RET
```

通常、C-x C-dは名前だけを含む、簡単なディレクトリーリストを表示します。数引数 (値は関係なし) は、サイズ、日付、所有者を含む詳細な一覧を作成するよう指示します。

ディレクトリーリストのテキストは、主に下位プロセスとして lsを実行することにより取得されます。2 つの Emacs 変数が、lsに指定するスイッチを制御します。list-directory-brief-switchesには、簡単な一覧に使うためのスイッチを文字列で指定します (デフォルトは"-CF")。list-directory-verbose-switchesには、詳細な一覧に使うためのスイッチを文字列で指定します (デフォルトは"-l")。

詳細なディレクトリーのリストでは、Emacs はディレクトリーを含むディスク上のフリー領域サイズに関する情報を追加します。

コマンド M-x delete-directoryは、ミニバッファーを使ってディレクトリー名の入力を求め、空のときはディレクトリーを削除します。ディレクトリーが空でない場合、再帰的に削除するか確認を求めます。“Trash(ごみ箱)”(または “Recycle Bin”) の機能をもつシステムでは、変数 delete-by-moving-to-trashを tに変更することにより、指定したディレクトリーを無条件に削除するかわりに、ごみ箱に移動します。ごみ箱の使い方についての情報は、Section 15.12 [Misc File Ops], page 167 を参照してください。

15.9 ファイルの比較

コマンド M-x diffは、ミニバッファーを使って 2 つのファイル名の入力を求め、*diff*という名前のバッファーに、2 つのファイルの違いを表示します。これは diffプログラムに、変数 diff-switches で指定されたオプションを指定して実行することにより機能します。diff-switchesには文字列を指定します。デフォルトは unified context diff 形式を指定する"-u"です。プログラムについての情報は、Section “Diff” in *Comparing and Merging Files* を参照してください。

diffコマンドの出力は、Diff モードと呼ばれるメジャーモードを使って表示されます。Section 15.10 [Diff Mode], page 164 を参照してください。

(より高機能な) 代替物は M-x ediffです (Section “Ediff” in *The Ediff Manual* を参照)。

コマンド M-x diff-backupは指定したファイルと、そのファイルの一番最近のバックアップを比較します。バックアップファイル名を指定したときは、diff-backupは指定されたバックアップファイルと、それの元となるファイルを比較します。それ以外の点は M-x diffと同じです。

コマンド M-x diff-buffer-with-fileは指定されたバッファーと、それに対応するファイルを比較します。これはバッファーを保存すると、ファイルにどのような変更がされるかを表示します。

コマンド M-x diff-buffersは指定した 2 つのバッファーのコンテンツを比較します。

コマンド M-x compare-windowsはカレントウィンドウと、カレントウィンドウの前に選択されていたウィンドウを比較します (Emacs のウィンドウについての詳細は Chapter 17 [Windows], page 186 を参照)。比較はそれぞれのウィンドウのポイント位置から、それぞれのバッファーのポイントの初期位置を、対応するバッファーのマークリング (Section 8.4 [Mark Ring], page 56 を参照) に push した後に開始されます。それから各ウィンドウのポイントを 1 文字ずつ前方に移動していきます。文字がマッチしなくなるとコマンドは終了します。

コマンドを開始したとき、2 つのウィンドウのポイントの後ろのテキストがマッチしない場合、M-x compare-windowsは 2 つのウィンドウでマッチするテキストが見つかるまでポイントを進めてから終了します。したがって M-x compare-windowsを繰り返し使うと (see Section 4.11 [Repeating], page 27 を参照)、毎回 1 つのマッチする範囲をスキップするか、次の開始点を探します。

数引数を指定すると、compare-windowsは空白文字の違いを無視します。変数 compare-ignore-caseが非 nilの場合、大文字小文字の違いを無視して比較します。変数 compare-ignore-whitespaceが非 nilの場合、compare-windowsはデフォルトでは空白文字の

違いを無視しますが、数引数が指定されたときは、その回のコマンド呼び出しでは、これをオフにします。

`M-x smerge-mode`を使って、Smerge モードに切り替えることができます。これは`diff3`プログラムの出力を編集するマイナーモードです。これは通常、バージョン管理システムと、バージョン管理システムの外での update をマージするとき、ファイルへの変更が競合して失敗した結果です。Smerge モードは特定の変更を選択することにより、競合を解決するコマンドを提供します。

ファイルをマージする強力なインターフェースを提供する Emerge 機能については、⟨undefined⟩ [Emerge], page ⟨undefined⟩ を参照してください。

15.10 Diff モード

Diff モードは、`M-x diff`や他の同様なコマンドの出力のために使用されるメジャーモードです。この種の出力は *patch* と呼ばれます。なぜならそれが特定の変更を自動的に適用するために、`patch`コマンドに渡されるからです。手動で Diff モードを選択するには、`M-x diff-mode`とタイプします。

パッチに指定された変更は、*hunk(*欲張り*)* にグループ化されます。これは変更された行を 1 行以上含むテキストと、それに隣接するテキストです。hunks もまた、変更のコンテキストを提供するために、変更されていない行も含みます。それぞれの hunk には *hunk* ヘッダーが前についていて、これは hunk の変更が発生した、古い行番号と、新しい行番号が指定されます。Diff モードは実際の hunk の内容と区別するため、hunk ヘッダーをハイライトします。

パッチ内の最初の hunk の前にはファイルヘッダーがあり、それはそのファイルの新たなバージョンと古いバージョンの名前と、それらのタイムスタンプを示します。パッチが複数ファイルの変更を表す場合、各ファイルがそのファイルの変更の最初の hunk の前にそのようなヘッダーをもつことになります。

他のバッファーと同様に、Diff モードのバッファーを編集することができます (もし読み込み専用の場合、最初にそれを書き込み可にする必要があります。Section 16.3 [Misc Buffer], page 179 を参照のこと)。hunk を編集すると、Diff モードは patch を正しい状態に保ち、`patch`により正しく適用できるように、hunk ヘッダーの行番号を自動的に修正しようと試みます。自動的な行番号の修正を無効にするには、変数`diff-update-on-the-fly`を`nil`に変更してください。

Diff モードは`M-g M-n`やエラーメッセージを処理する他のコマンドによりコンパイラーのエラーメッセージとして扱われるように hunk をアレンジします (Section 24.2 [Compilation Mode], page 310 を参照)。したがって、対応するソースの位置を visit するために、Compilation モードのコマンドを使用できます。

それに加えて Diff モードは、移動、操作、patch の一部を適用するために、以下のコマンドを提供します:

`M-n` 次の hunk-start に移動します (`diff-hunk-next`)。プレフィクス引数を指定した場合は、次の *n*番目の hunk に前方へ移動します。

デフォルトでは Diff モードは Emacs が表示する際により良い粒度で変更をハイライトできるように hunk をリファイン *(refine:* 洗練する*)* します。`diff-refine`にシンボル`navigation`をセットすると、Diff はこのコマンドや`diff-hunk-prev`で移動した hunk だけをリファインします。

`M-p` 前の hunk が開始される位置に移動します (`diff-hunk-prev`)。プレフィクス引数を指定した場合には、前の *n*番目の hunk に後方へ移動します。`diff-refine`にシンボル`navigation`をセットしていれば、このコマンドは`M-n`と同様に移動先の hunk をリファインします。

`M-}` 複数ファイルへの patch で、次のファイルが開始される位置に移動します (`diff-file-next`)。プレフィクス引数を指定した場合は、次の *n*番目のファイルの先頭へ、前方に移動します。

`M-{` 複数ファイルへの patch で、前のファイルが開始される位置に移動します (`diff-file-prev`)。プレフィクス引数を指定した場合は、前の *n*番目のファイルの先頭へ、後方に移動します。

`M-k` ポイントがある位置の hunk を kill します (`diff-hunk-kill`)。

`M-K` 複数ファイルへの patch で、現在のファイル部分を kill します (`diff-file-kill`)。

`C-c C-a` その hunk を、ターゲットファイルに適用します (`diff-apply-hunk`)。`C-u`によるプレフィクス引数を与えた場合は、この hunk をリバート (“新しい” バージョンを “古い” バージョンに変更するリバース hunk を適用する) します。`diff-jump-to-old-file` が非 `nil`なら、かわりに s の hunk を “古い” バージョンに適用します。

`C-c C-b` ポイント位置の hunk の変更を、よりよい粒度でハイライトします (`diff-refine-hunk`)。これにより変更された各行について実際に変更された箇所を確実に見ることができます。

デフォルトでは Diff モードは Emacs が hunk を表示する際に hunk をリファインするので、`diff-refine`を非デフォルト値にカスタマイズしている場合にはこのコマンドが有用だと思うかもしれません。

`C-c C-c` その hunk に対応するソースファイルの該当行にジャンプします (`diff-goto-source`)。デフォルトではファイルヘッダーの最初に示される、“新しい” バージョンのファイルにジャンプします。プレフィクス引数を与えた場合には、かわりに “古い” バージョンにジャンプします。`diff-jump-to-old-file`が非 `nil`なら、このコマンドはデフォルトで “古い” バージョンにジャンプして、プレフィクス引数の意味も逆になります。プレフィクス引数が 8 より大 (たとえば `C-u C-u C-c C-c`とタイプした場合) なら、このコマンドは次回呼び出しにたいして `diff-jump-to-old-file`のセットも行います。ソースファイルがバージョンコントロール (Section 25.1 [Version Control], page 332 を参照) の配下にある場合には、デフォルトでは作業ファイルにジャンプします。プレフィクス引数を与えると、ポイントが古い行にあれば “旧” リビジョン (Section 25.1.6 [Old Revisions], page 341 を参照)、それ以外なら “新” リビジョンにジャンプします。

`C-c C-e` このパッチで Ediff セッションを開始します。Section “Ediff” in *The Ediff Manual* を参照してください。

`C-c C-n` 表示を現在の hunk に制限します (`diff-restrict-view`)。Section 11.5 [Narrowing], page 81 を参照してください。プレフィクス引数を指定すると、複数ファイルへの patch で、表示を現在のファイルに制限します。制限を解除するには、`C-x n w` (`widen`) を使います。

`C-c C-r` バッファー全体にたいする比較方向を逆転します (`diff-reverse-direction`)。プレフィクス引数を与えた場合は、カレントリージョンの内部でのみ方向を逆転します (Chapter 8 [Mark], page 52 を参照)。方向の逆転とは、hunk とファイル冒頭のヘッダーを、“新しい” バージョンから “古い” バージョンにするパッチを生成するように変更することを意味します。

`C-c C-s` ポイント位置で hunk を 2 つの別個の hunk に分割します (`diff-split-hunk`)。これは hunk ヘッダーの挿入ち、カレント hunk のヘッダーの変更を行います。これは手動

で patch を編集するために有用で、`diff`プログラムに`-u`または`--unified`オプションを指定して生成された、*unified diff format(*統一 *diff* フォーマット*)* だけで機能します。`diff`に`-c`または`--context`オプションを指定して生成された、*context diff format(*コンテキスト *diff* フォーマット*)* の hunk を分割するには、最初に `C-c C-u`で、バッファーを unified diff format に変換する必要があります。

`C-c C-d` バッファー全体を、*context diff format* に変換します (`diff-unified->context`)。プレフィクス引数を指定すると、リージョンの hunk だけを変換します。

`C-c C-u` バッファー全体を unified diff format に変換します (`diff-context->unified`)。プレフィクス引数を指定すると、unified format から context format に変換します。マークがアクティブのときは、リージョンの hunk だけを変換します。

`C-c C-l` カレント hunk を再生成します (`diff-refresh-hunk`)。

`C-c C-w` 空白文字の変更を無視して、カレント hunk を再生成します (`diff-ignore-whitespace-hunk`)。

`C-x 4 A` それぞれの hunk について、`C-x 4 a`が行うように ChangeLog(Section 25.3 [Change Log], page 355 を参照してください) のエントリーを生成します (`diff-add-change-log-entries-other-window`)。これは、あとで実際に変更の説明を記入できるように、変更ログの雛形を作ります。Diff モードでの `C-x 4 a`自体は、現在の hunk のファイルのためのものですが、関数名は patch 自体から取得します。これは patch により削除される関数のための、ログエントリーを作るのに有用です。

patch には変更された行の行末に、無意識に入力された望んでいない空白文字が含まれている場合があります。この問題を扱うには 2 つの方法があります。1 つ目は Diff バッファーで Whitespace モード (Section 11.17 [Useless Whitespace], page 95 を参照してください) を有効にする方法で、これは自動的に変更された行の行末にある空白文字をハイライトします。2 つ目はコマンド `M-x diff-delete-trailing-whitespace`を使う方法で、patch により変更された行の行末の空白文字を検索して、patch と patch されたソースファイルの両方からそれを取り除きます。このコマンドは変更を保存しないので、ユーザーが変更を保存するか決定することができます (変更されたファイルはエコーエリアに表示されます)。プレフィクス引数を指定すると、patch された (“新しい”) ファイルではなく、元の (“古い”) ソースファイルを変更しようと試みます。

`diff-font-lock-syntax`が非 `nil`なら hunk 内のソース断片は適切なメジャーモードに応じてハイライトされます。

15.11 ファイルのコピー、命名、リネーム。

Emacs にはファイルをコピー、命名、リネームするためのコマンドがいくつかあります。これらはすべてミニバッファーを使用して *old*(または *target*) と *new*の 2 つのファイル名を読み取り、それらをコピー、またはファイル名に調整します。これらのコマンドは、ワイルドカードを含むファイル名は許容しません。

これらすべてのコマンドは、引数 *new*が単なるディレクトリー名 (Section “Directory Names” in *the Emacs Lisp Reference Manual* を参照) の場合には、そのディレクトリーを実際の新しい名前として、*old*をそれの非ディレクトリー成分とします。たとえば、コマンド `M-x rename-file RET ~/foo RET /tmp/ RET` は、`~/foo`を`/tmp/foo`にリネームします。GNU、およびその他の POSIX システムでは、ディレクトリー名は ‘`/`’で終端されます。

これらのコマンドはすべて、新しいファイル名がすでに存在する場合は確認を求めます。

`M-x copy-file`はファイル *old*のコンテンツを、ファイル *new*にコピーします。

`M-x copy-directory`は、シェルコマンド `cp -r`と同じようにディレクトリーをコピーします。*new*がディレクトリー名の場合、このコマンドは*old*ディレクトリーのコピーを作成して、それを *new* の下に配します。それ以外では、このコマンドは*old*のコンテンツを、新しい *new*という名前のディレクトリーにすべてコピーします。`copy-directory-create-symlink`が非 `nil`で *old*がシンボリックリンクなら、このコマンドはそのシンボリックリンク、`nil` (デフォルト) ならリンクをフォローしてかわりにその内容をコピーします。

`M-x rename-file`は、ファイル *old*を *new*にリネームします。すでにファイル名 *new*が存在する場合、確認に `yes`と答えなければリネームは行われません。なぜなら、名前 *new*の古い内容が失われてしまうからです。*old*と *new*が異なるファイルシステム上にある場合は、ファイル *old*がコピーされた後に削除されます。

`M-x add-name-to-file`は、既存のファイルの古い名前を削除せずに、新しい名前を追加します。新しい名前は、既存のファイルのハードリンクとして作成されます。新しい名前は、そのファイルがあるのと同じファイルシステムになければなりません。MS-Windows では、このコマンドはファイルが NTFS ファイルシステムにあるときだけ機能します。MS-DOS、およびその他いくつかのリモートシステムタイプでは、ファイルをコピーすることにより機能します。

`M-x make-symbolic-link`は、*target*を指す *new*という名前のシンボリックリンクを作成します。これにより、今後 *new*というファイルを開こうとすると、その時点で *target*という名前のファイルが何であれ、ファイルのオープンが行われたときは、そのファイルを開きます。その時点で *target*という名前のファイルが存在しないときはエラーになります。このコマンドは引数 *target*を展開しないので、リンクの対象を相対パスで指定できます。しかし、このコマンドは *target*内の先頭の '`~`'は展開するので、簡単にホームディレクトリーを指定できます。また、先頭の '`/:`'は取り除くので、リテラル '`~`'および '`/:`'で始まる相対パスを指定することができます。Section 15.16 [Quoted File Names], page 170 を参照してください。MS-Windows では、このコマンドは MS Windows Vista 以降だけで機能します。*new*がリモートのときは、そのシステムタイプに依存して機能します。

15.12 その他のファイル操作

Emacs には、ファイルを操作する他のコマンドがたくさん存在します。それらはすべて 1 つのファイルを操作します。ファイル名にワイルドカードは指定できません。

`M-x delete-file`はファイルの入力を求め、そのファイルを削除します。1 つのディレクトリーにある、複数のファイルを削除する場合、`delete-file`より Dired を使う方が便利でしょう。Section 27.3 [Dired Deletion], page 382 を参照してください。

`M-x move-file-to-trash`は、ファイルをシステムの *Trash*(または *Recycle Bin*) に移動します。この機能は、ほとんどのオペレーティングシステムで利用可能です。Trash に移動されたファイルは、後で気が変わったとき元に戻すことができます (trash に移動されたファイルのリストアはシステムに依存する)。

デフォルトでは、Emacs の削除コマンドは Trash を使いません。一般的な削除コマンドで Trash(それが利用可能な場合) を使うには、変数 `delete-by-moving-to-trash`を `t`に変更します。これはコマンド `M-x delete-file`と `M-x delete-directory` (Section 15.8 [Directories], page 162 を参照してください)、および Dired (Section 27.3 [Dired Deletion], page 382 を参照してください)の削除コマンドに影響を与えます。`M-x delete-file`および `M-x delete-directory`にプレフィクス引数を与えると、`delete-by-moving-to-trash`の値にかかわらず、Trash を使わずに完全に削除します。

もし `delete-by-moving-to-trash`をセットしていて Emacs で Trash ディレクトリーから手作業でファイルを削除する場合には、Trash ディレクトリーでは `D` (`dired-do-delete`) のようなコマンドを使ってもうまく動作しません (単に新たな名前をファイルに与えるだけで何も削除しない)。

削除できるようにするには、以下のような内容を含んだ.dir-locals.elを Trash ディレクトリーに作成する必要があります:

```
((dired-mode . ((delete-by-moving-to-trash . nil))))
```

ただしシステムの “empty trash(ごみ箱を空にする)” コマンドを使うとこの.dir-locals.elファイルも削除されてしまうので、手作業で Trash ディレクトリーからファイルを削除する場合のみ行う必要があることに注意してください。

M-x insert-file(C-x iも) は、指定したファイルの内容のコピーを、現在のポイント位置に挿入し、ポイントの位置は変更せずに挿入された内容の前に残します。挿入した内容の後の位置はマークリングに追加され、マークは非アクティブになります (Section 8.4 [Mark Ring], page 56 を参照してください)。

M-x insert-file-literallyも M-x insert-fileと同様ですが、ファイルは literally(そのまま) 挿入されます。つまり M-x find-file-literallyコマンド (Section 15.2 [Visiting], page 146 を参照してください) と同様に、特別なエンコーディングや変換なしに、ASCII文字の並びとして扱われます。

M-x write-regionは M-x insert-fileの逆です。このコマンドはリージョンの内容を、指定されたファイルにコピーします。M-x append-to-fileはリージョンのテキストを、指定されたファイルの末尾に加えます。Section 9.4 [Accumulating Text], page 67 を参照してください。変数write-region-inhibit-fsyncの値は、これらのコマンドおよびファイルの保存に影響を与えます。Section 15.3.3 [Customize Save], page 154 を参照してください。

M-x set-file-modesはファイル名と、その後にファイルモード *(file mode)* を読み込んで、指定されたファイルにそのファイルモードを適用します。ファイルモード (またはファイルパーミッション *(file permissions)* とも呼ばれます) は、ファイルが読み込み可能か、書き込み可能か、実行可能か、そしてそれは誰にたいしてなのかを決定します。このコマンドは、chmodコマンドに指定する形式の、シンボルまたは 8 進のファイルモードを読み取ります。たとえば ‘u+x’は、そのファイルを所有するユーザーに実行可能の権限を追加することを意味します。ファイルモードをサポートしないオペレーティングシステムでは、効果はありません。chmodはこの関数の便利なエイリアスです。

15.13 圧縮ファイルへのアクセス

Emacs は、圧縮されたファイルを visit するとき、自動的に解凍します。また、それらのファイルを変更して保存するときも、自動的に再圧縮します。Emacs は圧縮ファイルを名前で認識します。ファイル名が ‘.gz’で終わっていれば、それはファイルが gzipで圧縮されていることを示します。他の拡張子の場合は、他の圧縮プログラムを示します。

自動的な解凍と圧縮は、Emacs がファイル内容を操作するすべてに適用されます。これには visit、保存、内容のバッファーへの挿入、ロード、バイトコンパイルが含まれます。

この機能を無効にするには、コマンド M-x auto-compression-modeとタイプします。変数auto-compression-modeをカスタマイズすることにより、永続的に無効にすることができます。

15.14 ファイルアーカイブ

名前が ‘.tar’で終わるファイルは通常、tarプログラムで作られたアーカイブです。Emacs はそれらを、Tar モードと呼ばれる特別なモードで表示します。これは内容を Dired に似たリストで提供します (Chapter 27 [Dired], page 380 を参照してください)。リストの移動は Dired のときと同様で、アーカイブに含まれるファイルを visit できます。しかし Tar モードでは、Dired コマンドのすべてが利用可能ではありません。

Auto Compression モードが有効な場合 (Section 15.13 [Compressed Files], page 168 を参照してください)、Tar モードは圧縮アーカイブ (ファイルの拡張子が '.tgz'、.tar.Z、.tar.gz) も使うことができます。

キー e、f、RETはすべて、ファイルをファイル自身のバッファーに展開します。それをバッファーで編集して、バッファーを保存すると、編集されたバージョンで Tar バッファーのものを置き換えます。Tar バッファーでファイル名をマウスでクリックしても、同様なことが行えます。vはファイルをバッファーに View モードで展開します (Section 11.6 [View Mode], page 82 を参照してください)。oは、ファイルを展開して他のウィンドウで表示するので、ファイルの編集とアーカイブの操作を同時に行うことができます。

キー Iは、新しい (標準) ファイルをアーカイブに追加します。ファイルは最初は空ですが、上記のコマンドを使用してすぐに編集することができます。このコマンドはカレントのファイルの前に新しいファイルを挿入するので、Tar バッファーの最上行で使用すると、新しいファイルがアーカイブの最上行となり、バッファーの最後で使用すると、新しいファイルがアーカイブの最下行になります。

Dired と同様に、dは後で xを使ったときにファイルを削除するためにマークし、uはマークを外します。Cはファイルをアーカイブからディスクにコピーし、Rはアーカイブのファイルをリネームします。gはバッファーをディスク上のアーカイブでリバートします。キー M、G、Oは、ファイルのパーミッションビット、グループ、所有者を変更します。

Tar バッファーの保存により、構成要素に変更が施された、新しいバージョンのアーカイブをディスクに書き込みます。

Tar モードを使うのに、tarプログラムは必要ありません。Emacs は直接アーカイブを読み込みます。しかし圧縮アーカイブへのアクセスには、適切な解凍プログラムが必要です。

arc、jar、lzh、zip、rar、7z、zoo、および自己解凍実行形式の exeには、互いに似通った異なる Archive モードが使われます。

Archive モードのキーバインドは Tar モードと同様で、それに加えてキー mは後に続く操作のためにファイルをマークし、M-DELはマークされたファイルのマークをすべて外します。キー aは、1 行に収まらないようなアーカイブの、詳細なファイル情報の表示を切り替えます。ファイルのリネーム、ファイルモードや所有者の変更をサポートするアーカイブ書式は、いくつかに限られます。

Tar モードとは異なり、Archive モードはアーカイブの展開と格納に、アーカイブプログラムを実行します。しかし、展開したりアーカイブ内のファイルを操作するときだけこれらのプログラムが必要で、アーカイブの目録を見るには必要ありません。プログラム名とセットできるオプションの詳細は、Customize グループ 'Archive'でセットできます (Section 33.1.1 [Customization Groups], page 499 を参照してください)。

15.15 リモートファイル

他のマシンにあるファイルを、特別なファイル名構文を使って参照できます:

/method:*host*:*filename*
/method:*user*@*host*:*filename*
/method:*user*@*host*#*port*:*filename*

このリクエストを発行するために、Emacs は sshのような、リモートログインプログラムを使います。どの method を使うかは、常にファイル名で指定しなければなりません。たとえば /ssh:*user*@*host*:*filename*は sshを使います。ファイル名の method に擬似 method の '-'を使用したとき、Emacs は以下により method を選択します:

1. ホスト名が 'ftp.'(ドット付き) で始まるとき、Emacs は FTP を使います。
2. ユーザー名が 'ftp'または 'anonymous'のとき、Emacs は FTP を使います。

3. 変数 `tramp-default-method`が '`ftp`'にセットされているとき、Emacs は FTP を使います。
4. `ssh-agent`が実行されているとき、Emacs は `scp`を使います。
5. 上記以外の場合、Emacs は `ssh`を使います。

変数 `tramp-mode`を `nil`にセットすることにより、リモートファイル名の機能を完全にオフにすることができます。個別のケースについて機能をオフにするには、ファイル名を '`/:`'でクォートします (Section 15.16 [Quoted File Names], page 170 を参照してください)。

FTP を通じたリモートファイルへのアクセスは、以下で説明する Ange-FTP パッケージで処理されます。他の方法によりリモートファイルへのアクセスは Tramp パッケージにより処理され、これにはそれ自身のマニュアルがあります。*The Tramp Manual* を参照してください。

Ange-FTP パッケージでは、リモートファイル名にユーザー名 *user*がしているときは、FTP を通じてその名前でログインします。*user*が指定されていないとき、Emacs はローカルシステムのユーザー名でログインします。しかし変数 `ange-ftp-default-user`に文字列がセットされているときは、かわりにその文字列を使用します。Emacs は、ログイン時にパスワードの入力も求めます。

パフォーマンス的な理由により、FTP を通じたファイルのアクセス時に、デフォルトでは Emacs はバックアップファイルを作成しません。バックアップを作成するには、変数 `ange-ftp-make-backup-files`を非 `nil`値に変更してください。

デフォルトではリモートファイルの自動保存ファイルは、変数 `auto-save-file-name-transforms`で指定された、ローカルマシンの一時ディレクトリーに作成されます。Section 15.6.1 [Auto Save Files], page 159 を参照してください。

匿名 FTP でアクセスできるファイルを visit するには、特別なユーザー名 '`anonymous`'または '`ftp`'を使います。これらのユーザー名にたいするパスワードは、特別に処理されます。これは変数 `ange-ftp-generate-anonymous-password`により制御されます。この変数の値が文字列の場合、その文字列がパスワードとして使用されます。非 `nil`(デフォルト) の場合、`user-mail-address`の値が使用されます。`nil`の場合、Emacs は通常どおりパスワードの入力を求めます (Section 5.7 [Passwords], page 38 を参照してください)。

セキュリティー上の理由で、リモートマシンとの間にあるファイアーウォール *(firewall)* により、ファイルにアクセスできないときがあります。対象ファイルにアクセスできるマシンからゲートウェイ *(gateway)* マシンにログインできて、FTP サーバーがゲートウェイ機能をサポートしている場合は、リモートファイル名を使うことができます。これを行うには変数 `ange-ftp-gateway-host`にゲートウェイマシンの名前をセットして、`ange-ftp-smart-gateway`を `t`にセットする必要があります。それ以外の場合でもリモートファイル名が機能するようにできますが、その方法は複雑です。これらの方法は、`M-x finder-commentary RET ange-ftp RET`とタイプして読むことができます。

15.16 ファイル名のクォート

特殊な文字や構文を含むファイルにたいする特別な効果を防ぐために、絶対ファイル名をクォートできます。これを行うには先頭に '`/:`'を追加します。

たとえばリモートにあるように見える名前の、ローカルなファイルの名前をクォートすることにより、リモートファイル名として扱われないようにすることができます。したがって名前が`/foo:`というディレクトリーがあり、そこに `bar`という名前のファイルがある場合、Emacs では、そのファイルを '`/:/foo:/bar`'という名前で参照できます。

リモートファイル名もローカル部分の特殊文字だけをクォートしたい場合はローカル部分だけをクォートできます。たとえば '`/ssh:baz:/:/foo:/bar`'は、ホスト `baz`上のディレクトリー`/foo:`のファイル `bar`を参照します。

‘/:’は、‘~’がユーザーのホームディレクトリーを意味する、特別な文字として扱われることを防ぐこともできます。たとえば/:/tmp/~hackは、ディレクトリー/tmpのファイル~hackを参照します。

‘/:’によるクォートは、ミニバッファーでの名前に‘$’を含むファイル名の入力にも使用できます。これが機能するには、ミニバッファーの最初の内容が‘/:’で始まらなければなりません (2 回‘$’を記述することでも同様な効果が得られます。詳細は [File Names with $], page 146 を参照してください)。

ファイルを visit するときに、ワイルドカードをクォートすることもできます。たとえば/:/tmp/foo*barは、ファイル/tmp/foo*barを visit します。

同じ効果を得るための別の方法は、/tmp/foo[*]barと入力する方法です。これは/tmp/foo*barだけにマッチするワイルドカード指定です。しかしクォートしなくても同じ結果が得られるので、ワイルドカード文字をクォートする必要がない場合がたくさんあります。たとえば/tmpの中に‘foo’で始まり‘bar’で終わるファイルが foo*barだけの場合、/tmp/foo*barと指定することにより、/tmp/foo*barだけを visit することができます。

15.17 ファイル名キャッシュ

ファイル名キャッシュ*(file name cache)* により、ファイルがどこにあるか正確に覚えていなくても、名前でファイルがどこにあるかを簡単に指定することができます。ファイル名をミニバッファーでタイプするとき、C-TAB (file-cache-minibuffer-complete) で、ファイル名キャッシュを使ったファイル名を補完が行なわれます。C-TABを繰り返すと、最初にタイプした内容から補完できる利用可能な候補を順番に表示します (しかし C-TAB文字は、多くのテキスト端末でタイプできないことに注意してください)。

ファイル名キャッシュは自動的に充填されません。かわりに以下のコマンドを使ってファイル名をキャッシュにロードします。

M-x file-cache-add-directory RET *directory* RET
: *directory*の各ファイルを、ファイル名キャッシュに加えます。

M-x file-cache-add-directory-using-find RET *directory* RET
: *directory*の各ファイルを、ファイル名キャッシュに加えるとともに、ネストされたサブディレクトリーのすべてのファイルを、ファイル名キャッシュに加えます。

M-x file-cache-add-directory-using-locate RET *directory* RET
: *directory*の各ファイルを、ファイル名キャッシュに加えるとともに、ネストされたサブディレクトリーのすべてのファイルを、ファイル名キャッシュに加えます。ファイルの検索には、locateを使用します。

M-x file-cache-add-directory-list RET *variable* RET
: *variable*にリストされた各ディレクトリーのファイル名を、ファイル名キャッシュに加えます。*variable*は Lisp 変数で、load-pathと同様、値はディレクトリーのリストです。

M-x file-cache-clear-cache RET
: キャッシュをクリアーして、すべてのファイル名を削除します。

ファイル名キャッシュは永続的ではありません。キャッシュが維持されるのは Emacs のセッションの間だけです。キャッシュの内容は、file-cache-displayコマンドで閲覧できます。

15.18 ファイル検索の便利な機能

このセクションでは、最近開いたファイルの検索、バッファーからのファイル名の読み取りなどの便利な機能を紹介します。

M-x recentf-modeで Recentf モードを有効にすると、Emacs g最近オープンしたファイルのリストを保守します。このリストからファイルをオープンするには、M-x recentf-openコマンドを使用します。このモードが有効な際には、'File'メニューにこれらのファイルのいずれかを visit するために使用できるサブメニューが含まれます。M-x recentf-save-listは現在の recentf-list をファイルに保存、M-x recentf-edit-listでそれを編集できます。

M-x ffapコマンドは、find-fileを一般化した、より強力なデフォルト決定のための機能で、基本的にはポイント位置のテキストにもとづいて決定を行ないます。Partial Completion モードは find-fileを拡張する、ffapとともに使用できるその他の機能を提案します。Section 5.4.5 [Completion Options], page 35 を参照してください。

15.19 イメージファイルの visit

イメージファイルを visit することにより Image モードが選択されます。このメジャーモードでは C-c C-c (image-toggle-display) とタイプすることにより、Emacs バッファーでファイルのイメージ表示とイメージの元となるテキスト (または raw byte) 表示を切り替えることができます。さらに C-c C-x (image-toggle-hex-display) とタイプすると、Emacs バッファー内でイメージとしてファイルを表示、または 16 進表記で表示を切り替えることができます。ファイルのイメージ表示は、Emacs がそのようなイメージの表示をサポートするようにコンパイルされているときだけ機能します。

イメージを表示するウィンドウよりイメージの幅または高さが大きければ、通常のポイント移動キー (C-f、C-p、...) によりイメージの他の部分が表示されます。しかしデフォルトではイメージはウィンドウにフィットするように自動的にリサイズされるので、これはオプション image-auto-resizeと image-auto-resizeを使用してデフォルトの挙動をカスタマイズしている場合だけ必要になります。

コマンド image-transform-fit-to-windowによって、手動でイメージをリサイズできます。これはイメージをウィンドウの高さと幅の両方にフィットさせるコマンドで、s wにバインドされています。イメージを元サイズのパーセントでスケーリングするために使うコマンドは、s pにバインドされている image-transform-set-percentです。スケール因子 (scale factor) を指定してイメージをスケーリングにするコマンドは、s sにバインドされている image-transform-set-scale、すべてをの変換を初期状態にリセットするコマンドは、s 0にバインドされている image-transform-reset-to-initial、あるいは s oにバインドされている image-transform-reset-to-originalを使用してください。

n (image-next-file) および p (image-previous-file) を押下することにより、同一ディレクトリー内の次または前のイメージを visit できます。これらのコマンドは次/前のイメージファイルがどれかを判断するために、“親” となる Dired バッファーを調べます。これらのコマンドはアーカイブファイルからのファイルオープン時にも機能し、その場合にはかわりに archive モードのバッファーを調べます。archive バッファーや dired の “親” バッファーが見つからなければ、Dired バッファーがオープンします。

イメージの閲覧時に、後で処理するために (たとえば別の場所へコピーするために一群のイメージを選択したいとき) ファイルをマークできると便利なことがあります。m (image-mode-mark-file) コマンドは、カレントファイルのディレクトリーを表示中のすべての Dired バッファーのカレントファイルをマークします。そのようなバッファーがオープンされていなければ、新たなバッファーにディレクトリーがオープンされます。ファイルのマークをオフにするにはコマンド u (image-mode-mark-file) を使用します。最後にカレントバッファーのファイル名だけを kill リングにコピーしたければ、コマンド w (image-mode-copy-file-name-as-kill) を使用できます。

アニメーションが可能なイメージの場合、コマンド RET (image-toggle-animation) で、アニメーションの開始と停止ができます。オプション image-animate-loopが非 nilでなければ、アニメーションの再生は 1 回です。f (image-next-frame) と b (image-previous-frame) により、アニメーションの各フレームを切り替えることができます。これらのコマンドは数引数を指定することにより、指定した数の分だけ先のフレームを表示できます。F (image-goto-frame) により、特定のフレームを指定することができます。フレームは 1 からインデックスがつきます。a + (image-increase-speed) とタイプすると、アニメーションのスピードが早くなり、a - (image-decrease-speed) で遅くなります。また a r (image-reverse-speed) で逆再生されます。コマンド a 0 (image-reset-speed) は、スピードを元の値にリセットします。

上述した Image モード固有のキーバインディングの他にも、任意の Emacs バッファーに表示されているイメージには、イメージやイメージ内ににポイントがある際の特別なキーバインディングが存在します。

i + イメージサイズを 20%拡大します (image-increase-size)。拡大率はプレフィックス数引数で制御します。値を *n*にするとサイズに 1 + *n* / 10 という因子を乗ずる意味となるので、C-u 5 i +ならサイズを 50%拡大する意味になります。

i - イメージサイズを 20%縮小します (image-increase-size)。縮小率はプレフィックス数引数で制御します。値を *n*にするとサイズに 1 - *n* / 10 という因子を乗ずる意味となるので、C-u 3 i +ならサイズを 30%縮小する意味になります。

i r イメージを時計回りに 90 度回転します (image-rotate)。プレフィックス引数を指定すると、反時計回りに 90 度回転します。このコマンドはスライス (slice: 特定の領域を選択されたイメージ) では利用できないことに注意してください。

i h イメージを水平方向にフリップ (flip: 反転) します (image-flip-horizontally)。これによりイメージはあたかも垂直な鏡で反射したかのように表示されます。このコマンドはスライスでは利用できないことに注意してください。

i v イメージを垂直方向にフリップ (flip: 反転) します (image-flip-vertically)。これによりイメージはあたかも水平な鏡で反射したかのように表示されます。このコマンドはスライスでは利用できないことに注意してください。

i o イメージをファイルに保存します (image-save)。このコマンドはイメージを保存するファイル名の入力を求めます。

i c イメージをクロップ (crop: イメージの切り取りや抜き出しの意) します (image-crop)。このコマンドは、システムにイメージのクロップやカットに使用できる外部プログラムがインストールされている場合のみ利用できます。どのプログラムを使用するかはユーザーオプション image-crop-crop-commandにより決定されます。デフォルトは ImageMagick's の convertプログラムです。このコマンドは、イメージの上にマウスで移動やリサイズできる重ね合わせられた長方形フレームを表示します。mをタイプするとリサイズではなくマウスでフレームを移動、sなら正方形のフレームを移動します。クロップするフレームの位置やサイズが決まったら、RETとタイプしてフレームの下にある部分を実際にクロップします。クロップせずに exit するには qをタイプしてください。クロップしたイメージは i o、または M-x image-save で保存できます。

i x イメージから矩形 (rectangle) をカットします (image-cut)。image-crop(変数 image-crop-cut-commandでイメージのカットを行う外部プログラムの定義も必要) と同じように機能しますが、このコマンドはクロップするのではなくフレーム内部分を削除して、その部分を image-cut-colorで指定したカラーで塗りつぶします。プレフィックス引数を指定した場合には、使用するカラーの入力を求めます。

サイズや回転は “繰り返す (repeat)” ことができるコマンドです。これはプレフィックスとして `i` を使うことなくイメージの調整を継続できることを意味します。

Emacs が ImageMagick のサポートつきでコンパイルされている場合、さまざまなイメージを描画するのに、ImageMagick を使うことができます。変数 `imagemagick-enabled-types`は、Emacs が ImageMagick を使って描画できるイメージの種類のリストです。リストの各要素は、ImageMagick 内部でのイメージ種類にたいする名前で、シンボルまたは等価な文字列で指定します (たとえば `BMP`は`.bmp`イメージです)。利用可能なイメージの種類にたいして ImageMagick を有効にするには、`imagemagick-enabled-types`を `t`に変更します。変数 `imagemagick-types-inhibit`は、変数 `imagemagick-enabled-types`の値にかかわらず mImageMagick を使わずに描画するイメージ種類のリストです (リストのデフォルトには `C`および `HTML`などが含まれ、これらは ImageMagick はイメージとして描画できるが、Emacs はイメージとして描画しないものです)。ImageMagick を完全に無効にするには、`imagemagick-types-inhibit`を `t`に変更してください。

要求されるイメージフォーマットにたいするネイティブサポートを Emacs がもたず、かつ `image-use-external-converter`が非 `nil`なら、Emacs は表示の前に要求されるイメージを PNGに変換するために使用可能な外部ユーティリティーの存在有無の判定を試みます。このイメージ変換には現在のところ GraphicsMagick、ImageMagick、`ffmpeg`がサポートされています。

加えて、特定のイメージフォーマットにたいして特別なハンドラーを追加したい場合もあるかもしれません。これらのハンドラーは `image-converter-add-handler`関数によって追加できます。たとえば Krita ファイルを単純なイメージとして閲覧したければ、以下のように記述することができます:

```
(image-converter-add-handler
 "kra"
 (lambda (file data-p)
   (if data-p
       (error "Can't decode non-files")
     (call-process "unzip" nil t nil
                   "-qq" "-c" "-x" file "mergedimage.png"))))
```

この関数は 2 つのパラメーターを受け取ります。1 つ目はファイル名のサフィックス、2 つ目は “変換” を行う関数です。変換を行う関数が受け取るパラメーターも 2 つで、1 つ目はファイル名かデータが格納されている文字列、2 つ目は 1 つ目のパラメーターがデータかどうかを示すパラメーターです。またこの変換関数はフォーマット `image-convert-to-format`でイメージをカレントバッファーに出力する必要があります。

Image-Dired パッケージはイメージをサムネールとして表示するのにも使用されます。Section 27.18 [Image-Dired], page 398 を参照してください。

15.20 ファイルセット

定期的に特定のファイルのグループを編集する場合、それらをファイルセット *(fileset)* として定義できます。これにより visit、`query-replace`、シェルコマンドなどの特別な操作を、すべてのファイルに一度に行うことができます。ファイルセットを使うには、init ファイル (Section 33.4 [Init File], page 528 を参照) に、式 `(filesets-init)`を追加しなければなりません。これはメニューバーの ‘`File`’メニューに、サブメニュー ‘`Filesets`’を追加します。

ファイルセットを定義する一番簡単な方法は、ファイルを 1 つずつ追加する方法です。ファイルセット *name*にファイルを追加するには、そのファイルを visit して `M-x filesets-add-buffer RET` *`name`* `RET`をタイプします。ファイルセット *name*が存在しない場合、現在のファイルだけを含む新し

いファイルセットを作成します。コマンド `M-x filesets-remove-buffer`は、ファイルセットから現在のファイルを削除します。

`M-x filesets-edit`(または‘`Filesets`’メニューの‘`Edit Filesets`’を選択) で、ファイルセットを直接編集することもできます。編集は Customize バッファーで行われます (Section 33.1 [Easy Customization], page 499 を参照してください)。ファイルセットは通常、単純なファイルのリストですが、ファイル名にマッチする正規表現で、ファイルセットを定義することもできます。より複雑なファイルセットの例は、Customize バッファーに記されています。将来の Emacs セッションで同じファイルセットを使うには、‘`Save for future sessions`’を選択するのを忘れないでください。

コマンド `M-x filesets-open`を使って、ファイルセットのすべてのファイルを visit し、それらを `M-x filesets-close`で閉じることができます。`M-x filesets-run-cmd`を使って、ファイルセットのすべてのファイルにたいして、シェルコマンドを実行します。これらのコマンドは‘`Filesets`’メニューからも利用可能で、メニューには既存のファイルセットが、サブメニューとして表示されています。

異なるコンセプトのファイルセット (バージョンコントロール操作のためにグループにまとめられたファイル) については、Section 25.1 [Version Control], page 332 を参照してください。この種のファイルセットには名前がなく、Emacs セッション間で引き継がれません。

16 複数バッファーの使用

Emacs で編集するテキストは、バッファー *(buffer)* と呼ばれるオブジェクトの中に存在します。ファイルを visit するたびに、そのファイルのテキストを保持するために、バッファーが使われます。Dired を呼び出すたびに、ディレクトリーリストを保持するためにバッファーが使われます。`C-x m`でメッセージを送信すると、メッセージのテキストを保持するためにバッファーが使われます。コマンドのドキュメントは、`*Help*`という名前のバッファーに表示されます。

バッファーは使用されているかぎりは存在して、ユーザー (Section 16.4 [Kill Buffer], page 179 を参照) または Emacs(たとえば Emacs の exit 時。Section 3.2 [Exiting], page 16 を参照) がもはや必要としなくなったときに削除 (または “kill”) されます。

それぞれのバッファーは、任意の長さの一意な名前を持っています。バッファーがウインドウに表示されているとき、バッファーの名前はモードライン (Section 1.3 [Mode Line], page 9 を参照してください) に表示されます。バッファー名での大文字と小文字の違いは重要です。ほとんどの場合、ほとんどのバッファーは visit しているファイルから作られ、それらの名前はファイル名から生成されます。しかし、新しい空のバッファーを、任意の名前で作成することもできます。新しく開始された Emacs にはいくつかのバッファーがあり、それらの中には`*scratch*`という名前の、Lisp 式を評価するのに使用されるバッファーも含まれます。そのバッファーはファイルに関連付けられていません (Section 24.10 [Lisp Interaction], page 330 を参照してください)。

選択されるバッファーは、常に 1 つだけです。そのバッファーをカレントバッファー *(current buffer:* 現在のバッファー*)* と呼びます。「コマンドは “そのバッファー (the buffer)” を操作します」という言い方をするときがあります。これはカレントバッファーを操作するというのが、本当の意味です。Emacs のウィンドウが 1 つだけのとき、そのウィンドウに表示されているバッファーがカレントになります。複数のウィンドウがあるとき、選択されたウィンドウに表示されているバッファーがカレントになります。Chapter 17 [Windows], page 186 を参照してください。

バッファーのコンテンツ *(contents:* 内容*)* とは、オプションでテキストプロパティ (Section 19.1 [International Chars], page 216 を参照) のセットをもつ一連の文字から構成されます。テキストプロパティにより、文字により多くの情報を指定できます。

バッファーのテキスト的な内容は別として、それぞれのバッファーはいくつかの情報を記録しています。それらは、(もしあれば)visit しているファイルは何か、変更されているか、有効なメジャーモードとマイナーモードは何か (Chapter 20 [Modes], page 240 を参照してください)、などの情報です。これらは、バッファーローカルな変数 *(buffer-local variables)* に格納され、これらの変数はバッファーごとに異なる値をもつことができます。Section 33.2.3 [Locals], page 511 を参照してください。

バッファーのサイズは、いくつかの最大値を超えて大きくすることはできません。これは一番大きいバッファーの位置が、*Emacs* の整数 *(Emacs integers)* で表されることにより定義されます。なぜなら Emacs はそのデータ型を使用して、バッファーの位置を追跡するからです。通常の 64 ビットマシンでは、バッファーの最大サイズは $2^{61} - 2$ バイト、およそ 2EiB です。通常の 32 ビットマシンでは、バッファーの最大サイズは通常 $2^{29} - 2$ バイト、およそ 512MiB です。バッファーのサイズはシステムのメモリー量によっても制限されます。

16.1 バッファーの作成と選択

`C-x b` *buffer* `RET`
: *buffer*という名前のバッファーを、選択または作成します (`switch-to-buffer`)。

`C-x 4 b` *buffer* `RET`
: 同様ですが、他のウィンドウで *buffer*を選択します (`switch-to-buffer-other-window`)。

`C-x 5 b` *buffer* `RET`
: 同様ですが、別のフレームで *buffer*を選択します (`switch-to-buffer-other-frame`)。

`C-x LEFT`
: バッファーリストの、前のバッファーを選択します (`previous-buffer`)。

`C-x RIGHT`
: バッファーリストの、次のバッファーを選択します (`next-buffer`)。

`C-u M-g M-g`
`C-u M-g g`
: 数字 *n*を読み取って、カレントバッファーではない、一番最近選択された、別のウィンドウにあるバッファーの、*n*行目に移動します。

`C-x b` (`switch-to-buffer`) コマンドは、ミニバッファーを使ってバッファー名を読み取ります。それからそのバッファーをカレントとして、現在選択されたウィンドウに表示します。空の入力は、そのとき他のウィンドウに表示されていない、一番最近カレントだったバッファーを指定します。

バッファー名を入力するとき、通常の補完とヒストリーコマンドを使うことができます (Chapter 5 [Minibuffer], page 28 を参照してください)。`C-x b`および関連するコマンドは、ミニバッファーの補完に、確認つきの寛大な補完 *(permissive completion with confirmation)* を使うことに注意してください。存在しないバッファー名にたいする補完の後、すぐに `RET`をタイプすると、Emacs は‘`[Confirm]`’を出力し、バッファー名を確定するために 2 回目の `RET`をタイプしなければなりません。詳細は、Section 5.4.3 [Completion Exit], page 33 を参照してください。その他の補完オプションと機能についての詳細は、Section 5.4.5 [Completion Options], page 35 を参照してください。

存在しないバッファーを指定すると、`C-x b`はファイルを visit していない新しい空のバッファーを作成し、編集用にそのバッファーを選択します。変数 `major-mode`は、新しいバッファーのメジャーモードのデフォルトを決定します。デフォルトでは、これは Fundamental モードです。Section 20.1 [Major Modes], page 240 を参照してください。新しいバッファーを作る 1 つの理由は、それを一時的なノートとして使うためです。これを保存しようと試みると、Emacs は保存に使うファイル名を尋ね、バッファーのメジャーモードは、ファイル名にもとづきメジャーモードを再割り当てします (Section 20.3 [Choosing Modes], page 244 を参照してください)。

少ないバッファーの切り替えでは、コマンド `C-x LEFT`および `C-x RIGHT`を使うのが便利です。`C-x LEFT` (`previous-buffer`) は前のバッファー (現在のフレームの一番最近選択されたバッファー順)、`C-x RIGHT` (`next-buffer`) は逆方向のバッファーに移動します。いずれのコマンドもプレフィクス数引数をサポートており、それは繰り返し回数として機能します。

現在のウィンドウとは別のウィンドウにバッファーを選択するには、`C-x 4 b` (`switch-to-buffer-other-window`) とタイプします。これはミニバッファーを使ってバッファー名の入力を求め、選択されているウィンドウではない別のウィンドウに、そのバッファーを表示して、そのウィンドウを選択します。

同様に `C-x 5 b` (`switch-to-buffer-other-frame`) はバッファー名の入力を求め、他のフレーム (Chapter 18 [Frames], page 194 を参照) にそのバッファーを表示して、そのフレームを選択します。他のウィンドウまたはフレームにすでにバッファーが表示されている場合、Emacs は新しく作成するのではなく、そのウィンドウまたはフレームを選択します。

`C-x 4 b`および `C-x 5 b`コマンドが表示するウィンドウまたはフレームを取得する方法については、Section 17.6 [Displaying Buffers], page 190 を参照してください。

これらに加えて、C-x C-fや、その他のファイルを visit するコマンドでも、すでにファイルを visit しているバッファーに切り替えることができます。Section 15.2 [Visiting], page 146 を参照してください。

プレフィクス引数だけを指定した C-u M-g M-g goto-lineは、ミニバッファーを使って数字 nを読み取り、別のウィンドウにあるカレントバッファー以外の、一番最近選択されたバッファーを選択して、ポイントをそのバッファーの n行目の先頭に移動します。これは他のバッファーの行番号を参照するバッファーで特に有用です。ポイントが数字の直後にある場合、goto-lineは nのデフォルトとしてその数字を使います。ただの C-uではないプレフィクス引数では、異なる振る舞いをすることに注意してください。C-u 4 M-g M-gはミニバッファーから数字を読み取らず、カレントバッファーの 4 行目にジャンプします (プレフィクス引数を指定しない M-g M-gは、数字 nを読み取ってカレントバッファーの n行目に移動することを思い出してください。Section 4.2 [Moving Point], page 18 を参照してください)。

Emacs はスペースで開始される名前のバッファーを、内部的な用途のために使用しています。それらのバッファーは特別な方法で扱われます。たとえば、それらのバッファーでは undo 情報が記録されません。そのようなバッファー名を使うことは避けるのが最良です。

16.2 既存のバッファーを一覧する

C-x C-b　既存のバッファーをリストします (list-buffers)。

既存のバッファーのリストを表示するには、C-x C-bとタイプします。これにより、*Buffer List*という名前のバッファーにバッファーメニューがポップアップします。リストの各行にはバッファー名、サイズ、メジャーモード、visit しているファイルが表示されます。バッファーは、カレントだった順でリストされます。したがって、一番最近カレントだったバッファーが先頭に表示されます。このセクションでは、バッファーリストが表示される方法と、リスト内に示されるさまざまなものの解釈の仕方について説明します。*Buffer List*バッファーの特別なモードと、利用可能なコマンドについては、Section 16.5 [Several Buffers], page 180 を参照してください。

行の最初のフィールドの '.'は、そのバッファーがカレントであることを示します。'%'は読み取り専用バッファーであることを示します。'*'はそのバッファーが変更されていることを示します。いくつかのバッファーが変更されていて、それらを保存するべきだと思ったときは、C-x s (Section 15.3.1 [Save Commands], page 150 を参照してください) で保存します。以下はバッファーリストの例です:

```
CRM Buffer                Size  Mode              File
. * .emacs                 3294  Emacs-Lisp        ~/.emacs
 %  *Help*                  101  Help
    search.c               86055  C                 ~/cvs/emacs/src/search.c
 %  src                    20959  Dired by name     ~/cvs/emacs/src/
  * *mail*                   42  Mail
 %  HELLO                  1607  Fundamental       ~/cvs/emacs/etc/HELLO
 %  NEWS                 481184  Outline           ~/cvs/emacs/etc/NEWS
    *scratch*                191  Lisp Interaction
  * *Messages*              1554  Messages
```

この例でバッファー*Help*は、ヘルプを要求されたことにより作成されます (Chapter 7 [Help], page 42 を参照してください)。これはファイルを visit していません。バッファー srcは、ディレクトリー~/cvs/emacs/src/にたいして、Dired が作成したバッファーです。ファイルを visit しているバッファーだけを一覧するには、C-u C-x C-bのように、コマンドにプレフィクス引数を与えます。

list-buffersは名前がスペースで始まるバッファーを省略します (そのバッファーがファイルを visit していない限り)。これらのバッファーは、Emacs により内部的に使用されます。

16.3 その他のバッファー操作

`C-x C-q` バッファーの読み取り専用の状態を切り替えます (`read-only-mode`)。

`C-x x r RET` *buffer* `RET`
カレントバッファーの名前を変更します。

`C-x x u` カレントバッファーの末尾に '`<`*number*`>`'を加えてリネームします。

`M-x view-buffer RET` *buffer* `RET`
バッファー *buffer*をスクロールして閲覧します。Section 11.6 [View Mode], page 82 を参照してください。

バッファーを読み取り専用にすることができます、これはバッファーのテキストにたいして挿入や削除を行うコマンドが許されないことを意味します (とはいえ、他の `C-x RET f`のようなコマンドは、バッファーを変更済みとマークすることは可能。Section 19.9 [Text Coding], page 227 を参照されたい)。読み取り専用バッファーのモードラインでは、左余白の近くに '`%%`'または '`%*`'が表示されます。Section 1.3 [Mode Line], page 9 を参照してください。読み取り専用バッファーは通常、Dired や Rmail のように、そのバッファーのテキストを操作する専用コマンドをもつ、サブシステムにより作成されます。アクセスが制御されているファイルを visit したときも、そのバッファーに書き込めない旨が通知されます。

コマンド `C-x C-q` (`read-only-mode`) は、読み取り専用バッファーを書き込み可能に、書き込み可能なバッファーを読み取り専用にします。これは各バッファーにローカルな変数 `buffer-read-only`をセットすることにより機能します (値が非 `nil`のときバッファーは読み取り専用)。オプション `view-read-only`を非 `nil`値に変更すると、`C-x C-q`でバッファーを読み取り専用にしたとき、そのバッファーで View モードが有効になります (Section 11.6 [View Mode], page 82 を参照してください)。

`C-x x r` (`rename-buffer`) はカレントバッファーの名前をリネームします。新しい名前はミニバッファーで指定します。デフォルトはありません。他のバッファーで使用済の名前を指定するとエラーとなり、リネームされません。

`C-x x u` (`rename-uniquely`) はカレントバッファーの後に数字を追加して、似てはいるが異なる名前を作成します。このコマンドに引数は必要ありません。これは複数の shell バッファーを作成するのに便利です。`*shell*`バッファーをリネームしてから再度 `M-x shell`を行うと、これは新しく `*shell*`という名前のバッファーを作成します。一方リネームされた古い shell バッファーは新しい名前で実行を続けます。この方法は mail バッファー、compilation バッファー、そして特定の名前で特別なバッファーを作成する Emacs 機能に適しています (これらの機能のいくつか、たとえば `M-x compile`、`M-x grep`などではコマンドを再度実行する前に他のバッファーに切り替える必要がある。さもないとカレントバッファーの名前を変更せずに、そのバッファーを再使用する)。

コマンド `M-x append-to-buffer`と `C-x x i` (`insert-buffer`) も、あるバッファーから他のバッファーへテキストをコピーするのに使用できます。Section 9.4 [Accumulating Text], page 67 を参照してください。

16.4 バッファーの kill

しばらくの間、Emacs セッションを続けていると、多くのバッファーが溜まってしまう場合があります。必要のないバッファーを *kill* したほうがよいと思うかもしれません (他のエディターではこの操作を *close*(閉じる) と呼び、"バッファーを閉じる" とか、ファイルを visit しているバッファーでは "ファイルを閉じる" のように表現するものもあります)。ほとんどのオペレーティングシステムでは、バッファーを kill することにより、バッファーのために Emacs が使用していたメモリーをオペレー

ティングシステムに解放するので、他のプログラムがそれを使えるようになります。以下はバッファーを kill するコマンドです:

`C-x k` *buffer* `RET`
: バッファー *buffer*を kill します (`kill-buffer`)。

`M-x kill-some-buffers`
: バッファーを 1 つずつ kill するか尋ねます。

`M-x kill-matching-buffers`
: 正規表現にマッチするすべてのバッファーを kill するか尋ねます。

`C-x k` (`kill-buffer`) は、ミニバッファーで指定した名前のバッファーを 1 つ kill します。`RET` だけをタイプしたときはデフォルトが使われ、これはカレントバッファーを kill します。カレントバッファーを kill すると、最近カレントだったが、今は別のウィンドウにも表示されていないバッファーがカレントになります。ファイルを visit しているバッファーが変更されているとき、それを kill しようとすると確認を求められます。そのバッファーを kill する前に、確認に `yes`を応えなければ kill できません。

コマンド `M-x kill-some-buffers`は、各バッファーについて 1 つずつ確認を求めます。`yes`と応えると、`kill-buffer`と同様にバッファーを kill することを意味します。このコマンドは名前がスペースで始まる、Emacs が内部的に使用するバッファーは無視します。

コマンド `M-x kill-matching-buffers`は、正規表現の入力を求め、名前がその正規表現にマッチするすべてのバッファーを kill します。Section 12.6 [Regexps], page 115 を参照してください。 `kill-some-buffers`と同様、このコマンドは kill する前に確認を求めます。このコマンドは通常、名前がスペースで始まる、Emacs が内部的に使うバッファーを無視します。内部的なバッファーも同じように kill するには、プレフィクス引数を指定して `kill-matching-buffers`を呼び出します。

さまざまなバッファーを kill するには Buffer Menu 機能も便利です。Section 16.5 [Several Buffers], page 180 を参照してください。

バッファーが kill されるときに特別な処理を行いたい場合、フック `kill-buffer-hook`にフック関数を追加できます (Section 33.2.2 [Hooks], page 509 を参照してください)。

(多くの人がそうしているように) 何日間も 1 つの Emacs セッションを使っていると、何日か前に使ったバッファーが溜まってくるかもしれません。コマンド `M-x clean-buffer-list`は、それらを一掃するのに便利です。これは長い間使用されていない、変更されていないバッファーをすべて kill します。3 日間表示されていない普通のバッファーは kill されます。しかし自動的に kill されるべきではない特定のバッファーを指定したり、使われていない時間が短いバッファーでも kill するように指定できます。これらのデフォルト、およびこのコマンドの挙動の他の動向は、`clean-buffer-list` のドキュメント文字列で説明されている、いくつかのオプションをカスタマイズすることにより制御できます。

Midnight モードを有効にすることにより、1 日ごとにバッファーを一掃することもできます。Midnight モードは毎日真夜中に `clean-buffer-list`、またはノーマルフック `midnight-hook`に指定された関数を実行します (Section 33.2.2 [Hooks], page 509 を参照してください)。Midnight モードを有効にするには、Customization バッファーを使って変数 `midnight-mode`を `t`にセットします。Section 33.1 [Easy Customization], page 499 を参照してください。

16.5 複数バッファーにたいする操作

`M-x buffer-menu`
: すべての Emacs バッファーの、バッファーリストの編集を開始します。

`M-x buffer-menu-other-window`
: 同様ですが、別のウィンドウで行います。

`C-x C-b` (Section 16.2 [List Buffers], page 178 を参照してください) でオープンされる *Buffer Menu* は、単にバッファーを一覧するだけではありません。これは Dired に似たインターフェースで、バッファーにたいしてさまざまな操作を行うことができます。ここでは、バッファーの保存、kill(Dired との一貫性を保つため、ここではそれらを削除 *(delete)* する、と呼びます)、表示ができます。

Buffer Menu を使うには `C-x C-b`とタイプして、`*Buffer List*`バッファーが表示されたウィンドウに切り替えます。`M-x buffer-menu`とタイプして、選択されたウィンドウに Buffer Menu を開くこともできます。コマンド `M-x buffer-menu-other-window`は、Buffer Menu を別のウィンドウに開いて、そのウィンドウを選択します。

Buffer Menu は読み取り専用バッファーで、このセクションで説明する特別なコマンドだけを通じて変更します。このバッファーでは、通常のカーソル移動コマンドを使うことができます。以下のコマンドは、バッファーのカレント行に適用されます。

`d`
: そのバッファーの削除 (kill) フラグをセットしてから、ポイントを次の行に移動します (`Buffer-menu-delete`)。削除フラグは、各行のバッファー名の前の文字 '`D`'により示されます。削除は `x`コマンド (以下参照) をタイプしたときだけ発生します。

`C-d`
: `d`と同様ですが、ポイントを下ではなく上に移動します (`Buffer-menu-delete-backwards`)。

`s`
: そのバッファーの保存フラグをセットします (`Buffer-menu-save`)。保存フラグは各行のバッファー名の前の文字 '`S`'により示されます。保存は `x`コマンド (以下参照) をタイプしたときだけ発生します。同じバッファーに保存と削除の両方をリクエストできます。

`x`
: すべての削除および保存フラグを処理します (`Buffer-menu-execute`)。

`u`
: カレント行のすべてのフラグを取り除いてから、下に移動します (`Buffer-menu-unmark`)。プレフィックス引数を指定した場合は、フラグを取り除いた後、上に移動します。

`DEL`
: 前の行に移動して、その行のすべてのフラグを取り除きます (`Buffer-menu-backup-unmark`)。

`M-DEL`
: すべて行から特定のフラグを取り除きます (`Buffer-menu-unmark-all-buffers`)。これは、1 文字の入力を求めて、その文字でマークされたバッファーのマークを取り除きます。`RET`をタイプした場合は、すべてのマークを取り除きます。

`U`
: すべての行のすべてのフラグを取り除いてから、下に移動します (`Buffer-menu-unmark-all`)。

フラグを取り除くコマンド `d`と `C-d`には、繰り返し回数として数引数を指定できます。

以下のコマンドは、カレント行にリストされたバッファーを即座に処理します。これらのコマンドにも、繰り返し回数を数引数として指定できます。

`~`
: バッファーを変更されていない (unmodified) とマークします (`Buffer-menu-not-modified`)。Section 15.3.1 [Save Commands], page 150 を参照してください。

`%`
: バッファーの読み取り専用属性を切り替えます (`Buffer-menu-toggle-read-only`)。Section 16.3 [Misc Buffer], page 179 を参照してください。

`t`
: そのバッファーを、tags テーブルとして visit します (`Buffer-menu-visit-tags-table`)。Section 25.4.3 [Select Tags Table], page 368 を参照してください。

以下は、他のバッファーを選択するために使われるコマンドです:

`q` Buffer Menu メニューを閉じます (`quit-window`)。一番最近に表示されていたバッファーが、その場所に表示されます。

`RET`
`f` そのウィンドウの`*Buffer List*`バッファーを置き換えて。この行のバッファーを選択します (`Buffer-menu-this-window`)。

`o` `C-x 4 b`のように`*Buffer List*`を表示したまま、その行のバッファーを他のウィンドウで選択します (`Buffer-menu-other-window`)。

`C-o` この行のバッファーを他のウィンドウで表示しますが、選択はしません (`Buffer-menu-switch-other-window`)。

`1` この行のバッファーを、フレーム全体のウィンドウで選択します (`Buffer-menu-1-window`)。

`2` 現在のフレームを 2 つのウィンドウにセットアップして、この行のバッファーを一方のウィンドウで選択し、もう一方のウィンドウに以前のカレントバッファー (`*Buffer List*` は除く) を表示します (`Buffer-menu-2-window`)。

`b` この行のバッファーを bury(つまりバッファーリストの最後) に移動します (`Buffer-menu-bury`)。

`m` `v`コマンドで抜けるとき、この行のバッファーを他のウィンドウで表示されるようにマークします (`Buffer-menu-mark`)。表示フラグは行の先頭に文字 '`>`'で示されます (1 つのバッファーが削除と表示のフラグを両方もつことはないでしょう)。

`v` この行のバッファーを選択し、`m`コマンドでフラグづけされたバッファーも他のウィンドウで表示します (`Buffer-menu-select`)。バッファーにフラグをつけていないとき、このコマンドは `1`と等価です。

以下はバッファーリスト全体に影響を与えるコマンドです:

`S` ポイントがある列の数値にしたがって、Buffer Menu 全体をソートします。数引数 *n*を指定すると、*n*番目の列でソートします (`tabulated-list-sort`)。

`}` カレント列の幅を *n*文字 (プレフィクス引数) 広くします。

`{` カレント列の幅を *n*文字 (プレフィクス引数) 狭くします。

`T` 非ファイルバッファーを削除または再挿入します (`Buffer-menu-toggle-files-only`)。このコマンドは、そのようなバッファーがバッファーリストに含まれるか否かを切り替えます。

バッファー`*Buffer List*`は通常、バッファーの作成および kill により自動的に更新はされません (内容は単なるテキストです)。バッファーを作成・削除・リネームした場合、それが行われたか確かめるために`*Buffer List*`を更新するには、`g` (`revert-buffer`) とタイプします。このバッファーの Auto Revert モードを有効にすれば、このバッファーが変更されたとマークされない限り、`auto-revert-interval`秒ごとに定期的に更新させることができます。Global Auto Revert モードは、`global-auto-revert-non-file-buffers`が非`nil`のときだけ、`*Buffer List*`バッファーに適用されます。

16.6 インダイレクトバッファー

インダイレクトバッファー *(indirect buffer:* 間接バッファー*)* は、そのインダイレクトバッファーのベースバッファー *(base buffer:* 基底バッファー*)* と呼ばれる、他のバッファーのテキストを共有します。ある点においては、ファイルに置けるシンボリックリンクの、バッファー版に例えることができます。

`M-x make-indirect-buffer RET` *base-buffer* `RET` *indirect-name* `RET`
: ベースバッファー *base-buffer*の、インダイレクトバッファー *indirect-name*を作成します。

`M-x clone-indirect-buffer RET`
: カレントバッファーの、インダイレクトバッファーを作成します。

`C-x 4 c`
: カレントバッファーのインダイレクトバッファーを作成して、それを別のウィンドウで選択します (`clone-indirect-buffer-other-window`)。

インダイレクトバッファーのテキストは、常にベースバッファーのテキストと等しく、どちらかを編集すると、その変更はすぐにもう一方から見えるようになります。ここで言う “テキスト” にはテキストの文字とそれらのテキストプロパティの両方が含まれます。しかし他の観点では、インダイレクトバッファーとベースバッファーは、完全に分離されています。これらのバッファーは異なる名前、異なるポイント値、異なるナローイング、異なるマーカー、異なるオーバーレイ、異なるメジャーモード、異なるローカル変数をもつことができます。

インダイレクトバッファーはファイルを visit できませんが、ベースバッファーは visit できます。インダイレクトバッファーの保存を試みると、それは実際にはベースバッファーの保存として機能します。ベースバッファーを kill するとインダイレクトバッファーも kill されますが、インダイレクトバッファーの kill は、ベースバッファーに影響を与えません。

インダイレクトバッファーの 1 つの使い方としては、アウトラインの複数の視点からの表示です。Section 22.9.5 [Outline Views], page 265 を参照してください。

手早くインダイレクトバッファーを作成するには、コマンド `C-x 4 c` (`clone-indirect-buffer-other-window`) を使う方法があります。これはカレントバッファーをベースバッファーとする、インダイレクトバッファーを作成して選択します。数引数を指定すると、インダイレクトバッファーの名前の入力を求めます。指定しない場合、カレントバッファー名の後ろに ‘`<n>`’を付加した名前を使います。

インダイレクトバッファーを作成する、より一般的な方法はコマンド `M-x make-indirect-buffer` です。これはバッファー *base-buffer*から、名前が *indirect-name*のインダイレクトバッファーを作成します。これらの名前は、ミニバッファーを使って入力が求められます。

インダイレクトバッファーを作成した関数はインダイレクトバッファーの作成後にフック `clone-indirect-buffer-hook`を実行します。このフックの実行時には、作成されたインダイレクトバッファーがカレントバッファーになります。

注意: バッファーのテキストにたいして変更が行われた際には、変更フックはベースバッファーでのみ実行されます。なぜならこれらのフックに登録されている関数のほとんどは、インダイレクトバッファーで正しく動作するように想定されていないからです。したがってインダイレクトバッファーで変更フック関数が必要なら、ベースバッファーにおいて手動でその関数を追加してから、その関数に目的であるインダイレクトバッファーを処理させる必要があります。

16.7 バッファー処理の便利な機能とカスタマイズ

このセクションではバッファーを切り替えをもっと便利にする、モードと機能をいくつか説明します。

16.7.1 バッファー名を一意にする

同じ名前のファイルを複数のバッファーで visit しているとき、Emacs はバッファーに区別可能な名前をつけなければなりません。デフォルトの方法はファイルをディレクトリーの一部を後ろに追加する方法です。たとえば同時にファイル`/foo/bar/mumble/name`と`/baz/quux/mumble/name`を visit している場合、バッファーの名前は‘`name<bar/mumble>`’と‘`name<quux/mumble>`’になるでしょう。Emacs は名前を一意にするために必要な分だけ、ディレクトリー名の一部を追加します。

オプション`uniquify-buffer-name-style`をカスタマイズして、一意なバッファー名を構築するための異なるスタイルを選択することができます。

命名方法`forward`は、バッファー名の先頭部分に、ファイルのディレクトリー名の一部を含めます。この方法を使うとファイル`/u/rms/tmp/Makefile`と`/usr/projects/zaphod/Makefile`を visit しているバッファーの名前は、‘`tmp/Makefile`’と‘`zaphod/Makefile`’になるでしょう。

対照的に命名方法`post-forward`は、バッファーを‘`Makefile|tmp`’および‘`Makefile|zaphod`’と名づけます。デフォルトの`post-forward-angle-brackets`は、`post-forward`と同様ですが、一意なパスを山形カッコ (angle brackets) で囲います。命名方法`reverse`では、‘`Makefile\tmp`’および‘`Makefile\zaphod`’となります。`post-forward`と`reverse`の重要な違いは、2 つのファイルを区別するのに 1 つのディレクトリ名では不足するときです。`reverse`はディレクトリー名を逆順にして命名するので、ファイル`/top/middle/file`は‘`file\middle\top`’となります。一方`post-forward`はディレクトリー名を正順でファイル名に追加するので‘`file|top/middle`’となります。`uniquify-buffer-name-style`が`nil`にセットされていると、バッファー名の後ろに単に‘`<2>`’、‘`<3>`’と加えることにより、バッファー名を作成します。

`uniquify-buffer-name-style`の値には *base*と *extra-strings*という 2 つの引数を受け取るカスタマイズされた関数にセットできます。ここで *base*は文字列、*extra-strings*は文字列のリストです。たとえば`post-forward-angle-brackets`にたいする現実装は以下のようになります:

```
(defun my-post-forward-angle-brackets (base extra-string)
  (concat base \"<\" (mapconcat #'identity extra-string \"/\") \">\"))
```

タイプする前にバッファー名に注意すれば、バッファー名にディレクトリー名を付加する方法に重要な違いはありません。しかしルールを知った経験豊富なユーザーは、それほど注意する必要はないでしょう。そして、その場合いずれかのルールでバッファー名が表示されていれば，思い出し利用するのが容易になることでしょう。

16.7.2 素早いミニバッファーの選択

グローバルなマイナーモードの Icomplete モードは、ミニバッファーで利用可能な補完候補を素早く選択する便利な方法を提供します。これが有効な場合、ミニバッファーでタイプすることにより、すでにタイプした文字にマッチする、利用可能な候補を連続で表示します。

`C-j`とタイプすることにより、常にリストの最初の補完候補を選択できます。つまり特定の補完候補を選択するには、まずその候補をリストの先頭にすることです。これを行うには 2 つの方法があります。1 つ目の方法は、補完候補の名前をさらにタイプすることにより、違う候補が含まれないように、希望する補完候補にリストを絞りこむ方法です。もう 1 つは`C-.`および`C-,`を使って、望む候補が先頭にくるまでリストをローテートする方法です。

`M-TAB`は`C-j`のようにリストの最初の補完候補を選択しますが、ミニバッファーを抜けないので、さらに編集を続けることができます。これはファイル名の入力で通常使われます。`M-TAB`により、いくつかのディレクトリー階層をたどることができます。

Icomplete モードを有効にするには、`M-x icomplete-mode`とタイプするか、変数`icomplete-mode`を`t`にカスタマイズします (Section 33.1 [Easy Customization], page 499 を参照してください)。

Fido モードは Icomplete の代替えとなるモードです。このモードは Icomplete モードと非常に似ていますが、Ido モードと呼ばれるポピュラーな拡張の機能のいくつかを残しています (実際のところこのモード名は “Fake Ido” から継承された)。特に Fido モードでは `C-s`と `C-r`を補完リストのローテートにも使用し、`C-k`はリスト内のファイル削除やバッファーの kill に使用できます。他に特筆すべき観点はデフォルトの保管スタイルとして `flex`を使用する点です (Section 5.4.4 [Completion Styles], page 34 を参照)。これを変更するには初期化ファイル (Section 33.4 [Init File], page 528 を参照) に以下を追加してください:

```
(defun my-icomplete-styles ()
  (setq-local completion-styles '(initials flex)))
(add-hook 'icomplete-minibuffer-setup-hook 'my-icomplete-styles)
```

Fido モードを有効にするには `M-x fido-mode`とタイプするか、変数 `fido-mode`を `t`にカスタマイズしてください (Section 33.1 [Easy Customization], page 499 を参照)。

Icomplete モードと Fido モードは、デフォルトでは同一行上の可能な補完をプロンプトとして表示します。補完候補をプロンプトの下に垂直に表示するには `M-x icomplete-vertical-mode`とタイプするか、あるいは変数 `icomplete-vertical-mode`を `t`にカスタマイズしてください (Section 33.1 [Easy Customization], page 499 を参照)。

16.7.3 バッファーメニューのカスタマイズ

`M-x bs-show`
: `M-x list-buffers`と同様にバッファーリストを作成しますが、カスタマイズが可能です。

`M-x ibuffer`
: バッファーのリストを作成して、Dired 様式でそれらを操作できるようにします。

`M-x bs-show`は、通常 `C-x C-b`で表示されるのと同じようなバッファーリストを表示しますが、これはより柔軟な方法で表示をカスタマイズできます。たとえば、表示すべきバッファー属性のリスト、バッファー名の列幅の最小と最大、決して表示されないバッファーや常に表示するバッファー名にたいする正規表現を指定することができます。通常のバッファーリストよりこちらのほうが好ましい場合、このコマンドを `C-x C-b`にバインドできます。このバッファーリストをカスタマイズするには、Custom グループの `bs`(Section 33.1 [Easy Customization], page 499 を参照) を使用するか、`bs-customize`を呼び出してください。

MSB グローバルマイナーモード (“MSB” は “mouse select buffer” が由来) は、カスタマイズ可能なマウスによる別の Buffer Menu を、好みに応じて提供します。これは通常 `C-Down-mouse-1`、または `C-F10`にバインドされている `mouse-buffer-menu`と、そのコマンドを置き換えます。メニューは Custom グループ `msb`でカスタマイズが可能です。

IBuffer とは、バッファーのリストを閲覧して、フィルタリング、マーキング、さまざまな方法によるソート、およびバッファーにたいする処理を含む、Dired(Chapter 27 [Dired], page 380 を参照) に似た操作をバッファーに行うメジャーモードです。

17 複数ウィンドウ

Emacs のフレームは、2 つ以上のウィンドウに分割できます。複数のウィンドウで異なるバッファーを表示したり、1 つのバッファーの異なる部分を表示できます。複数フレームは自ずと複数ウィンドウになります。なぜならフレームには、それぞれウィンドウがあるからです。ウィンドウは 1 つのフレームだけに属します。

17.1 Emacs ウィンドウの概念

Emacs の各ウィンドウには、常に 1 つの Emacs バッファーが表示されます。1 つのバッファーは、複数のウィンドウに表示される場合があります。この場合、バッファーのテキストへの任意の変更は、それが表示されているすべてのウィンドウで表示されます。しかし各ウィンドウは独自にポイント値をもっているので、ウィンドウごとにバッファーの異なる部分を表示できます。

常に 1 つの Emacs ウィンドウが、選択されたウィンドウとなります。このウィンドウに表示されているバッファーが、カレントバッファーとなります。グラフィカルなディスプレイでは、選択されたウィンドウのカーソルは、点滅する塗りつぶされたカーソルとなり、選択されていないウィンドウでは、中抜きボックスのカーソルになります。テキスト端末では、カーソルは選択されたウィンドウだけで描画されます。Section 11.21 [Cursor Display], page 99 を参照してください。

ポイントを移動するコマンドは、選択された Emacs ウィンドウのポイント値だけに影響します。他の Emacs ウィンドウのポイント値は、たとえ同じバッファーを表示していたとしても変更されません。これと同じことは、`C-x b`のようなバッファー切り替えコマンドについても言えます。これらは他のウィンドウには影響を与えません。しかし `C-x 4 b`のような、別のウィンドウを選択して、バッファーを切り替えるコマンドもあります。(たとえば)`C-h f` (`describe-function`) や `C-x C-b` (`list-buffers`) を含む、ウィンドウで情報を表示するコマンドは、選択されたウィンドウに影響を与えることなく、通常は選択されていないウィンドウ内にバッファーを表示することにより機能します。

複数ウィンドウで同じバッファーを表示しているとき、これらは異なるリージョンを持ちます。なぜなら、それらは異なるポイント値を持つことができるからです。しかしバッファーごとにマーク位置は 1 つだけなので、これらは同じマーク位置をもちます。

それぞれのウィンドウには、独自のモードラインがあり、それにはバッファー名、変更状態、そのウィンドウに表示されているバッファーのメジャーモードとマイナーモードが表示されます。選択されたウィンドウのモードラインは、異なる色で表示されます。詳細については、Section 1.3 [Mode Line], page 9 を参照してください。

17.2 ウィンドウの分割

`C-x 2`　選択されたウィンドウを上下に 2 分割します (`split-window-below`)。

`C-x 3`　選択されたウィンドウを左右に 2 分割します (`split-window-right`)。

`C-mouse-2`
　ウィンドウのモードライン上では、ウィンドウを分割します。

`C-x 2` (`split-window-below`) は、選択されたウィンドウを上下に 2 つのウィンドウに分割します。分割した後は、上が選択されたウィンドウになり、新たに分割されたウィンドウが下になります。2 つのウィンドウは最初同じポイント値をもち、(可能な限り) 同じバッファー部分を表示します。必要ならポイントをスクリーン上に残したまま、ウィンドウをスクロールできます。デフォルトでは 2 つのウィンドウの高さは、元のウィンドウの高さの半分になります。正の数引数は上のウィンドウの高さが何行分かを指定し、負の数引数は下のウィンドウが何行分の高さかを指定します。

変数 split-window-keep-pointを nilに変更すると、C-x 2はスクリーンに表示されるテキストができるだけ前と同じになるように、ウィンドウに表示されるバッファー部分と、各ウィンドウのポイント位置を調整します。さらにポイントが元のウィンドウの下半分にあったときは、上ではなく下のウィンドウが選択されます。

C-x 3 (split-window-right) は、選択されたウィンドウを左右に 2 つのウィンドウに分割します。左のウィンドウが選択されたウィンドウとなり、右のウィンドウには同じバッファーの同じ部分が表示され、ポイント位置も同じです。正の数引数は左のウィンドウの幅を列数で指定し、負の数引数は右のウィンドウの幅を数引数で指定します。

ウィンドウを C-x 3で分割すると、分割されたウィンドウの幅はフレーム全体の幅より小さくなります。ウィンドウの幅が狭くなりすぎると、継続行が使われたバッファーを読むことが困難になる場合があります (Section 4.8 [Continuation Lines], page 23 を参照してください)。したがってウィンドウの幅が 50 列より狭くなったとき、Emacs は自動的に行を切り詰めに切り替えます。この切り詰めは、変数 truncate-linesの値に関係なく発生します (Section 11.22 [Line Truncation], page 100 を参照してください)。truncate-linesのかわりに、この自動切り詰めを制御するのは、変数 truncate-partial-width-windowsです。この変数の値が正の整数 (デフォルトは 50) の場合、それは自動的な行切り詰めが発生する前の、分割ウィンドウの最小のトータル幅 (total width) を指定します。この値が nilの場合、自動的な行切り詰めは無効です。他の非 nil値では、分割されたウィンドウの幅に関係なく、Emacs は行を切り詰めます。ウィンドウのトータル幅 (total width) とは、window-total-width (Section “Window Sizes” in *The Emacs Lisp Reference Manual* を参照してください) で報告される列単位の値で、これにはフリンジと、継続および切り詰めのためのグリフ、マージン、スクロールバーが含まれます。

テキスト端末では、左右に分割されたウィンドウを垂直に分割する分割線は、vertical-borderフェイスで描画されます。

ウィンドウのモードライン上で C-mouse-2をクリックすると、クリックした位置に垂直分割線を配してウィンドウを分割します。Emacs がコンパイルされた方法に依存しますが、ウィンドウのスクロールバー上で C-mouse-2をクリックすると、クリックした位置に水平分割線を配してウィンドウを分割します (この機能は、Emacs が GTK+のスクロールバーを使っているときは機能しません)。

デフォルトでは、ウィンドウを分割したとき、Emacs は分割されたウィンドウのサイズをフレームのデフォルトフォントの整数倍にします。これによりスクリーンが正確に 2 分割されない場合があります。変数 window-resize-pixelwiseを非 nil値にセットすると、Emacs は分割されたウィンドウのサイズを、同じピクセル数にします (元のサイズが奇数のピクセル数の場合、1 ピクセル増減されます)。フレームのピクセル数がフレームの文字サイズ倍でない場合、このオプションが nilでも、少なくとも 1 つのウィンドウがピクセル幅のサイズ変更をされることに注意してください。

17.3 他のウィンドウの使用

C-x o　他のウィンドウを選択します (other-window)。

C-M-v　次のウィンドウを上方にスクロールします (scroll-other-window)。

C-M-S-v　次のウィンドウを下方にスクロールします (scroll-other-window-down)。

C-M-S-l　次のウィンドウを再センタリングします (recenter-other-window)。

Mouse-1　ウィンドウのテキスト領域を mouse-1でクリックすると、そのウィンドウを選択してクリックした位置にポイントを移動します。モードラインをクリックすると、ポイントを移動せずにそのウィンドウを選択します。

キーボードで C-x o (other-window) をタイプして、ウィンドウを切り替えることができます。oは “other” の o で、0(ゼロ) ではありません。2 つ以上のウィンドウがある時、このコマンドはすべてのウィンドウを順繰りに選択します (一般的に上からした、左から右)。一番右または一番下のウィンドウの後は、左上のウィンドウに戻ります。数引数は上記の順番で何番目のウィンドウに移動するかを意味します。負の引数は逆向きで同じことを行います。ミニバッファーがアクティブのとき、ミニバッファーウィンドウはこの順番では最後のウィンドウになります。ミニバッファーのウィンドウから他のウィンドウに切り替えて、後からミニバッファーに戻って引数の入力を終了させることができます。Section 5.3 [Minibuffer Edit], page 30 を参照してください。

other-windowコマンドは通常は (別フレームが設定されていないかぎり) カレントフレームの次のウィンドウだけに切り替えます。マルチフレーム環境において、このサイクルの一部にすべてのフレームのウィンドウを含めたければ、C-x oを next-window-any-frameコマンドにリバインドすることができます (コマンドをリバインドする方法は Section 33.3.5 [Rebinding], page 521 を参照)。

通常のスクロールコマンド (Chapter 11 [Display], page 77 を参照) は選択されたウィンドウだけに適用されますが、次のウィンドウをスクロールするコマンドは他にもあります。C-M-v (scroll-other-window) は C-x oにより選択されるようなウィンドウをスクロールします。このコマンドは別の観点から見ると C-vのように振る舞います。いずれのコマンドもバッファーテキストをウィンドウから相対的に上方へ移動して、どちらも正と負の引数を受け取るからです (ミニバッファーではミニバッファーに関連付けられたヘルプウィンドウがあれば、C-M-vは標準的なサイクル順での次ウィンドウではなくヘルプウィンドウをスクロールする。Section 5.3 [Minibuffer Edit], page 30 を参照)。C-M-S-v (scroll-other-window-down) は同様の方法で次のウィンドウを下方にスクロールします。同様に C-M-S-l (recenter-other-window) は次ウィンドウにおいて C-l (recenter-top-bottom) のように振る舞います。

mouse-autoselect-windowを非 nil値にセットしている場合、マウスが選択されたウィンドウ以外のウィンドウに移動すると、そのウィンドウが選択されます。この機能はデフォルトでオフです。

17.4 他のウィンドウでの表示

C-x 4は、異なるウィンドウ (他の既存のウィンドウや、選択されたウィンドウを分割することにより新たに作成されたウィンドウ) のバッファーに切り替える、さまざまなコマンドのプレフィクスキーです。Emacs がウィンドウを選択または作成する方法については、Section 17.6.1 [Window Choice], page 191 を参照してください。

C-x 4 b *bufname* RET
: 他のウィンドウのバッファー *bufname*を選択します (switch-to-buffer-other-window)。Section 16.1 [Select Buffer], page 176 を参照してください。

C-x 4 C-o *bufname* RET
: バッファー *bufname*を選択せずに、別のウィンドウに表示します (display-buffer)。ウィンドウが選択される方法についての詳細は、Section 17.6 [Displaying Buffers], page 190 を参照してください。

C-x 4 f *filename* RET
: ファイル *filename*を visit して、他のウィンドウでバッファーを選択します (find-file-other-window). Section 15.2 [Visiting], page 146 を参照してください。

C-x 4 d *directory* RET
: *directory*の Dired バッファーを、別のウィンドウで選択します (dired-other-window)。Chapter 27 [Dired], page 380 を参照してください。

`C-x 4 m` `C-x m` (Chapter 29 [Sending Mail], page 420 を参照してください) と同様に、メールメッセージの編集を開始しますが、別のウィンドウで行います (`compose-mail-other-window`)。

`C-x 4 .` `M-.` (Section 25.4 [Xref], page 357 を参照してください) と同様に、識別子の定義を検索しますが、別のウィンドウで行います (`xref-find-definitions-other-window`)。

`C-x 4 r` *filename* `RET`
ファイル *filename*を読み取り専用で visit して、別のウィンドウでバッファーを選択します (`find-file-read-only-other-window`)。Section 15.2 [Visiting], page 146 を参照してください。

`C-x 4 4` このプレフィックスコマンドが呼び出した次コマンドが表示するバッファーに効果を及ぼす、より一般的なプレフィックスコマンドです。これは別ウィンドウに表示する次コマンドのバッファーを要求します。

`C-x 4 1` これは同一ウィンドウに表示する次コマンドのバッファーを要求する一般的なコマンドです。

17.5 ウィンドウの削除とリサイズ

`C-x 0` 選択されたウィンドウを削除します (`delete-window`)。

`C-x 1` フレームから選択されたウィンドウ以外のすべてのウィンドウを削除します (`delete-other-windows`)。

`C-x 4 0` 選択されていたウィンドウを削除して、それに表示されていたバッファーを kill します (`kill-buffer-and-window`)。このキーシーケンスの最後の文字はゼロです。

`M-x delete-windows-on RET` *buffer* `RET`
指定された *buffer*を表示しているウィンドウを削除します。

`C-x ^` 選択されたウィンドウの高さを増やします (`enlarge-window`)。

`C-x }` 選択されたウィンドウの幅を増やします (`enlarge-window-horizontally`)。

`C-x {` 選択されたウィンドウの幅を減らします (`shrink-window-horizontally`)。

`C-x -` バッファーに多くの行数が必要ない場合、そのウィンドウを縮小します (`shrink-window-if-larger-than-buffer`)。

`C-x +` すべてのウィンドウの高さを同じにします (`balance-windows`)。

選択されたウィンドウを削除するには `C-x 0` (`delete-window`) とタイプします (これはゼロ)。一度ウィンドウが削除されると、そのウィンドウが占めていたスペースは隣接したウィンドウに与えられます (ミニバッファーの場合にはアクティブな場合でも適用されない)。ウィンドウの削除はウィンドウを表示用に使っていたバッファーに影響を与えません。そのバッファーは存在を継続して、`C-x b`で切り替えることができます。オプション `delete-window-choose-selected`は、新たに選択されたウィンドウとしてどのウィンドウを選択するかを制御します (Section “Deleting Windows” in *The Emacs Lisp Reference Manual* を参照)。

`C-x 4 0` (`kill-buffer-and-window`) は、コマンド `C-x 0`より強力なコマンドです。これはカレントバッファーを kill してから、選択されたウィンドウを削除します。

`C-x 1` (`delete-other-windows`) は、選択されたウィンドウ以外のすべてのウィンドウを削除します。選択されたウィンドウはフレーム全体に拡張されます (このコマンドは、ミニバッファーのウィンドウがアクティブのとき使うことができません。これを試みるとエラーがシグナルされます)。

M-x delete-windows-onは、特定のバッファーを表示するウィンドウを削除します。これは、そのバッファーの入力を求めます (デフォルトはカレントバッファー)。C-u 0のようにプレフィックス引数が 0 の場合、このコマンドはカレントディスプレイ上のフレームのウィンドウだけを削除します。

コマンド C-x ^ (enlarge-window) は、フレームの高さを変えずに垂直方向に隣接するウィンドウのスペースを縮小して、選択されたウィンドウの高さを増やします。正の数引数を与えると、このコマンドは指定した行数分ウィンドウの高さを増やします。負の数引数を与えると、指定した行数分ウィンドウの高さを増やします。垂直方向に隣接するウィンドウが存在しない場合 (たとえばウィンドウの高さがフレーム全体の高さと同じとき)、エラーをシグナルします。このコマンドは変数 window-min-height(デフォルトは 4) で指定された、最小行数よりウィンドウの高さを縮小しようとしても、エラーをシグナルします。

同様に C-x } (enlarge-window-horizontally) は、選択されたウィンドウの幅を増やし、C-x { (shrink-window-horizontally) は幅を減らします。これらのコマンドは、変数 window-min-width(デフォルトは 10) で指定された最小列数よりウィンドウの幅を縮小すると、エラーをシグナルします。

モードライン (Section 18.5 [Mode Line Mouse], page 198 を参照してください)、またはウィンドウ分割線 (window dividers。Section 18.13 [Window Dividers], page 208 を参照してください) をマウスでクリックすることにより、ウィンドウの高さの変更およびウィンドウの分割や削除を行なう、別の方法を提供します。

C-x - (shrink-window-if-larger-than-buffer) は、バッファー全体を表示するのに必要な高さより選択されたウィンドウの高さが大きいときは、選択されたウィンドウの高さを減らします。余った行数はフレームの他のウィンドウに与えられます。

C-x + (balance-windows) を使って、選択されたフレームのすべてのウィンドウの高さを均等にすることもできます。

17.6 ウィンドウでのバッファーの表示

ユーザーのコマンドの結果として、任意のバッファーが表示またはポップアップされるのは、Emacs では一般的な処理です。コマンドがこれを行うには、いくつかの異なる方法があります。

C-x C-f (find-file) のような多くのコマンドは、デフォルトでは選択されたウィンドウを “乗っ取って” バッファーを表示します。

選択されたウィンドウを乗っ取らずに、たとえばウィンドウを分割して新しいウィドウを作り、そこにバッファーを表示するといったような、利口な表示を試みるコマンドがいくつかあります。さまざまなヘルプコマンド (Chapter 7 [Help], page 42) を含む、そのようなコマンドは内部的にdisplay-bufferを呼び出すことにより機能します。詳細は、Section 17.6.1 [Window Choice], page 191 を参照してください

他のコマンドは display-bufferと同じことを行いますが、それに加えてバッファーの編集を開始できるように、表示されたウィンドウを選択します。コマンド M-g M-n (next-error) が 1 つの例です (Section 24.2 [Compilation Mode], page 310 を参照してください)。そのようなコマンドは、内部的に関数 pop-to-bufferを呼び出すことにより機能します。Section “Switching to a Buffer in a Window” in *The Emacs Lisp Reference Manual* を参照してください。

名前が-other-windowで終わるコマンドは、display-bufferと同じように振る舞います。例外はそれらが決して選択されたウィンドウに表示しない点です。これらのコマンドのいくつかは、プレフィクスキー C-x 4にバインドされています (Section 17.4 [Pop Up Window], page 188 を参照してください)。

名前が`-other-frame`で終わるコマンドは、`display-buffer`と同じように振る舞います。例外は、i) 選択されたウィンドウに決して表示しない、ii) 望むバッファーを表示するために新たなフレームを作成するか、他のフレーム上のウィンドウを使用する、という 2 点です。これらのコマンドのいくつかは、プレフィクスキー `C-x 5`にバインドされています。

17.6.1 `display-buffer`が機能する方法

`display-buffer`コマンド (およびこのコマンドを内部的に呼び出すコマンド) は、以下で与えられたステップに従って、表示するウィンドウを選択します。このステップの順番を変更する方法については、Section “Choosing a Window for Displaying a Buffer” in *The Emacs Lisp Reference Manual* を参照してください。

- 他に考慮されるべき点とは無関係に、そのバッファーが選択されたウィンドウ内に表示されるべき場合は、選択されたウィンドウを再利用します。デフォルトではこのステップはスキップされますが、オプション `display-buffer-alist` (Section “Choosing a Window for Displaying a Buffer” in *The Emacs Lisp Reference Manual* を参照) にバッファー名にマッチする正規表現を追加して、アクション関数 `display-buffer-same-window` (Section “Action Functions for Buffer Display” in *The Emacs Lisp Reference Manual* を参照) でそれを参照することにより、Emacs にスキップしないよう告げることができます。たとえば、選択されたウィンドウ内に優先的にバッファー`*scratch*`を表示するには、以下のように記述します:

  ```
  (setopt
   display-buffer-alist
   '(("\\*scratch\\*" (display-buffer-same-window))))
  ```

  デフォルトでは、`display-buffer-alist`は `nil`です。

- 上記以外の場合、バッファーがすでに既存のウィンドウに表示されているときは、そのウィンドウを再利用します。通常は選択されたフレームのウィンドウだけが考慮されますが、対応するアクション alist`reusable-frames`エントリー (Section “Action Alists for Buffer Display” in *The Emacs Lisp Reference Manual* を参照) を使用している場合は、他のフレームのウィンドウも再利用可能です。これを行う例は、次のステップを参照してください。

- 上記以外の場合、オプションで新しいフレームを作成して、バッファーをそこに表示します。デフォルトではこのステップはスキップされます。これを有効にするには、以下のようにオプション `display-buffer-base-action` (Section “Choosing a Window for Displaying a Buffer” in *The Emacs Lisp Reference Manual* を参照) の値を変更してください:

  ```
  (setopt
   display-buffer-base-action
   '((display-buffer-reuse-window display-buffer-pop-up-frame)
     (reusable-frames . 0)))
  ```

  このカスタマイズでは、すべての可視およびアイコン化されたフレーム上の再利用可能なウィンドウを検索するステップを先行して試みるでしょう。

- 上記以外の場合、選択されたフレームのウィンドウを分割することにより、新しいウィンドウを作成して、バッファーを新しく作成したウィンドウに表示しようと試みます。

  分割は垂直または水平に行われる可能性があり、それは変数 `split-height-threshold` および `split-width-threshold`に依存します。これらの変数には整数値を指定します。`split-height-threshold`が選択されたウィンドウの高さより小さい場合、分割により下が新しいウィンドウになります。上記以外の場合、`split-width-threshold`が選択されたウィンドウの幅より小さい場合、分割により右が新しいウィンドウになります。どちらの条件も適用で

きなかったとき、Emacs は分割により下を新しいウィンドウにしようと試みますが、それは選択されたウィンドウが以前に分割されていなかった場合に限られます (過剰な分割を避けるため)。

- 上記以外の場合、そのウィンドウに前に表示されていたバッファーを表示します。通常は選択されたフレームのウィンドウだけが考慮されますが、適正なアクション alist エントリー `reusable-frames` (上記参照) により、他のフレームのウィンドウかもしれません。
- 上記以外の場合、選択されたフレームの既存のウィンドウのバッファーを表示します。
- 何らかの理由により上記すべてが失敗した場合、新しいフレームを作成して、そこにバッファーを表示します。

17.6.2 編集不可バッファーの表示

ウィンドウに表示されるバッファーの中には、編集のためではなく閲覧するためのものがあります。Help コマンド (Chapter 7 [Help], page 42 を参照) は通常、この目的のために`*Help*`と呼ばれるバッファーを使用し、ミニバッファーの補完 (Section 5.4 [Completion], page 31 を参照) は別の`*Completions*`と呼ばれるバッファーなどを使用します。このようなバッファーは通常、短時間しか表示されません。

Emacs は通常、このような一時的に表示されるウィンドウを、前のサブセクションで説明したように `display-buffer`を通じて表示します。一方、`*Completions*`バッファーは通常、そのフレームにいくつウィンドウが表示されているかに関わらず、選択されたフレームの最下のウィンドウに表示されます。

一時的なバッファーを他のやり方で Emacs に表示させたい場合、変数 `display-buffer-alist` (Section “Choosing a Window for Displaying a Buffer” in *The Emacs Lisp Reference Manual* を参照) をカスタマイズしてください。たとえば、常に選択されたウィンドウの下に`*Completions*`を表示するには、初期化ファイル (Section 33.4 [Init File], page 528 を参照) で以下のフォームを使用します:

```
(setopt
 display-buffer-alist
 '(("\\*Completions\\*" display-buffer-below-selected)))
```

Emacs は通常、バッファーの内容のすべてを表示するのに必要な大きさのウィンドウを作成するという点において、`*Completions*`バッファーは特別です。たとえば`*Help*`バッファーなど、他の一時表示でこのようなウィンドウのリサイズを行なうには、マイナーモード (Section 20.2 [Minor Modes], page 241 を参照) の `temp-buffer-resize-mode` (Section “Temporary Displays” in *The Emacs Lisp Reference Manual* を参照) に切り替えます。

`temp-buffer-resize-mode`でリサイズされるウィンドウの最大サイズは、オプション `temp-buffer-max-height`と `temp-buffer-max-width` (Section “Temporary Displays” in *The Emacs Lisp Reference Manual* を参照) で制御できます。最大サイズは、ウィンドウが含まれるフレームのサイズを超えることはできません。

17.7 ウィンドウ処理のための便利な機能

Winner モードはウィンドウの構成変更 (たとえばフレームのウィンドウがどのように分割されたか) を記録するグローバルマイナーモードなのでそれらを undo できます。Winner モードは `M-x winner-mode`、または変数 `winner-mode`をカスタマイズすることにより切り替えることができます。このモードが有効な場合には、`C-c left` (`winner-undo`) は左のウィンドウの構成変更を undo します。undo してから気が変わったら、`C-c right` (`M-x winner-redo`) を使って undo した変更を redo(再実行) することができます。Winner モードによる `C-c left`と `C-c right`のバインドを抑制

するために、変数 winner-dont-bind-my-keysを非nil値にカスタマイズできます。デフォルトではWinner モードはフレームごとに最大 200 個のウィンドウ構成を格納しますが、変数 winner-ring-sizeにより変更できます。Winner モードにリストアさせたくないようなウィンドウをもつバッファーがある場合には、それらの名前を変数 winner-boring-buffersか正規表現 winner-boring-buffers-regexpに追加してください。

Follow モード (M-x follow-mode) は、複数のウィンドウの同じバッファーを同期するので、バッファーの隣接した部分が常に表示されます。Section 11.7 [Follow Mode], page 82 を参照してください。

Windmove パッケージはフレーム内で隣接するウィンドウに方向的に移動するコマンドを定義します。M-x windmove-rightはカレントで選択されたウィンドウのすぐ右のウィンドウを選択して、他の方向 (left、up、down) にたいしても同じように機能します。windmove-default-keybindingsにより、これらのコマンドはS-right等にバインドされます。これを行うことにより、それらのキーによるシフト選択は無効になります (Section 8.6 [Shift Selection], page 57 を参照)。ウィンドウを方向的に選択するコマンドにたいしても同じ方法でキーバインディングを定義できます。次のコマンドが表示しようとするバッファー用のウィンドウをどの方向に表示するか指定するコマンドの定義には windmove-display-default-keybindingsを使用できます。方向的にウィンドウを削除するコマンド用にキーバインディングを定義するには windmove-delete-default-keybindings、選択されたウィンドウと指定方向のウィンドウのコンテンツの入れ替えるコマンド用のキーバインディング定義には windmove-swap-states-default-keybindingsがあります。

コマンド M-x compare-windowsは、異なるウィンドウに表示されたテキストを比較します。Section 15.9 [Comparing Files], page 163 を参照してください。

Scroll All モード (M-x scroll-all-mode) は、スクロールおよびポイント移動コマンドが、表示されているすべてのウィンドウに適用されるグローバルマイナーモードです。

17.8 ウィンドウのタブライン

コマンド global-tab-line-modeは各ウィンドウのスクリーン行上端へのタブライン *(tab line)* の表示を切り替えます。タブラインは各バッファーにたいしてウィンドウ内に特別なボタン (“tabs”) を表示して、対応するボタンをクリックすることによりバッファーを切り替えることができます。+アイコンをクリックすればバッファーのウィンドウローカルタブに新たなバッファーを追加、タブの xアイコンをクリックすればバッファーを削除します。タブライン上のマウスホイールはタブを水平方向にスクロールします。

前のウィンドウローカルタブの選択は C-x LEFT (previous-buffer)、次のタブの選択は C-x RIGHT (next-buffer) をタイプすることと等価です。いずれのコマンドも繰り返し回数としてプレフィクス数引数をサポートします。

タンブラインに優先されるコンテンツを定義するために、変数 tab-line-tabs-functionをカスタマイズできます。デフォルトでは上述のようにそのウィンドウで以前 visit したすべてのバッファーが表示されます。しかしカレントバッファーと同じメジャーモードのバッファーリストを表示したり、メジャーモードでグループ化したバッファー (最初のタブでモード名をクリックするとバッファーの別グループを選択可能なすべてのメジャーモードが表示される) を表示するようにセットすることもできます。

タブラインはタブバーとは異なることに注意してください (Section 18.17 [Tab Bars], page 210 を参照)。各フレーム上端にあるタブバーのタブはバッファーをもつ複数のウィンドウを含むウィンドウ構成間を切り替えますが、各ウィンドウ上端にあるタブラインのタブはウィンドウ内のバッファーの切り替えに使用します。

18 フレームとグラフィカルなディスプレイ

Emacs がグラフィカルなディスプレイ (たとえば X ウィンドウシステム) で開始されたときは、システムレベルのグラフィカルな表示領域 (display region) を占有します。このマニュアルではこれをフレームと呼び、“ウィンドウ” という言葉はフレームでバッファーを表示する部分のために使います。フレームには最初 1 つのウィンドウが含まれていますが、これを複数のウィンドウに分割することができます。フレームには通常、メニューバー、ツールバー、エコーエリアも含まれます。

追加のフレームを作ることもできます (Section 18.6 [Creating Frames], page 199 を参照してください)。同じ Emacs セッションで作られたすべてのフレームは、背後にあるバッファーや、その他のデータにアクセスします。たとえば 1 つ以上のフレームで表示されているバッファーは、あるフレームに表示されているものに変更を加えると、即座に他のフレームに反映されます。

`C-x C-c`とタイプすると、現在表示されているすべてのフレームを閉じて、他に表示されているフレームがなければ、Emacs セッションを終了します (Section 3.2 [Exiting], page 16 を参照してください)。選択されたフレームだけを閉じるには、`C-x 5 0`(これは `o`ではなくゼロです) とタイプします。

このセクションでは、グラフィカルなディスプレイに特有の機能 (特にマウスコマンド) と、複数フレームを管理する機能について説明します。テキスト端末では、これらの機能の多くは利用できません。しかしテキスト端末で複数のフレームを作ることは可能です。そのようなフレームは 1 度に 1 つだけ表示され、テキスト画面全体を占有します (Section 18.21 [Non-Window Terminals], page 215 を参照してください)。テキスト端末の中には、マウスを使うことが可能なものがいくつかあります (GNU および Unix systems でこれを行うには、Section 18.22 [Text-Only Mouse], page 215 を、MS-DOS でこれを使うには、Section “MS-DOS Mouse” in *Specialized Emacs Features* を参照してください)。メニューはすべてのテキスト端末でサポートされています。

18.1 編集のためのマウスコマンド

`mouse-1`　クリックした場所にポイントを移動します (`mouse-set-point`)。

`Drag-mouse-1`
: ドラッグにより選択されたテキストを取り囲むリージョンをアクティブ化して、そのテキストをプライマリー選択に置きます (`mouse-set-region`)。

`mouse-2`　クリックした場所にポイントを移動して、そこにプライマリー選択の内容を挿入します (`mouse-yank-primary`)。

`mouse-3`　リージョンがアクティブなときは、近くにあるリージョンの終端をクリックした位置に移動します。アクティブでないときは現在のポイントにマークをセットして、ポイントをクリックした位置に移動します。結果となるリージョンは kill リングに保存されます。2 回目のクリックでリージョンを kill します (`mouse-save-then-kill`)。

`C-M-mouse-1`
: ドラッグにより選択されたテキストを取り囲む矩形リージョンをアクティブにします。Section 9.5 [Rectangles], page 68 を参照してください。

もっとも基本的なマウスコマンドは`mouse-set-point`で、これはウィンドウのテキスト領域でマウスの左ボタン、`mouse-1`をクリックすることにより呼び出されます。これはポイントをクリックされた位置に移動します。そのウィンドウが選択されたウィンドウでなかったとき、そのウィンドウが選択されたウィンドウになります。`mouse-1`をダブルクリックして、リージョンをアクティブにすることもできます (Section 18.2 [Word and Line Mouse], page 196 を参照)。

クリックしたフレームが選択されたフレームでなかった場合は通常、クリックされたフレームが選択されたフレームになるのに加えて、ウィンドウも選択されてカーソルがセットされます。X ウィンドウシステムでは、変数 x-mouse-click-focus-ignore-positionを tにセットすることにより、これを変更できます。この場合、選択されていないフレームへの最初のクリックではフレームだけを選択し、他は変更しません。次にクリックするとそのウィンドウを選択してカーソルをその位置にセットします。

mouse-1を押してテキストの周辺をドラッグすると、最初にマウスボタンを押した位置にマークが置かれ、ボタンを離した位置にポイントがセットされ (Chapter 8 [Mark], page 52 を参照してください)、その領域がアクティブになります (mouse-set-region)。それに加えてリージョンのテキストがプライマリー選択となります (Section 9.3.2 [Primary Selection], page 66 を参照してください)。

変数 mouse-drag-copy-regionを非 nil値に変更すると、テキストの周囲をドラッグすることにより、そのテキストを kill リングに追加します。デフォルトは nilです。

この変数が non-emptyの場合には、リージョンが空でない場合にかぎり kill リングへコピーします。たとえばマウスで 1 文字の半分に満たないエリアをドラッグすると、通常なら空の文字列が kill リングにコピーされますが、non-emptyならこの短いマウスドラッグが kill リングに影響を与えることはありません。

ドラッグしている途中でマウスがウィンドウの上または下を超えた場合、マウスがウィンドウ内に戻るまで、ウィンドウが一定の割合でスクロールします。この方法により、スクリーン全体に収まらないリージョンを選択できます。1 度に何行スクロールするかは、マウスがウィンドウの縁からどれだけ離れたかに依存します。変数 mouse-scroll-min-linesは、最小ステップサイズを指定します。

オプション mouse-drag-mode-line-bufferが有効、かつウィンドウシステムがファイルのドラッグをサポートしていれば、モードライン内のバッファー名部分をドラッグすることによって、そのバッファーのファイルを別のプログラムやフレームにドラッグできます。

マウスの真ん中のボタン、mouse-2をクリックすると、クリックした位置にポイントを移動して、プライマリー選択の内容を挿入します (mouse-yank-primary)。Section 9.3.2 [Primary Selection], page 66 を参照してください。この振る舞いは、他の X アプリケーションと一貫性があります。かわりに mouse-2を、mouse-yank-at-clickにバインドできます。これはクリックした位置に yank するコマンドです。

変数 mouse-yank-at-pointを非 nil値に変更すると、mouse-2はポイントを移動しません。これはどこをクリックしたか、フレームのどのウィンドウをクリックしたかに関係なく、ポイントのある位置にテキストを挿入します。この変数は mouse-yank-primaryと mouse-yank-at-clickの両方に影響します。

マウスの右ボタン、mouse-3をクリックすると、コマンド mouse-save-then-killが実行されます。これはどこをクリックしたかと、リージョンの状態に依存していくつかのアクションを処理します。

- アクティブなリージョンがないときは、mouse-3のクリックにより、ポイントがあった位置にマークをセットし、クリックした位置にポイントを置いて、リージョンをアクティブにします。
- リージョンがアクティブなときは、mouse-3のクリックにより、クリックした位置に近いリージョンの終端を、クリックした位置に調整します。調整されたリージョンのテキストは、kill リングにコピーされます。元のリージョンのテキストがすでに kill リングにある場合は、それを置き換えます。
- 元のリージョンが mouse-1のダブルクリックまたはトリプルクリックで選択されたものである場合、リージョンは単語全体、または行全体にたいして定義されているので (Section 18.2 [Word

and Line Mouse], page 196 を参照してください)、mouse-3によるリージョンの調整も単語全体または行全体を単位として行われます。

- 同じ場所で連続して 2 回mouse-3を使うことにより、すでに選択されているリージョンを kill できます。したがってマウスでテキストを kill する簡単な方法は、まずリージョンの始端を決めるため mouse-1をクリックして、もう一方の終端で mouse-3を 2 回クリックします。テキストを削除せずに kill リングにコピーするには、mouse-3を 1 回だけクリックするか、テキスト範囲をドラッグします。その後は適当な場所で yank してコピーできます。

mouse-save-then-killコマンドは、変数 mouse-drag-copy-regionの値にもしたがいます (上記参照)。変数の値が非 nilのときは、コマンドがアクティブなリージョンをセットまたは調整したとき、常にリージョンのテキストは kill リングにも追加されます。一番最近の kill リングのエントリーが同じ方法で追加されたものである場合、新しいエントリーを作成せず、そのエントリーを置き換えます。

上記で説明した任意のマウスコマンドを使ってセットしたリージョンは、シフト選択以外のマウス移動コマンド、および通常のマークを非アクティブ化する方法により、マークが非アクティブになります。Section 8.6 [Shift Selection], page 57 を参照してください。

スクロールに使うことができる “ホイール” があるマウスもいくつかあります。ほとんどのグラフィカルなディスプレイで Emacs はデフォルトでマウスホイールによるウィンドウのスクロールをサポートします。この機能を切り替えるには M-x mouse-wheel-modeを使います。変数 mouse-wheel-follow-mouseおよび mouse-wheel-scroll-amountは、(どこでホイールによるスクロール操作が行われたかによる) スクロール対象の選択方法と、バッファーがスクロールされる量を決定します。変数 mouse-wheel-progressive-speedは、スクロールの早さがホイールを移動した早さにリンクするかを決定します。このモードはフォントサイズの拡大と縮小もサポートしており、デフォルトでは Ctrl修飾によるスクロールにバインドされています。このモードが有効だと、マウスホイールは wheel-upや wheel-downのようなスペシャルイベントを生成します (mouse-4や mouse-5と報告する古いシステムも一部あり)。マウスに水平スクロール用ホイールがあれば、wheel-leftや wheel-rightのようなイベントも同様に生成されます。

Emacs は Shift修飾による水平スクロールもサポートします。水平スクロールを開始する前に数プレフィックス引数 (例: M-5) をタイプすることにより、ユーザーオプション mouse-wheel-scroll-amount-horizontalが定義するステップ値が変更されます。

マウスがホイールのチルトが可能、あるいはタッチパッドがチルトをサポートしていれば、変数 mouse-wheel-tilt-scrollを非 nil値にカスタマイズして水平スクロールも有効にできます。デフォルトではマウスホイールをチルトするとチルトした方向にウィンドウのビューがスクロールします。つまり右にチルトするとウィンドウは右にスクロールするのでウィンドウに表示されていたテキストは水平左方向に移動します。水平スクロールの方向を逆転したい場合は、変数 mwheel-flip-directionを非 nil値にカスタマイズしてください。

Image モード (Section 15.19 [Image Mode], page 172 を参照) ではイメージ上にマウスポインターがあるときには、Ctrl修飾とともにマウスホイールのスクロールはマウスポインター下のイメージをスケーリング、Shift修飾でイメージを水平方向にスクロールします。

18.2 単語と行にたいするマウスコマンド

以下の mouse-1の変種は、1 度に単語全体または行全体を選択します。Emacs は選択されたテキスト周辺のリージョンをアクティブにして、kill リングにもコピーされます。

Double-mouse-1
: クリックした単語や文字周辺のテキストを選択します。

シンボルの構文をもつ文字 (C mode でのアンダースコアーなど) をダブルクリックすると、シンボルを取り囲むその文字を選択します。開きカッコ (または閉じカッコ) の構文をもつ文字をダブルクリックすると、そのカッコで始まる (または終わる) グループを選択します。区切り文字の構文をもつ文字 (C のシングルクォーテーションやダブルクォーテーション) をダブルクリックすると、文字列定数を選択します (Emacs はその文字により開始するのか (または終了するのか) を、発見的な手法を使って見つけ出します)。

カッコによるグループ化、または文字列の区切り文字の先頭をダブルクリックすると、そのリージョンの最後にポイントが移動します。新たな位置を表示するために必要なら、バッファーの表示を前方にスクロールします。カッコによるグループ化、または文字列の区切り文字の終端をダブルクリックしても、デフォルトではポイントはそのリージョンの終端に留まり、リージョンの先頭がウィンドウ上端より上にある場合は表示されません。必要ならバッファーの表示を後方にスクロールすてリージョンの先頭に移動するように変更するには、ユーザーオプション `mouse-select-region-move-to-beginning` を非 `nil`にセットしてください。

`Double-Drag-mouse-1`
: 単語全体を単位として、ドラッグした箇所のテキストを選択します。

`Triple-mouse-1`
: クリックした行を選択します。

`Triple-Drag-mouse-1`
: 行全体を単位として、ドラッグした箇所のテキストを選択します。

18.3 マウスで参照をフォローする

Emacs のバッファーにはボタンや、アクティブ化 (例えばクリック) したとき何らかのアクション (例えば参照をフォローする) を行う、ハイパーリンクを含むものがあります。ボタンのテキストは通常、アンダーラインが引かれていたり、周囲にボックスが描かれて、視覚的にハイライトされています。ボタンの上にマウスを移動すると、マウスカーソルの形状が変化して、ボタンがライトアップされます。変数 `mouse-highlight`を `nil`に変更すると、Emacs はこのハイライト機能を無効にします。

ボタンをアクティブにするには、ポイントをそこに移動して `RET`をタイプするか、`mouse-1`または `mouse-2`でボタンをクリックします。たとえば Dired バッファーでは、ファイル名がボタンです。これをアクティブにすることにより、Emacs はそのファイルを visit します (Chapter 27 [Dired], page 380 を参照してください)。`*Compilation*`バッファーでは、各エラーメッセージがボタンです。これをアクティブにすることにより、そのエラーにたいするソースコードを visit します (Section 24.1 [Compilation], page 309 を参照してください)。

ボタンを `mouse-1`でクリックすると、ボタンがアクティブになりますが、マウスボタンを押してから離すまで一定時間 (厳密に言うと 450 ミリ秒以上) が経過すると、Emacs はボタンをアクティブにせず、ポイントをクリックした場所に移動します。この方法によりボタンをアクティブにせずにポイントを移動するのに、マウスを使用できます。マウスをボタンの上にドラッグすると、通常どおりリージョンをセットして、ボタンはアクティブにしません。

ボタンにたいして `mouse-1`がどのように適用されるかは、`mouse-1-click-follows-link`をカスタマイズすることにより変更できます。変数の値が正の整数の場合、それはボタンのアクティブ化を取り消すのに、何ミリ秒マウスボタンを押しつづける必要があるかを指定します。前のパラグラフで説明したように、デフォルトは 450 です。値が `nil`の場合、`mouse-1`は単にクリックした場所にポイントを移動するだけで、ボタンをアクティブにしません。値が `double`の場合、シングルクリックでポイントのセット、ダブルクリックでボタンをアクティブにします。

選択されていないウィンドウのボタンでも通常、mouse-1でクリックすればボタンがアクティブになります。変数mouse-1-click-in-non-selected-windowsをnilに変更した場合、選択されていないウィンドウのボタンをmouse-1でクリックすると、クリックした位置にポイントを移動してウィンドウを選択しますが、ボタンはアクティブになりません。

18.4 メニューにたいするマウスクリック

CtrlおよびSHIFTで修飾されたマウスクリックにより、メニューが表示されるものがいくつかあります。

C-mouse-1
: このメニューはバッファーを選択するためのものです。

  MSB(“mouse select buffer”) グローバルマイナーモードは、このメニューをスマートで、よりカスタマイズ可能なものにします。Section 16.7.3 [Buffer Menus], page 185 を参照してください。

C-mouse-2
: このメニューには、フェイスや他のテキストプロパティをテストしたり、それらを設定するものが含まれます (後者は主に Enriched text を編集するのに便利です。Section 22.14 [Enriched Text], page 275 を参照してください)。

C-mouse-3
: このメニューは、モードに特有なメニューです。Menu-bar モードがオンの場合、ほとんどのモードでは、このメニューに、そのモード特有なメニューバーのメニューと同じアイテムを表示します。このボタンに異なるメニューを指定するモードもいくつかあります。Menu-bar モードがオフの場合、このメニューにはモード特有のメニューだけでなく、本来メニューバーに含まれるべきすべてのアイテムが含まれるので、メニューバーを表示せずに、それらにアクセスすることができます。

S-mouse-1
: このメニューはそのウィンドウのバッファーの、デフォルトのフェイスを変更するためのものです。Section 11.12 [Text Scale], page 88 を参照してください。

GUI アプリケーションの多くはコンテキストメニュー *(context menus)* の表示にmouse-3を使用しています。コンテキストメニューはマウスがクリックされた位置とコンテキスト (文脈) にたいして適している、さまざまなセッティングやアクションへのアクセスを提供します。マイナーモードcontext-menu-modeを有効にすれば、Emacs でmouse-3のデフォルト関数 (mouse-save-then-killにバインドされている。Section 18.1 [Mouse Commands], page 194 を参照) をコンテキストメニューにできます。有効にした後は、mouse-3のクリックにより Emacs はコンテキストメニューを表示します。これらのコンテキストメニューの正確な内容は、カレントメジャーモード、およびマウスクリック箇所周辺のバッファーコンテンツに依存します。コンテキストメニューの内容をカスタマイズするために、変数context-menu-functionsを使用できます (Section “Major Mode Conventions” in *The Emacs Lisp Reference Manual* を参照)。S-F10を押下してコンテキストメニューを呼び出すこともできます。

18.5 モードラインのマウスコマンド

ウィンドウのモードラインをマウスでクリックして、ウィンドウを選択したり操作することができます。

モードラインのいくつかの領域、たとえばバッファー名や、メジャーモードおよびマイナーモードは、独自のマウスバインディングをもっています。これらの領域にマウスを移動すると、その領域が

ハイライトされ、特別なバインディングが表示されます (Section 18.19 [Tooltips], page 213 を参照してください)。このセクションのコマンドは、それらの領域には適用できません。

`mouse-1` モードラインを `mouse-1`でクリックすると、それが属するウィンドウを選択します。モードライン上で `mouse-1`でドラッグすることにより、それを移動することができるので、ウィンドウの高さを変更することができます。マウスでの高さの変更により、ウィンドウが削除されることはありません。ウィンドウの高さが定められた最小値より小さくなる場合は、拒絶されます。

`mouse-2` モードラインを `mouse-2`でクリックすると、そのウィンドウがフレーム全体に表示されます。

`mouse-3` モードラインを `mouse-3`でクリックすると、それが属するウィンドウを削除します。フレームにウィンドウが 1 つしかないときは、何もしません。

`C-mouse-2`
モードラインを `C-mouse-2`でクリックすると、クリックした位置でウィンドウを左右に分割します (Section 17.2 [Split Window], page 186 を参照してください)。

さらに、左右に並んだモードラインの間にある分割線を、`mouse-1`でクリックしてドラッグすることにより、垂直の境界線を左右に移動できます。

ウィンドウのサイズ変更は、`window-resize-pixelwise`の値に影響されることに注意してください。詳細は Section 17.2 [Split Window], page 186 を参照してください。

18.6 フレームの作成

プレフィクスキー `C-x 5`は、`C-x 4`に類似しています。`C-x 4`コマンドが、選択されたフレームの別ウィンドウにバッファーを表示するのにたいして、`C-x 5`は異なるフレームを使います。可視またはアイコン化 ("最小化" とも言われる。Section "Visibility of Frames" in *The Emacs Lisp Reference Manual* を参照) されたフレームで、すでに要求されたバッファーが表示されている場合、そのフレームを手前に表示して非アイコン化 ("最小化解除") されます。それ以外の場合は、新しいフレームが現在の表示端末に作成されます。

以下の `C-x 5`コマンドは、選択するバッファーを検索したり作成する方法が異なります。

`C-x 5 2` デフォルトフレームパラメーターを使用して新しいフレームを作成します (`make-frame-command`)。

`C-x 5 c` カレントフレームのウィンドウ構成とフレームパラメーターを使用して新しいフレームを作成します (`clone-frame`)。

`C-x 5 b` *bufname* `RET`
バッファー *bufname*を他のフレームで選択します。これは`switch-to-buffer-other-frame`を実行します。

`C-x 5 f` *filename* `RET`
ファイル *filename*を visit して、そのバッファーを他のフレームで選択します。これは`find-file-other-frame`を実行します。Section 15.2 [Visiting], page 146 を参照してください。

`C-x 5 d` *directory* `RET`
ディレクトリー *directory*にたいする Dired バッファーを、他のフレームで選択します。これは `dired-other-frame`を実行します。Chapter 27 [Dired], page 380 を参照してください。

`C-x 5 m` 他のフレームでメールメッセージの作成を開始します。これは`compose-mail-other-frame`を実行します。これは`C-x m`の異なるフレーム版です。Chapter 29 [Sending Mail], page 420 を参照してください。

`C-x 5 .` 他のフレームで識別子の定義を検索します。これは`xref-find-definitions-other-frame`を実行する、`M-.`の複数フレーム版です。Section 25.4 [Xref], page 357 を参照してください。

`C-x 5 r` *filename* `RET`
ファイル *filename*を読み取り専用で visit して、そのバッファーを他のフレームで選択します。これは`find-file-read-only-other-frame`を実行します。Section 15.2 [Visiting], page 146 を参照してください。

`C-x 5 5` このプレフィックスコマンド直後に呼び出した次コマンドが表示するバッファーに効果を及ぼす、より一般的なプレフィックスコマンドです (`other-frame-prefix`)。これには別フレームに表示される次コマンドのバッファーが必要です。

フレームパラメーター *(frame parameters)* を指定することにより、新しく作成されるフレームの外見と動作を制御できます。Section 18.11 [Frame Parameters], page 206 を参照してください。

18.7 フレームコマンド

以下のコマンドは、フレームを削除したり操作するために使われます:

`C-x 5 0` 選択されたフレームを削除します (`delete-frame`)。1 つしかフレームがないときは、エラーをシグナルします。

`C-x 5 u` `undelete-frame-mode`が有効なら、最近削除された 16 のフレームのうちの 1 つのフレームにたいする削除を取り消します。プレフィックス引数なしならもっとも最近削除されたフレーム、1 から 16 までのプレフィックス数引数を指定すると対応するフレームの削除を取り消します (1 が最近削除されたフレーム)。

`C-z` 選択された Emacs フレームを最小化 (またはアイコン化) します (`suspend-frame`)。Section 3.2 [Exiting], page 16 を参照してください。

`C-x 5 o` 他のフレームを選択して手前に表示します。このコマンドを繰り返すと、端末のすべてのフレームを循環することができます。

`C-x 5 1` 現在の端末の、選択されたフレーム以外のすべてのフレームを削除します。

`M-F10` カレントフレームの最大化を切り替えます。フレームが最大化されているときはスクリーン全体に表示されます。

`F11` カレントフレームのフルスクリーンモードを切り替えます (フルスクリーンと最大化の違いは、前者がウィンドウマネージャーの装飾を隠すことで、これにより Emacs 自身のスクリーンスペースが若干増えます)。

フレームを本当に最大化またはフルスクリーンにするためには、変数`frame-resize-pixelwise`を非`nil`値にカスタマイズする必要があるウィンドウマネージャーもいくつかあります。この変数を非`nil`値にセットすると、一般的にフレームのサイズ変更を行や列の整数倍ではなく、ピクセル単位で行うことができます。

`C-x 5 0` (`delete-frame`) コマンドは、選択されたフレームを削除します。しかし Emacs セッションとの対話能力が失われるのを防ぐため、Emacs セッションの最後のフレームの削除は拒絶します。Emacs がデーモンとして実行されているとき (Section 31.6 [Emacs Server], page 469 を参照

してください) は、普通の対話的なフレームがすべて削除された後も、常に仮想的なフレーム (virtual frame) が残ります。この場合、`C-x 5 0`は最後の対話的なフレームを削除できます。Emacs セッションに再接続するには、`emacsclient`を使うことができます。

`C-x 5 1` (`delete-other-frames`) コマンドは、現在の端末 (端末にはグラフィカルなディスプレイとテキスト端末の両方が含まれます) の、カレントのフレーム以外のすべてのフレームを削除します。Section 18.21 [Non-Window Terminals], page 215 を参照してください)。他のグラフィカルなディスプレイ、またはテキスト端末で開いたフレームをもっている場合、これらは削除されません。

`C-x 5 o` (`other-frame`) コマンドは、現在の端末の次のフレームを選択します。Emacs を X ウィンドウシステム上のウィンドウマネージャーで使っていて、どんなフレームだろうと、マウスカーソルが上にくるとそのフレームを選択 (またはフォーカスを与える) するようになっている場合、このコマンドが正常に機能するために、変数`focus-follows-mouse`を`t`に変更する必要があります。これは`C-x 5 o`を呼び出し、マウスカーソルを選択されたフレームにワープさせます。

18.8 フォント

デフォルトでは Emacs はグラフィカルなディスプレイでテキストを表示するのに 10 ポイントの monospace フォントを使います。フォントのサイズはインタラクティブに変更可能です (Section 11.12 [Text Scale], page 88 を参照)。

異なるフォントを指定するために複数の異なる方法があります:

- '`Options`'メニューの '`Set Default Font`'をクリックします。これは既存のグラフィカルなフレームすべてのデフォルトを、選択したフォントにします。これを将来のセッションのために保存するには、'`Options`'メニューの '`Save Options`'をクリックしてください。
- 以下のように、`font`パラメーターを指定するように、変数`default-frame-alist`を変更する行を init ファイルに追加します:

  ```
  (add-to-list 'default-frame-alist
               '(font . "DejaVu Sans Mono-10"))
  ```

  これはこの init ファイルで再起動した後の、Emacs が作るすべてのグラフィカルなフレームのデフォルトを、指定したフォントにします。
- 以下のように X リソースファイルに、X リソースセッティング '`emacs.font`'を追加します:

  ```
  emacs.font: DejaVu Sans Mono-12
  ```

  X リソースファイルが効果を表すには、X を再起動するか`xrdb`コマンドを使わなければなりません。Section D.1 [Resources], page 591 を参照してください。X リソースファイルでは、フォント名をクォートしないでください。
- Emacs を GNOME デスクトップや Haiku で実行している場合には、変数`font-use-system-font`を`t`(デフォルトは`nil`) にセットすることによって、Emacs にデフォルトのシステムフォントの変更とともに、フレームのデフォルトフォントを調節するよう指示できます。これが機能するには、Emacs が Gsetting(または古い Gconf) のサポートつきでコンパイルされていなければなりません (使用する Gsetting 設定名は具体的には '`org.gnome.desktop.interface monospace-font-name`'と '`org.gnome.desktop.interface font-name`')。
- コマンドラインオプション '`-fn`'(または '`--font`') を使います。Section C.6 [Font X], page 585 を参照してください。

現在使っているフォントをチェックするには、`C-u C-x =`コマンドが有用です。これはポイント位置の文字の説明と、それを描画しているフォント名を表示します。

フォント名を表現する異なる方法がいくつか存在します。1 番目は *Fontconfig* パターンを使う方法です。Fontconfig パターンは以下の形式をもちます:

fontname[-*fontsize*][:*name1*=*values1*][:*name2*=*values2*]...

このフォーマットでは、大カッコ (brackets) の中の要素は省略可能です。*fontname*は、'Monospace' や 'DejaVu Sans Mono'のような、フォントのファミリー名です。*fontsize*は、フォントのポイントサイズ (1 プリンターポイントはおよそ 1/72 インチです) で、エントリー '*name*=*values*'は、フォントの slant や weight などのセッティングを指定します。*values*には 1 つの値か、カンマで区切られた値のリストを指定します。それらに加えていくつかのプロパティ値は、ある種のプロパティ名だけで有効なものがあり、それらについては '*name*='部分を省略できます。

以下は、一般的なフォントプロパティの一覧です:

'slant' 'italic'、'oblique'、'roman'のうちの、どれか 1 つです。

'weight' 'light'、'medium'、'demibold'、'bold'、'black'のうちの、どれか 1 つです。

'style' slant と weight を組み合わせた、特別な style を定義するフォントがいくつかあります。たとえば 'Dejavu Sans'は、'book' style を定義し、これは slant および weight プロパティをオーバーライドします。

'width' 'condensed'、'normal'、'expanded'のうちの、どれか 1 つです。

'spacing' 'monospace'、'proportional'、'dual-width'、'charcell'のうちの、どれか 1 つです。

以下は Fontconfig パターンの例です:

```
Monospace
Monospace-12
Monospace-12:bold
DejaVu Sans Mono:bold:italic
Monospace-12:weight=bold:slant=italic
```

Fontconfig パターンの、より詳細な説明は、Fontconfig のマニュアルを参照してください。これは Fontconfig とともに配布されており、`https://fontconfig.org/fontconfig-user.html`からオンラインで利用可能です。

MS-Windows ではすべてのフォントにたいして *fontname*[-*fontsize*] 形式のサブセットだけがサポートされています。これらの形式すべてにたいして完全な Fontconfig パターンは機能しないかもしれません。

フォントを指定する 2 番目の方法は、*GTK* フォントパターンを使う方法です。これらは以下の構文を使います。

fontname [*properties*] [*fontsize*]

*fontname*はファミリー名、*properties*はスペースで区切られたプロパティ値のリストで、*fontsize*はポイントサイズです。GTK フォントパターンで指定するプロパティは以下のようなものでしょう:

- Slant プロパティ: 'Italic'、'Oblique'。省略したときはデフォルトの slant(roman) が暗に指定されます。
- Weight プロパティ: 'Bold'、'Book'、'Light'、'Medium'、'Semi-bold'、'Ultra-light'。省略したときは 'Medium' weight が暗に指定されます。
- Width プロパティ: 'Semi-Condensed'、'Condensed'。省略したときはデフォルトの width が使用されます。

以下に GTK フォントパターンの例をいくつか示します:

```
Monospace 12
Monospace Bold Italic 12
```

MS-Windows では *fontname*のサブセットだけがサポートされています。

フォントを指定する 3 番目の方法は *XLFD(X Logical Font Description)* を使う方法であり、MS-Windows でもサポートされています。これは X でフォントを指定する際の伝統的な手法です。以下のように、各 XLFD は 14 の単語か数字をダッシュで区切ったものからなります:

```
-misc-fixed-medium-r-semicondensed--13-*-*-*-c-60-iso8859-1
```

XLFD でのワイルドカード文字 (‘`*`’) は、任意の文字の並び (none を含む) にマッチし、‘`?`’は任意の 1 文字にマッチします。しかしマッチングは実装依存で、長い名前の中のダッシュにたいするワイルドカードのマッチが不正確なことがあります。信頼できる結果を得るためには、14 個すべてのダッシュを指定して、ワイルドカードを 1 つのフィールドだけに使います。XLFD では大文字小文字の違いは重要ではありません。XLFD の構文は以下のとおりです:

```
-maker-family-weight-slant-widthtype-style...
...-pixels-height-horiz-vert-spacing-width-registry-encoding
```

エントリーは以下の意味をもちます:

maker フォントの manufacturer(製造者名) です。

family フォントのファミリー名です (たとえば ‘`courier`’)。

weight フォントの weight で、通常は ‘`bold`’、‘`medium`’、‘`light`’のどれかです。他の値をサポートするフォント名もいくつかあります。

slant フォントの slant で、通常は ‘`r`’(roman)、‘`i`’(italic)、‘`o`’(oblique)、‘`ri`’(reverse italic)、‘`ot`’(other) のどれかです。他の値をサポートするフォント名もいくつかあります。

widthtype フォントの width で、通常は ‘`normal`’、‘`condensed`’、‘`semicondensed`’、‘`extended`’のどれかです。他の値をサポートするフォント名もいくつかあります。

style オプションの追加の style 名です。通常これは空で、ほとんどの XLFD では、この位置に 2 つのハイフンを続けて指定します。style 名には、‘`ja`’や ‘`ko`’のような、ISO-639 language name の 2 文字を指定することもできます。CJK スクリプトをサポートするいくつかのフォントは、style 名の部分にこの記述をもちます。

pixels フォントの height をピクセルで指定します。

height プリンターのポイントの 1/10 を単位とした、スクリーン上のフォントの height です。フォントのポイントサイズの 10 倍です。垂直解像度 (vertical resolution) を与えれば、*height*と *pixels*は比例します。したがって一方を指定して、もう一方に ‘`*`’を指定するのが一般的です。

horiz フォントが意図するインチあたりのピクセル数で表した、水平解像度 (horizontal resolution) です。

vert フォントが意図するインチあたりのピクセル数で表した、垂直解像度 (vertical resolution) です。通常、システムのフォント解像度は、スクリーンにたいして正しい値になっています。したがって、これと *horiz*には ‘`*`’を指定するのが普通です。

spacing これには ‘`m`’(monospace)、‘`p`’(proportional)、‘`c`’(character cell) を指定します。

width ピクセル単位で表した文字の平均 width の 10 倍です。

registry
encoding フォントを描画する X フォント文字セット (X font character set) です (X フォント文字セットは Emacs 文字セットと同じではありませんが、似ています)。フォントの選択をチェックするのに、`xfontsel`コマンドを使うことができます。通常は、*registry*に‘`iso8859`’、*encoding*に‘`1`’を使うべきです。

フォントを指定する 4 番目の方法は、フォントニックネーム (font nickname) を使う方法です。特定のフォントは通常のフォント指定のかわりに、短いニックネームをもつものがあります。たとえば‘`6x13`’は以下と同じです

```
-misc-fixed-medium-r-semicondensed--13-*-*-*-c-60-iso8859-1
```

この形式は MS-Windows ではサポートされていません。

X 上での Emacs は、2 つのタイプのフォントを認識します。1 つはクライアントサイドのフォントで、これは Xft および Fontconfig ライブラリーにより提供されます。もう 1 つはサーバーサイドのフォントで、これは X サーバー自身により提供されます。ほとんどのクライアントサイドフォントは、アンチエイリアシング (antialiasing) やサブピクセルンティング (subpixel hinting) などの、サーバーサイドフォントにはない、進んだフォント機能をサポートします。Fontconfig と GTK パターンは、クライアントサイドフォントだけにマッチします。

すべての文字が同じ幅をもつ固定幅フォントを使いたいと思うでしょう。Xft および Fontconfig フォントでは、`fc-list`コマンドを使って、以下のようにして利用可能な固定幅フォントを一覧することができます。

```
fc-list :spacing=mono
fc-list :spacing=charcell
```

サーバーサイドの X フォントにたいしては、`xlsfonts`プログラムを使って、以下のようにして利用可能な固定幅フォントを一覧することができます。

```
xlsfonts -fn '*x*' | grep -E '^[0-9]+x[0-9]+'
xlsfonts -fn '*-*-*-*-*-*-*-*-*-*-*-*-m*'
xlsfonts -fn '*-*-*-*-*-*-*-*-*-*-*-*-c*'
```

XLFD の *spacing*フィールドが、‘`m`’または‘`c`’のフォントは固定幅フォントです。特定のフォントの外見を見るには、以下のように `xfd`コマンドを使います:

```
xfd -fn 6x13
```

これはフォント‘`6x13`’の全体を表示します。

Emacs を実行しているときは、特定の種類のテキスト (Section 11.8 [Faces], page 82 を参照してください) や、特定のフレーム (Section 18.11 [Frame Parameters], page 206 を参照してください) のフォントをセットすることもできます。

18.9 スピードバーフレーム

スピードバー *(speedbar)* は、簡単に他のフレームを操作したり処理するための、特別なフレームです。スピードバーが存在するとき、それは常にアタッチされたフレーム *(attached frame)* と呼ばれる、特定のフレームに対応しています。すべてのスピードバー操作は、そのフレームにたいして処理されます。

`M-x speedbar`とタイプすると、スピードバーを作成して、カレントフレームに関連付けます。スピードバーを解除するには、再度 `M-x speedbar`とタイプするか、スピードバーを選択して `q`とタイプします (他の Emacs フレームを削除するのと同じ方法で、スピードバーフレームを削除することも

できます)。スピードバーを他のフレームに関連付けたいときは、一旦解除してから、そのフレームで `M-x speedbar`を呼び出します。

スピードバーは、さまざまなモードを処理できます。デフォルトモードは *File Display(*ファイル表示*)* モードで、これはアタッチされたフレームの、選択されたウィンドウのカレントディレクトリーのファイルを、1 行に 1 ファイルずつ表示します。ディレクトリー以外をクリックすると、アタッチされたフレームの、選択されたウィンドウでそのファイルを visit し、ディレクトリーをクリックすると、スピードバーでそのディレクトリーを表示します (Section 18.3 [Mouse References], page 197 を参照してください)。各行には '`[+]`'や '`<+>`'が記されたボックスがあり、それをクリックすると、そのアイテムの内容を展開します。ディレクトリーを展開すると、ディレクトリーの内容を、ディレクトリー自身の行の下に加えてスピードバー表示します。通常のファイルを展開すると、ファイルの中の tag リストをスピードバー表示に加えます。tag 名をクリックして、アタッチされたフレームの選択されたウィンドウで、tag にジャンプできます。ファイルまたはディレクトリーが展開されているときは、'`[+]`'が '`[-]`'に変化します。ボックスを再度クリックすると、アイテムを収納して、内容を隠すことができます。

キーボードを使ってスピードバーを操作することもできます。スピードバーでポイントのある行で `RET`をタイプするのは、そのアイテムをクリックするのと等しく、`SPC`はアイテムを展開または収納します。`U`は、カレントディレクトリーの親ディレクトリーを表示します。カレント行のファイルをコピー、削除、リネーム<するには、`C`、`D`、`R`をタイプします。新しいディレクトリーを作成するには、`M`をタイプします。

スピードバーのモードで、他に一般的な目的に使われるのは *Buffer Display(*バッファー表示*)* モードです。このモードでは、スピードバーは Emacs バッファーのリストを表示します。このモードに切り替えるにはスピードバーで `b`をタイプします。File Display モードに戻るには、`f`をタイプします。スピードバーのウィンドウのどこかを `mouse-3`でクリック (またはモードラインを `mouse-1`でクリック) して、ポップアップメニューの '`Displays`'を選択しても、表示モードを変更することができます。

Rmail モード、Info、GUD を含むいくつかのメジャーモードは、スピードバーに選択可能な便利なアイテムを配する、特別な方法をもっています。たとえば Rmail モードでは、スピードバーは Rmail ファイルのリストを表示し、カレントメッセージを他の Rmail ファイルに移動する場合は、それの '`<M>`'ボックスをクリックします。

スピードバーの使い方とプログラミングに関する詳細は、*Speedbar Manual* を参照してください。

18.10 複数ディスプレー

1 つの Emacs が、1 つ以上の X ディスプレイと通信できます。最初、Emacs は環境変数 `DISPLAY`、または '`--display`'オプション (Section C.2 [Initial Options], page 576 を参照してください) で指定された、ただ 1 つのディスプレイに表示されます。他のディスプレイに接続するには、コマンド `make-frame-on-display`を使います:

`M-x make-frame-on-display RET` *`display`* `RET`
: ディスプレイ *display*に新しいフレームを作成します。

1 つの X サーバーは、1 つ以上のスクリーンを処理できます。1 つのサーバーに属する 2 つのスクリーンでフレームを開いた場合、Emacs はそれらが 1 つのキーボードを共有するのを知っているので、これらのスクリーンから到着するすべてのコマンドを 1 つの入力ストリームとして扱います。

異なる X サーバーでフレームを開いた場合、Emacs は各サーバーごとに別な入力ストリームを作成します。各サーバーは、それぞれ選択されたフレームをもちます。特定の X サーバーで入力したコマンドは、そのサーバーの選択されたフレームに適用されます。

マルチモニターディスプレイではコマンド `make-frame-on-monitor`が使用可能です:

`M-x make-frame-on-monitor RET` *`monitor`* `RET`
: モニター *monitor*上にスクリーン領域がカレントディスプレイの一部であるようなフレームを新たに作成します。

18.11 フレームパラメーター

変数 `default-frame-alist`の中の、フレームパラメーター *(frame parameters)* のデフォルトリストを指定することにより、すべてのフレームのデフォルトの外見と振る舞いを制御することができます。この値はエントリーのリストで、各エントリーにはパラメーター名と、そのパラメーターの値を指定します。これらのエントリーは Emacs が新しいフレームを作るとき (初期フレームを含む) に効果を及ぼします。

たとえば以下のファイルを init ファイル (Section 33.4 [Init File], page 528 を参照してください) に追加することにより、デフォルトのフレーム幅が 90 列、デフォルトのフレーム高さが 40 行、デフォルトフォントに 'Monospace-10'を指定します。

```
(add-to-list 'default-frame-alist '(width  . 90))
(add-to-list 'default-frame-alist '(height . 40))
(add-to-list 'default-frame-alist '(font . "Monospace-10"))
```

フレームパラメーターのリストと、その効果については、Section “Frame Parameters” in *The Emacs Lisp Reference Manual* を参照してください.

変数 `initial-frame-alist`をカスタマイズすることにより、初期フレームだけに適用されるフレームパラメーターのリストを指定することもできます。

Emacs が X ツールキットを使ってコンパイルされている場合、カラーとフォントを指定するフレームパラメーターは、メニューとメニューバーには影響を及ぼしません。なぜなら、それらは Emacs が直接描画しているのではなく、ツールキットにより描画されているからです。

フレームの外観と振る舞いも、X リソースを通じてカスタマイズできます (Appendix D [X Resources], page 591 を参照)。これらは init ファイル内で指定された初期フレームのパラメーターをオーバーライドします。

セッションの保存とリストアのために desktop ライブラリーを使用している場合は、desktop ファイルに記録されたフレームが、これらのパラメーターとともにリストアされることに注意してください。これらのフレームがリストアされるときは、init ファイル内で `default-frame-alist`および `initial-frame-alist`で指定されているフレームパラメーターよりも、記録されたパラメーターが優先されます。これを避ける方法については、Section 31.10 [Saving Emacs Sessions], page 482 を参照してください。

18.12 スクロールバー

グラフィカルなディスプレイでは、Emacs ウィンドの横に垂直スクロールバー *(vertical scroll bar)* があります。スクロールバーの up ボタンまたは down ボタンを `mouse-1`でクリックすると、ウィンドウを 1 行ずつスクロールします (しかし、これらのボタンがないようにスクロールバーをカスタマイズできるツールキットもある)。スクロールバー内部のボックスの上または下を `mouse-1`でクリックすると、`M-v`または `C-v`と同様に、ほぼウィンドウ全体の高さ分スクロールします (Section 4.2 [Moving Point], page 18 を参照) (これも、いくつかのツールキットでは動作が異なるかもしれない)。スクロールバー内部のボックスをドラッグすると、連続してスクロールします。

Emacs が X ウィンドウシステム上で X ツールキットサポートなしでコンパイルされている場合、スクロールバーは違った振る舞いをします。スクロールバーの任意の箇所を `mouse-1`でクリックする

と C-vのように前方にスクロールし、mouse-3でクリックすると M-vのように後方にスクロールします。スクロールバーで mouse-2をクリックすると、スクロールバー内部のボックスを上下にドラッグできます。

垂直スクロールバーの使用を切り替えるには、M-x scroll-bar-modeとタイプします。このコマンドは、まだ作成されていないフレームも含めて、すべてのフレームに適用されます。選択されたフレームの垂直スクロールバーだけ切り替えたい場合は、コマンド M-x toggle-scroll-barを使用してください。

起動時に垂直スクロールバーの使用を制御するには、変数 scroll-bar-mode (Chapter 33 [Customization], page 499 を参照) をカスタマイズします。この変数の値は、right(ウィンドウの右にスクロールバーを配します)、left(ウィンドウの左にスクロールバーを配します)、nil(垂直スクロールバーを無効にします) のどれかです。Emacs が X ウィンドウシステム上で GTK+サポートつきでコンパイルされている、または MS-Windows、macOS の場合、デフォルトでは右にスクロールバーを配します。Emacs が X ウィンドウシステム上で GTK+サポートなしでコンパイルされている場合、(古い X アプリケーションの慣例にしたがって) スクロールバーを左に配します。

X リソース 'verticalScrollBars'を使って、スクロールバーの有効または無効にすることができます (Section D.1 [Resources], page 591 を参照してください)。スクロールバーの幅を制御するにはフレームパラメーター scroll-bar-widthを変更してください (Section "Frame Parameters" in *The Emacs Lisp Reference Manual* を参照してください)。

(GTK+または Motif とともに)X 上で Emacs を使っている場合、変数 scroll-bar-adjust-thumb-portionをカスタマイズして、スクロールバーのオーバースクロール *(overscrolling:* たとえばバッファーの最後が表示されていてもさらに下にスクロールします*)* を制御できます。変数の値が非 nilの場合、バッファーの最後が表示されていてもスクロールバーを下にドラッグできます。nilの場合、バッファーの最後が表示されたとき、内部のボックスはスクロールバーの最下になります。バッファー全体が表示されているときは、オーバースクロールできません。

スクロールバーの視覚的な外見は、scroll-barフェイスにより制御されます。

グラフィカルなフレームでは、垂直スクロールバーは暗に横並びのウィンドウを区別する役目を担っています。垂直スクロールバーが無効なとき、Emacs はデフォルトでは幅 1 ピクセルの垂直ボーダー *(vertical border)* によりそのようなウィンドウを区別します。このボーダーは右側のウィンドウの最初の 1 ピクセルの列を占め、したがってそこに表示されていたグリフの左端ピクセルの上に上書きされることになるでしょう。これらのピクセルに重要な情報が含まれる場合は、window divider を有効にしてそれらを可視にできます (Section 18.13 [Window Dividers], page 208 を参照)。垂直ボーダーの見栄えを複製するには、フレームの right-divider-widthパラメーターを 1 にセットして、window-dividerフェイスを vertical-borderのフェイスから継承してください (Section "Window Dividers" in *The Emacs Lisp Reference Manual* を参照)。

ツールキットサポート付きのグラフィカルなディスプレイでは、Emacs は各ウィンドウの最下に水平スクロールバー *(horizontal scroll bar)* も提供します。スクロールバーの left ボタンまたは right ボタンを mouse-1でクリックすると、ウィンドウを 1 列ずつ水平にスクロールします (これらのボタンを表示しないようスクロールバーをカスタマイズできるツールキットがあることに注意)。スクロールバーの内部ボックスの左または右下を mouse-1でクリックすると、ウィンドウは 4 列スクロールします。スクロールバー内部のボックスをドラッグすると、連続してスクロールします。

このような水平スクロールにより、そのウィンドウのポイント位置が、右側または左側の見えない場所になるかもしれないことに注意してください。文字をタイプしてテキストを挿入したり、キーボードコマンドでポイントを移動することにより通常、ポイントは視界に戻るでしょう。

水平スクロールバーの使用を切り替えるには、M-x horizontal-scroll-bar-modeとタイプします。このコマンドは、まだ作成されていないフレームも含めて、すべてのフレームに適

用されます。選択されたフレームの水平スクロールバーだけ切り替えたい場合は、コマンド `M-x toggle-horizontal-scroll-bar`を使用してください。

起動時に水平スクロールバーの使用を制御するには、変数 `horizontal-scroll-bar-mode`をカスタマイズしてください。

X リソース '`horizontalScrollBars`'を使って、水平スクロールバーを有効または無効にすることができます (Section D.1 [Resources], page 591 を参照してください)。スクロールバーの高さを制御するには、フレームパラメーター `scroll-bar-height`を変更してください (Section "Frame Parameters" in *The Emacs Lisp Reference Manual* を参照してください)。

18.13 ウィンドウ divider

グラフィカルなディスプレイでは、ウィンドウの外観を分割するために、*window dividers* を使用することができます。window dividers はマウスでドラッグできるバーなので、隣接したウィンドウを簡単にリサイズできます。

window dividers の表示を切り替えるには、コマンド `M-x window-divider-mode`を使用します。

dividers をどこに表示するかカスタマイズするには、オプション `window-divider-default-places`を使用します。変数の値には `bottom-only` (dividers をウィンドウの下だけに表示します)、`right-only` (dividers をウィンドウの右だけに表示します)、または `t` (下と右の両方に表示します) を指定します。

このモードにより表示される window dividers の幅を調整するには、オプション `window-divider-default-bottom-width`および `window-divider-default-right-width` をカスタマイズします。

ウィンドウの列の最初の 1 ピクセル (window divider が無効な場合は横並びのウィンドウを区別する垂直ボーダーに使用されます) を可視とする window divider は、垂直スクロールバーが無効のときにも有用です (Section 18.12 [Scroll Bars], page 206 を参照)。

window dividers についての詳細は、Section "Window Dividers" in *The Emacs Lisp Reference Manual* を参照してください。

18.14 ドラッグアンドドロップ

ほとんどのグラフィカルなデスクトップ環境で、Emacs はドラッグアンドドロップ *(drag and drop)* 操作にたいする、基本的なサポートを提供します。たとえば Emacs のフレームにテキストをドロップすると、ドロップされた箇所にテキストを挿入し、ファイルをドロップすると、Emacs フレームはそのファイルを visit します。特別なケースとしては、Dired バッファーにファイルをドロップすることにより、バッファーに表示されているディレクトリーにファイルを移動またはコピーします (アプリケーションの慣例に基づきます)。

ファイルをドロップすると、通常はドロップされたウィンドウでファイルを visit します。そのような場合に、新しいウィンドウでファイルを visit したいときは、変数 `dnd-open-file-other-window` をカスタマイズします。

現在のところ、XDND および Motif drag and drop protocols と、古い KDE 1.x protocol がサポートされています。

Emacs のウィンドウにテキストをドラッグしている際に、スクロールやドロップされたテキストが挿入される場所を判断するのが難しい場合があるでしょう。オプション `dnd-indicate-insertion-point`を非 `nil`値にセットすれば、ドラッグ中にマウスがウィンドウ内を移動する際にドロップされるテキストが挿入される位置にポイント移動がするようにできます。また `dnd-scroll-margin`を整

数値にセットすれば、ドラッグ中にウィンドウの上端または下端の指定した行数部分にマウスが移動するとウィンドウがスクロールするようにできます。

Emacs はオプションで、テキストのリージョンを他のバッファーの他の部分へドラッグする操作もサポートします。これを有効にするには、変数 `mouse-drag-and-drop-region`を非 `nil`値にカスタマイズしてください。通常そのテキストは、ドロップ先が元のリージョンと同じなら移動 (カットアンドペースト) され、他のバッファーならそのリージョンがコピーされます。この変数の値が‘`shift`’、‘`control`’、‘`alt`’のような修飾キー名の場合、その修飾キーを押しながらテキストをドロップすると、ドロップ先がドラッグ元と同じバッファーでも、テキストはカットされずにコピーされます。

ドラッグ元とドロップ先が異なるバッファーのときにもテキストをカットするためには、オプション `mouse-drag-and-drop-region-cut-when-buffers-differ`を非 `nil`値にセットします。デフォルトでは、グラフィカルなディスプレイの選択されたテキストはツールチップ内に表示され、ドラッグ中にはマウスカーソルとともにポイントが移動します。このような動作を抑止するには、オプション `mouse-drag-and-drop-region-show-tooltip`、および/または `mouse-drag-and-drop-region-show-cursor`に `nil`をセットしてください。

Emacs から他のプログラムにテキストをドラッグできるようにするには、オプション `mouse-drag-and-drop-region-cross-program`に非 `nil`値をセットしてください。

X ウィンドウシステムでは、Emacs によるファイルの保存を期待してファイルを Emacs にドロップできるプログラムがいくつかあります。通常だと Emacs はファイルを保存する名前の入力を求めた後にそのファイルをオープンしますが、変数 `x-dnd-direct-save-function`を変更することによってこの挙動を変えることができます。Section “Drag and Drop” in *The Emacs Lisp Reference Manual* を参照してください。

18.15 メニューバー

`M-x menu-bar-mode`で、メニューバーの使用を切り替えることができます。引数を指定しないと、このコマンドはグローバルマイナーモードの Menu Bar を切り替えます。引数を指定した場合、引数が正なら Menu Bar モードをオンにして、負ならオフにします。起動時にメニューバーの使用を制御するには、変数 `menu-bar-mode`をカスタマイズしてください。

熟練したユーザーは、テキストのために更なる行を得るために、メニューをオフにしている場合がしばしばあります (特にテキスト端末時)。メニューバーがオフでも、ポップアップメニューをサポートするディスプレイなら、`C-mouse-3`でメニュー内容を含むポップアップメニューを表示できます。あるいは `mouse-3`でコンテキストメニューをポップアップするために、`context-menu-mode`を有効にして変数 `context-menu-functions`をカスタマイズすることもできます。Section 18.4 [Menu Mouse Clicks], page 198 を参照してください。

メニューバーのコマンドを呼び出す方法については、Section 1.4 [Menu Bar], page 10 を参照してください。メニューバーのメニューの視覚的な外見をカスタマイズする方法は、Appendix D [X Resources], page 591 を参照してください。

18.16 ツールバー

グラフィカルなディスプレイでは、Emacs はフレームのトップ、メニューバーの直下にツールバー *(tool bar)* を配します。これはマウスでクリックすることにより、さまざまなコマンドを呼び出すことができるアイコンが 1 列に並んだものです。

(デフォルトの) グローバルツールバーは、一般的なコマンドを含みます。自身のツールバーを定義するメジャーモードもいくつか存在します。バッファーがそれらのメジャーモードの場合、モードのツールバーにより、グローバルツールバーが置き換えられます。

ツールバーの使用を切り替えるには、`M-x tool-bar-mode`とタイプします。このコマンドは、まだ作成されていないフレームを含むすべてのフレームに適用されます。起動時にツールバーの使用を制御するには、変数`tool-bar-mode`をカスタマイズします。

Emacs が GTK+サポートつきでコンパイルされている場合、ツールバーアイテムには、イメージ、テキストラベル、またはそれら両方を含むことができます。デフォルトでは Emacs は Gnome デスクトップの、ツールバースタイルセッティングにしたがいます。もし何も定義されていない場合、ツールバーのアイテムにはイメージだけが表示されます。特定のツールバースタイルを強要するには、変数`tool-bar-style`をカスタマイズしてください。

フレームパラメーター`tool-bar-position`で、GTK+ツールバーのためのツールバーの配置を制御することもできます。Section “Frame Parameters” in *The Emacs Lisp Reference Manual* を参照してください。

NS ビルドはツールバーをウィンドウ装飾と判断するため、ウィンドウが装飾されていないときはツールバーを表示しません。Section “Frame Parameters” in *The Emacs Lisp Reference Manual* を参照してください。macOS では、フレームがフルスクリーンのときツールバーは表示されませんが、スクリーン上端にマウスポインターを移動することにより表示することができます。

18.17 タブバー

グラフィカルなディスプレイとテキスト端末では、Emacs は変数`tab-bar-position`に応じてオプションで各フレームのトップ、メニューバー (Section 18.15 [Menu Bars], page 209 を参照) の直下にタブバー *(Tab Bar)*、上または下にツールバー (Section 18.16 [Tool Bars], page 209 を参照) を表示できます。タブバーとはタブ *(tabs)* の行 (そのフレーム上のウィンドウ構成間を切り替えるためにクリック可能) です。

タブバー上の各タブは、そのフレームの名前をもった永続的なウィンドウ構成 (そのフレームがウィンドウに分割される方法と各ウィンドウにどのバッファーを表示するか) を表します。タブの名前はウィンドウ構成で可視なバッファーの名前のリストから構成されます。タブをクリックすることによってタブに記録されたウィンドウ構成 (以前そのタブがカレントタブだったときにフレームで使用されたウィンドウとバッファーの構成) に切り替わります。

セッションの保存とリストアにデスクトップライブラリー (Section 31.10 [Saving Emacs Sessions], page 482 を参照) を使用している場合には、タブバーのタブは関連付けられたウィンドウ構成とともにデスクトップファイルに記録されて、それはセッションをリストアした後に利用することができます。

タブバーはタブラインとは異なることに注意してください (Section 17.8 [Tab Line], page 193 を参照)。各ウィンドウ上端にあるタブラインのタブはウィンドウ内でのバッファーの切り替え、各フレーム上端にあるタブバーのタブは 1 つ以上のバッファーを表示する複数のウィンドウを含むウィンドウ構成の切り替えに使用します。

タブバーの使用を切り替えるには、`M-x tab-bar-mode`とタイプします。このコマンドは、まだ作成されていないフレームを含むすべてのフレームに適用されます。起動時にタブバーの使用を制御するには、変数`tab-bar-mode`をカスタマイズして保存してください。

変数`tab-bar-show`は Tab Bar モードを自動的にオンにするかどうかを制御します。この値が`t`なら、新たなタブを作成するコマンドの使用時に`tab-bar-mode`モードが有効に成増。値が`1`ならタブが 1 つだけのときはタブバーを隠して、さらにタブが作成されたら再び表示します。より一般的には値が非負の整数なら、タブ数がその数より多くなったときだけタブバーを表示します。値`nil`は常にタブバーを非表示にします。この場合でも`M-x tab-next`、`M-x tab-switcher`、およびタブ名の補完を提供するその他のコマンドを使用することによってタブバーなしで名前つきのウィンドウ構成

を切り替えることができます。M-x tab-new、M-x tab-close等のコマンドを使用することにより、タブバーの作成やクローズも可能です。

tab-bar-showに数値をセットすることで、フレームごとに作成されたタブ数に応じて、あるフレームではタブバーを表示したり、他のフレームでは表示しないようにすることができます。

選択されたフレーム上のタブバーのみ使用を切り替えるには M-x toggle-frame-tab-barとタイプします。このコマンドにより tab-bar-modeと tab-bar-showの値に関わらず、あるフレームではタブバー表示を有効、他のフレームでは無効にすることができます。

プレフィクスキー C-x tは C-x 5と似ています。C-x 5コマンドはバッファーを別フレームにポップアップしますが (Section 18.6 [Creating Frames], page 199 を参照)、C-x tコマンドは選択されたフレームで異なるウィンドウ構成に別のタブを使用します。

以下の C-x tコマンドは選択するバッファーを検索したり作成する方法が異なります。以下のコマンドは新たなタブでのバッファーの選択に使用できます。:

C-x t 2
: タブを新たに追加します (tab-new)。変数 tab-bar-new-tab-choiceのカスタマイズによって新たなタブに表示するバッファーの選択を制御できます。変数 tab-bar-tab-name-functionをカスタマイズすれば新しいタブに付与されるデフォルト名を制御できます。

C-x t b *bufname* RET
: バッファー *bufname*を他のタブで選択します。これは switch-to-buffer-other-tab を実行します。

C-x t f *filename* RET
: ファイル *filename*を visit(Section 15.2 [Visiting], page 146 を参照) して、そのバッファーを他のタブで選択します。これは find-file-other-tabを実行します。

C-x t d *directory* RET
: 別のタブで指定された *directory*を編集します (Chapter 27 [Dired], page 380 を参照)。ディレクトリーにたいする Dired バッファーを、他のタブで選択します。これは dired-other-tabを実行します。

C-x t t
: これはプレフィクスコマンド (other-tab-prefix) であり、これの直後に呼び出される次コマンドに効果を及ぼします。このプレフィクスコマンドは次コマンドにより表示されるバッファーが別のタブに表示されるるよう要求します。

デフォルトでは新たなタブを追加したコマンドの呼び出し前にカレントだったバッファーとともに新たなタブが開始されます。別のバッファーとともに新たなタブを開始するには、変数 tab-bar-new-tab-choiceをカスタマイズしてください。

変数 tab-bar-new-tab-toは新たなタブを配置する場所を定義します。デフォルトでは新たなタブはカレントタブの右側に追加されます。

以下のコマンドはタブの削除に使用されます:

C-x t 0
: 選択されたタブをクローズします (tab-close)。変数 tab-bar-close-last-tab-choiceがデフォルト値なら、1 つのタブの場合には効果はありません。

C-x t 1
: 選択されたタブを除き、選択されたフレーム上のすべてのタブを削除します。

変数 tab-bar-close-tab-selectはカレントタブをクローズ後に選択するタブを定義します。デフォルトではもっとも最近使用したタブを選択します。

コマンド tab-undoは最後にクローズしたタブをリストアします。

以下のコマンドはタブ間を切り替えるために使用されます:

C-x t o
C-TAB 次のタブに切り替えます (tab-next)。このコマンドを繰り返すと選択されたフレームのすべてのタブを巡回します。正の数引数 *n*では次の *n*番目のタブ、負の引数 −*n*では前の *n*番目のタブに切り替えます。

S-C-TAB 前のタブに切り替えます (tab-previous)。正に数引数 *n*では前の *n*番目のタブ、負の引数 −*n*では次の *n*番目のタブに切り替えます。

C-x t RET *tabname* RET
すべてのタブの名前の補完つきで名前でタブを切り替えます (tab-switch)。デフォルト値、およびタブ名の “将来のヒストリー (future history)” はもっとも最近使用された順にソートされたタブ名なので、M-n (next-history-element) で最後に visit したタブ、その前のタブ、... を取得します。

modifier-*tab-number*
タブ名でタブを番号 *tab-number*で切り替えます (tab-select)。1 つ以上の *modifier* キーを指定するために変数 tab-bar-select-tab-modifiersをカスタマイズした後には、選択するタブ番号と指定した修飾キーの組み合わせの 1 つによりタブの序数でタブを選択できます。番号 9 は最後のタブの選択に使用できます。Emacs がサポートする任意の修飾キー選択できます。Section 33.3.7 [Modifier Keys], page 523 を参照してください。別の変数 tab-bar-tab-hintsをカスタマイズして、タブ名とともにタブ番号を表示できます。これは番号でタブを選択するためにどの数字キーを押下すれば判断する助けになるでしょう。

modifier-9
最後に使用したタブに切り替えます (tab-last)。キーの組み合わせは tab-bar-select-tab-modifiersで指定した修飾キーと 9です。数引数 *n*では最後から *n*番目に使用したタブに切り替えます。

modifier-0
最近使用したタブに切り替えます (tab-recent)。キーの組み合わせは tab-bar-select-tab-modifiersで指定した修飾キーと 0です。数引数 *n*では *n*番目に最近使用したタブに切り替えます。

以下のコマンドはタブを操作するために使用されます:

C-x t r *tabname* RET
カレントタブを *tabname*りリネームします (tab-rename)。

C-x t m カレントタブの位置を 1 つ右に移動します (tab-move)。正の数引数 *n*ではその回数分右、負の引数 −*n*では *n*個左の位置へ移動します。

タブの操作にマウスを使うことができます。mouse-2のクリックでタブのクローズ、mouse-3のクリックによりクリックしたタブを操作するアイテムを含んだコンテキストメニューをポップアップ、mouse-1でタブをドラッグすることによってタブバー上の別の位置にタブを移動できます。マウスホイールをスクロールすると次、または前のタブに切り替わります。スクロール中に SHIFTキーを押下すればタブが左、または右に移動します。

すべてのタブに使用されたウィンドウ構成の記録と後でリストアするために tab-bar-history-modeを有効にできます。

`M-x tab-bar-history-back`
: カレントタブで前に仕様されたウィンドウ構成をリストアします。これによりウィンドウ構成のヒストリーを戻ります。

`M-x tab-bar-history-forward`
: 前のウィンドウ構成のリストアを取り消します。これはウィンドウ構成のヒストリーを前に移動します。

ユーザーオプション `tab-bar-format`でタブバー上に表示されるアイテムをカスタマイズできます。

18.18 ダイアログボックスの使用

ダイアログボックスとは、yes-or-no の質問 (はい／いいえで応える問いかけ) をしたり、他の何か特別な質問を問いかけるための、特別な種類のメニューです。コマンドを呼び出すためにマウスを使い、それがユーザーへの質問を要するような操作の場合、多くの Emacs コマンドは yes-or-no を問うために、ダイアログボックスを使います。

ダイアログボックスの使用を無効にするには、変数 `use-dialog-box`を `nil`に変更します。この場合、Emacs はエコーエリアとキーボード入力を使って yes-or-no プロンプトを処理します。この変数はファイル選択ウィンドウの使用も制御します (しかしこれらはすべてのプラットフォームでサポートされているわけではありません)。

ファイル選択ウィンドウはファイル名を問うための、特別な種類のダイアログボックスです。変数 `use-file-dialog`をカスタマイズすれば、他の種類のダイアログボックスを使用して、ファイル選択ウィンドウを使用しないようにできます。変数 `use-dialog-box`ですべてのダイアログボックスを使用しないようにしている場合、この変数は効果がありません。

Emacs が GTK+サポートつきでコンパイルされている場合、Emacs は GTK+のファイル選択ダイアログを使います。Emacs は非表示のファイル (名前がドットで始まるファイル) の表示の有効・無効を切り替えるボタンを、ダイアログボックスに追加します。この切り替えをデフォルトで有効にしたい場合、変数 `x-gtk-show-hidden-files`を `t`に変更します。さらに Emacs は、GTK+ファイル選択ダイアログにヘルプテキストを追加します。このヘルプテキストを無効にするには、変数 `x-gtk-file-dialog-help-text`を `nil`に変更してください。

18.19 ツールチップ

ツールチップは、小さい特別なフレームに、マウスのカレント位置に関するテキスト情報を表示します。ツールチップはウィンドウの重要なテキストの上や、ツールバーのボタンやメニューアイテムのような、Emacs フレームの他の部分の上でマウスを停止させたときにアクティブになります。

ツールチップの使用は、コマンド `M-x tooltip-mode`で切り替えることができます。Tooltip モードが無効な場合、ヘルプテキストは、かわりにエコーエリアに表示されます。ツールチップの使用を起動時に制御するには、変数 `tooltip-mode`をカスタマイズしてください。

以下の変数はツールチップ表示にたいするカスタマイズオプションを提供します:

`tooltip-delay`
: この変数は、最初のツールチップを表示する前に、Emacs が wait する長さを指定します。値は秒です。

`tooltip-short-delay`
: この変数は、すでに最初のツールチップが表示されているとき、異なるアイテムにたいする後続のツールチップを表示する前に、Emacs が wait する長さを指定します。値は秒です。

`tooltip-hide-delay`
: マウスが静止しているとき、ツールチップを隠すまでの秒数です。

`tooltip-x-offset`
`tooltip-y-offset`
: マウスポインター位置から、ツールチップの左上隅までの X オフセットと Y オフセットを、ピクセルで指定します。`tooltip-frame-parameters`が、`left`および`top`パラメーターをインクルードするためにカスタマイズされている場合、これらは無視されることに注意してください。オフセットには、ツールチップがマウスのクリックと干渉するような、マウスポインターのホットスポットを覆い隠さない値を指定するべきです。

`tooltip-frame-parameters`
: ツールチップの表示のために、フレームパラメーターが使用されます。Section “Frame Parameters” in *The Emacs Lisp Reference Manual*、および Section “Tooltips” in *The Emacs Lisp Reference Manual* を参照してください。

ツールチップの表示にたいする追加のカスタマイズオプションについては、`M-x customize-group RET tooltip RET`を使用してください。

Emacs が GTK+ツールキット、あるいは Nextstep や Haiku のウィンドウサポートつきでビルドされていれば、それらのツールキットがもつツールチップのデフォルトの外観を用いて、ツールキットを通じたツールチップ表示ができます [1]。無効にする場合には、変数 `use-system-tooltips`を `nil` に変更してください。無効にした場合や、適切なウィンドウサポートなしで Emacs がビルドされていた場合には、ツールチップテキストの属性のほとんどが `tooltip`、および X リソースによって指定されることになります (Appendix D [X Resources], page 591 を参照)。

GUD ツールチップは、プログラムを GUD でデバッグしているときに変数の値を表示する、特別なツールチップです。Section 24.6.2 [Debugger Operation], page 316 を参照してください。

18.20 マウスの回避

グラフィカルな端末では、マウスポインターが Emacs フレームのテキストを隠してしまうことがありえます。この問題を避けるために、Emacs は 2 つの方法を提供します。

1 番目の方法は、ユーザーが自己挿入文字をタイプしたとき、Emacs がマウスポインターを非表示にして、マウスポインターを動かしたときに、ポインターが Emacs フレームの内側にあるときは、再びポインターを表示する方法です。この機能を無効にするには、変数 `make-pointer-invisible` を `nil`にセットしてください。Section 11.24 [Display Custom], page 102 を参照してください。

2 番目の方法は Mouse Avoidance(マウス回避) モードを使って、マウスポインターをポイントから遠ざける方法です。Mouse Avoidance モードを使うには、変数 `mouse-avoidance-mode`をカスタマイズします。この変数にさまざまな値をセットすることにより、マウスを遠ざけるいくつかの方法を選択できます。

`banish`
: キーが押されたらポインターをフレームの隅に移動します。ポインターをどこに遠ざけるかは、変数 `mouse-avoidance-banish-position`でカスタマイズできます。

`exile`
: カーソルがポインターに近づきすぎたときだけポインターを遠ざけて、カーソルが離れたらポインターを元に戻します。

[1] Nextstep ではツールキットが作成するツールチップのフォアグラウンドとバックグラウンドのカラーは、`tooltip-frame-parameters`に含まれるフレームパラメーター `foreground`と `background`をセットしてカスタマイズすることもできます

`jump` カーソルがポインターに近づきすぎたら、ランダムな方向と距離にポインターを移動します。

`animate` `jump`と同様ですが、移動モーションをアニメ化します。

`cat-and-mouse`
`animate`と同じです。

`proteus` `animate`と同様ですが、マウスポインターの外見も変更します。

コマンド `M-x mouse-avoidance-mode`を使って、このモードを有効にすることもできます。Mouse Avoidance モードがマウスを移動したときは、常にそのフレームを前に表示します。

18.21 非ウィンドウ端末

テキスト端末では、Emacs は 1 度に 1 つの Emacs フレームしか表示できません。それでも複数の Emacs フレームを作成して、それらを切り替えることができます。これらの端末でフレームを切り替えるのは、異なるウィンドウの設定を切り替えるのによく似ています。

`C-x 5 2`を使うと新しいフレームを作成してそれに切り替えます。`C-x 5 o`を使うと既存のフレームを巡回します。`C-x 5 0`を使うと、カレントフレームを削除します。

各フレームには区別するための番号があります。端末が 1 度に 1 つのフレームしか表示できない場合、選択されたフレームの番号 *n*が、モードラインの先頭に近い位置に、'`Fn`'という形式で表示されます。

'`Fn`'は、フレームの実際の初期名称です。フレームにもっと意味のある名前を与えて、その名前でフレームを選択できます。コマンド `M-x set-frame-name RET` *name* `RET`を使うと、選択されたフレームに新しい名前を指定し、`M-x select-frame-by-name RET` *name* `RET`を使うとその名前に一致するフレームを選択します。指定した名前は、そのフレームが選択されたときに、モードラインに表示されます。

18.22 テキスト端末でのマウスの使用

端末ウィンドウでのマウスクリックをサポートするテキスト端末がいくつかあります。

`xterm`と互換性のある端末エミュレーターでは、`M-x xterm-mouse-mode`を使って、Emacs に簡単なマウスの使用 — 基本的には修飾なしのシングルクリックだけがサポートされます — を制御させることができます。より新しいバージョンの `xterm`は、マウストラッキングもサポートします。そのようなクリックにたいする通常の `xterm`のマウス機能は、マウスボタンを押すときに `SHIFT`キーを押すことにより、利用できます。Xterm Mouse モードはグローバルマイナーモードです (Section 20.2 [Minor Modes], page 241 を参照してください)。コマンドを繰り返すと、このモードを再びオフにします。

GNU/Linux のコンソールでは、`M-x gpm-mouse-mode`を使ってマウスサポートを有効にできます。これが機能するためには gpm サーバーがインストールされていて、システムで実行されていなければなりません。このモードが有効な場合、Emacs と GPM を使用する他のプログラムとの間で、マウスによるテキスト転送ができないことに注意してください。これは GPM と Linux kernel による制限です。

MS-DOS でのマウスサポートに関する情報は、Section "MS-DOS Mouse" in *Specialized Emacs Features* を参照してください。

19 国際化文字セットのサポート

Emacsは、広範囲な国際化文字セット (international character sets) をサポートします。それらには、ラテンアルファベットの変種である European と Vietnamese、同様に Arabic scripts、(Bengali、Hindi、Thai のような言語にたいする)Brahmic scripts 、Cyrillic、Ethiopic、Georgian、Greek、(Chinese と Japanese にたいする)Han、(Korean にたいする)Hangul、Hebrew、IPA が含まれます。Emacs は他の国際化されたソフトウェアー (ワープロやメーラー) などで使われる、それらの文字にたいするさまざまなエンコーディングもサポートします。

Emacs は関連するアクティビティーのすべてをサポートすることにより、国際化文字セットの編集を可能にします:

- 非 ASCII文字のファイルを visit したり、非 ASCIIのテキストを保存したり、非 ASCIIのテキストを、Emacs と Emacs が呼び出すプログラム (コンパイラー、スペルチェッカー、メーラーなど) に引き渡すことができます。言語環境のセッティングとは、コーディングシステムのセッティングと、その他の言語に特有な文化のためのオプションを処理することです。かわりに各コマンドにエンコードあるいはデコードする方法を指定できます。Section 19.9 [Text Coding], page 227 を参照してください.
- さまざまなスクリプトでエンコードされた、非 ASCII文字を表示することができます。これはグラフィカルなディスプレー上で適切なフォントを使うこと (Section 19.15 [Defining Fontsets], page 233 を参照してください)、そしてテキスト表示のために特別なコードを送信すること (Section 19.13 [Terminal Coding], page 231 を参照してください) により機能します。正しく表示できない文字があるときは、Section 19.17 [Undisplayable Characters], page 235 を参照してください。これには考えられる原因と、解決方法が記述されています。
- 本来、右から左に記述されるスクリプトの文字は、表示のために再配置されます (Section 19.20 [Bidirectional Editing], page 238 を参照)。これらのスクリプトには Arabic、Hebrew、Syriac、Thaana、それ以外にもいくつか存在します。
- 非 ASCII文字を挿入したり検索することができます。これを行うために、言語にあった Emacs のインプットメソッド (IM: input method。Section 19.4 [Select Input Method], page 222 を参照) を指定するか、言語環境を選択したときにセットアップされた、デフォルトのインプットメソッドを使うことができます。キーボードが非 ASCII文字を生成できる場合、適切なキーボードコーディングシステムを選択できます。Emacs はそれらの文字を受け入れることができるでしょう。グラフィカルなディスプレイ上の現代的なシステムでは、通常だとネイティブなインプットメソッドを提供しており、Latin-1 文字は `C-x 8`プレフィクスを使って入力することもできます。Section 19.18 [Unibyte Mode], page 236 を参照.

  X ウィンドウシステムでは、Emacs がキーボード入力を正しく解釈するために、locale に適切な値をセットする必要があります。Section 19.2 [Language Environments], page 218 および Section 19.12 [X Coding], page 230 を参照してください。

このチャプターの残りの部分では、これらの問題について詳細を説明します。

19.1 国際化文字セットのイントロダクション

国際化文字セットとスクリプトのユーザーは、ファイルを保存するために、多少の差はありますが、標準化された多くのコーディングシステムを確立しています。これらのコーディングシステムは通常はマルチバイト *(multibyte)* で、これは 1 つの非 ASCII文字を表すのに、2 つ以上のバイトシーケンスを対応させることを意味します。

Emacs は、内部的には *Unicode* 標準のスーパーセットである、マルチバイト文字エンコーディングを使用します。この内部的なエンコーディングは、ほとんどすべての既知のスクリプトを、1 つの

バッファーまたは文字列に混成することを可能にします。Emacs はファイルを読み書きしたり、サブプロセスとデータをやりとりするとき、このマルチバイト文字エンコーディングと、他のさまざまなコーディングシステムをコード変換します。

コマンド `C-h h` (`view-hello-file`) は、ファイル `etc/HELLO`を表示します。これは、多くの異なる言語で、“hello” をどのように記述するかを、さまざまな文字で例示するファイルです。もしもある文字が端末で表示できないときは、それらの文字は ‘?’か、中抜きのボックスで表示されます。

これらの文字セットを使う国のキーボードでも、一般的にはすべての文字に対応するキーはもっていません。キーボードがサポートしない文字は、`C-x 8 RET` (`insert-char`) を使って挿入することができます。Section 4.1 [Inserting Text], page 17 を参照してください。一般的な文字のいくつかは略記が利用できます。たとえば `C-x 8 [`とタイプ (Electric Quote モードでは、通常は単に `` ` ``とタイプ) することにより、left single quotation mark ‘ を挿入できます。Emacs はさまざまなインプットメソッド *(IM: input methods)* をサポートします。これはあるスクリプトの文字をタイプするのを簡単にするもので、通常 1 つのスクリプトまたは言語に 1 つです。Section 19.3 [Input Methods], page 220 を参照してください。

プレフィクスキー `C-x RET`は、マルチバイト文字、コーディングシステム、インプットメソッドに属するコマンドにたいして使用されます。

コマンド `C-x =` (`what-cursor-position`) は、ポイント位置にある文字の情報を表示します。文字の位置に加えて、Section 4.9 [Position Info], page 24 で説明したように、このコマンドはその文字がどのようにエンコードされているかを表示します。たとえば、このコマンドは文字 ‘`c`’に大して、以下のような行をエコーエリアに表示します:

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

‘`Char:`’の後ろの 4 つの値は、ポイント位置の文字を説明するためのもので、1 つ目はその文字自身、その後ろに文字コードを 10 進 (decimal)、8 進 (octal)、16 進 (hex) で表示します。非 ASCII のマルチバイト文字の場合、バッファーのコーディングシステムでその文字を安全に 1 バイトでエンコードできる場合は、‘`file`’とそのバッファーのコーディングシステムで表した文字コードの 16 進表記が続きます。その文字のエンコーディングが 1 バイトより長い場合、Emacs は ‘`file ...`’と表示します。

Emacs が *raw* バイト *(raw bytes)* に遭遇する場合がまれにあります。raw バイトとは、既知の非 ASCII 文字エンコーディングとして Emacs が解釈できない、値 128(8 進 0200) から 255(8 進 0377) の範囲の単一バイトのことです。そのような raw バイトは、特別な文字セット `eight-bit`に属すものとして扱われます。Emacs はそれらをエスケープされた 8 進コードで表示します (これはカスタマイズ可能。Section 11.24 [Display Custom], page 102 を参照されたい)。この場合、`C-x =`は ‘`file`’のかわりに ‘`raw-byte`’を示します。加えて、`C-x =`はそれらがあたかも Emacs が範囲`#x0080..#x00FF`のユニコード文字から区別するためにマップする範囲`#x3FFF80..#x3FFFFF`にあるかのように raw バイトの文字コードを表示します。

プレフィクス引数を指定した (`C-u C-x =`) では、その文字の詳細な情報をウィンドウに表示する`describe-char`コマンドを追加で呼び出します:

- 文字セット名 (character set name) と、文字セットでその文字が識別されるコード。ASCII文字の場合、`ascii`文字セットに属すると識別されます。
- その文字のスクリプト (script)、構文 (syntax)、カテゴリー (categories)。
- 現在のインプットメソッドで (もしその文字をサポートしていれば)、その文字を入力するためにタイプするキー。
- その文字のエンコード。バッファー内部のエンコード、およびバッファーをファイルに保存する際の外部のエンコードの両方。

- グラフィカルなディスプレイで Emacs を実行しているときはフォント名と、その文字にたいするグリフコード (glyph code)。Emacs をテキスト端末で実行している場合、端末に送るコード。
- 1 つ以上の書記素クラスタ (grapheme cluster) を形成するためにその文字が後続の文字と合成される場合には合成情報 (グラフィカルなディスプレイのフレームならフォントグリフおよび合成文字)。
- その文字のテキストプロパティ (Section "Text Properties" in *the Emacs Lisp Reference Manual* を参照してください)。これにはその文字を表示するのに使われるデフォルト以外のフェイスと、それを含むオーバーレイ (overlays) が含まれます (Section "Overlays" in *the same manual* を参照してください)。

以下は例です (マニュアルに収まるように折り返している行もあります):

```
              position: 1 of 1 (0%), column: 0
             character: ^^c3^^aa (displayed as ^^c3^^aa) (codepoint 234, #o352, #xea)
    preferred charset: unicode (Unicode (ISO10646))
code point in charset: 0xEA
               script: latin
               syntax: w        which means: word
             category: .:Base, L:Left-to-right (strong), c:Chinese,
                       j:Japanese, l:Latin, v:Viet
             to input: type "C-x 8 RET ea" or
                       "C-x 8 RET LATIN SMALL LETTER E WITH CIRCUMFLEX"
          buffer code: #xC3 #xAA
            file code: #xC3 #xAA (encoded by coding system utf-8-unix)
              display: by this font (glyph code)
    xft:-PfEd-DejaVu Sans Mono-normal-normal-
        normal-*-15-*-*-*-m-0-iso10646-1 (#xAC)

Character code properties: customize what to show
  name: LATIN SMALL LETTER E WITH CIRCUMFLEX
  old-name: LATIN SMALL LETTER E CIRCUMFLEX
  general-category: Ll (Letter, Lowercase)
  decomposition: (101 770) ('e' '^')
```

19.2 言語環境

サポートされているすべての文字セットは、マルチバイト文字が利用可能なときは、Emacs バッファーの中でサポートされます。その文字を表示するために、特定の言語を選択する必要はありません。しかしさまざまなデフォルト値をセットするために、言語環境 *(language environment)* を選択することは重要です。大まかに言うと、言語環境とは、言語の選択というよりも、好ましいスクリプト選択の提示です。

言語環境は、テキストを読み込むとき、それを認識するコーディングシステムを制御します (Section 19.6 [Recognize Coding], page 225 を参照してください)。これはファイル、到着メール、その他の Emacs で読む任意のテキストに適用されます。これは新しくファイルを作成するときに使う、デフォルトのコーディングシステムも指定します。それぞれの言語環境は、デフォルトのインプットメソッドも指定します。

言語環境を選択するには、`current-language-environment`をカスタマイズするか、コマンド `M-x set-language-environment`を使います。このコマンドを使うとき、どのバッファーがカレントかで違いは生じません。なぜなら、その効果は Emacs セッションにグローバルで適用されるからです。サポートされている言語環境の一覧は、変数 `language-info-alist`を参照してください。コマンド `C-h L` *lang-env* `RET` (`describe-language-environment`) を使うと、言語環境 *lang-env* の、より詳細な情報が参照できます。サポートされる言語環境には、以下が含まれます:

ASCII、Arabic、Belarusian、Bengali、Brazilian Portuguese、Bulgarian、Burmese、Cham、Chinese-BIG5、Chinese-CNS、Chinese-EUC-TW、Chinese-GB、Chinese-GB18030、Chinese-GBK、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、Latin-9、Latvian、Lithuanian、Malayalam、Oriya、Persian、Polish、Punjabi、Romanian、Russian、Sinhala、Slovak、Slovenian、Spanish、Swedish、TaiViet、Tajik、Tamil、Telugu、Thai、Tibetan、Turkish、UTF-8、Ukrainian、Vietnamese、Welsh、Windows-1255

グラフィカルなディスプレイでは、使用されている言語環境で使われているスクリプトを表示するために、適切なフォントをもっている必要があります。フォントのセットアップについては、Section 19.14 [Fontsets], page 232 を参照してください。

環境変数 `LC_ALL`、`LC_CTYPE`、`LANG`をセットすることにより、使用する文字セットの locale を指定するオペレーティングシステムがいくつかあります (もしこれらの 1 つ以上がセットされている場合、特に 1 番目のものが空でない場合、それは正にこの目的のために locale を指定しています)。起動の間、Emacs は文字セットの locale 名を、システムの locale エイリアステーブルから探して、その正規化された名前 (canonical name) を、変数`locale-charset-language-names`および`locale-language-names`(前者は後者をオーバーライドします) のエントリーにたいしてマッチし、マッチが見つかったら対応する言語環境を選択します。これはディスプレイテーブルと端末のコーディングシステム、locale コーディングシステム、locale に必要な好ましいコーディングシステム— そして最後に重要なのは — Emacs がキーボードから送られた非 ASCII文字をデコードする方法を調整します。

Emacs 実行中に、(`M-x setenv`を使って) 環境変数`LC_ALL`、`LC_CTYPE`、`LANG`を変更した場合、新しい locale にたいする言語環境を再調整するために、後で`set-locale-environment`コマンドを呼び出したいと思うかもしれません。

`set-locale-environment`関数は、通常はシステムメッセージをデコードするために、言語環境により確立された優先コーディングシステムを使用します。しかし locale が変数`locale-preferred-coding-systems`のエントリーにマッチした場合には、Emacs はかわりに対応するコーディングシステムを使用します。たとえば locale の‘`ja_JP.PCK`’が`locale-preferred-coding-systems`の`japanese-shift-jis`にマッチすれば、通常なら`utf-8`が使われるような場合でも、Emacs はエンコーディングにそのコーディングシステムを使用します。

init ファイルで明示的にコマンド`set-language-environment`を使うか、`current-language-environment`をカスタマイズすることにより、起動時に選択された言語環境をオーバーライドできます。

特定の言語環境 *lang-env*の効果に関する情報を表示するには、コマンド `C-h L` *lang-env* `RET` (`describe-language-environment`) を使います。これはこの言語環境に有効な言語、文字セットのリスト、コーディングシステム、インプットメソッドを表示します。これはこの言語環境で使われるスクリプトを例示する、サンプルテキストも表示します。*lang-env*に空の入力を与えると、このコマンドは選択されている言語環境を説明します。

ノーマルフック`set-language-environment-hook`により、任意の言語環境をカスタマイズできます。コマンド`set-language-environment`は、新しい言語環境をセットアップした後に、このフックを実行します。フック関数は変数`current-language-environment`をチェックすることにより、特定の言語環境をテストできます。このフックはキーボード入力にたいするコーディングシステ

ムや端末出力、デフォルトのインプットメソッドなど、特定の言語環境にたいして非デフォルトのセッティングが必要な場所に設定します

新しい言語環境のセットアップを開始する前に、set-language-environmentはまずフック exit-language-environment-hookを実行します。このフックは set-language-environment-hookによるカスタマイズを取り消すのに便利です。たとえば、特定の言語環境にたいして set-language-environment-hookを使って特別なキーバインドをセットアップした場合は、exit-language-environment-hookで通常のキーバインドに復元するべきです。

19.3 インプットメソッド

インプットメソッド *(input method)* とは、対話的入力に特化してデザインされたある種の文字変換のことです。このセクションではEmacsに同梱されているインプットメソッドについて説明します。背後にあるOSが提供するインプットメソッドについては、Section 19.18 [Unibyte Mode], page 236を参照してください。

Emacsでは、各言語はそれぞれ独自のインプットメソッドをもっています。同じ文字を使う複数の言語で、1つのインプットメソッドを共有できます。複数のインプットメソッドをサポートする言語もあります。

一番簡単な種類のインプットメソッドは、ASCII文字を他のアルファベットにマッピングすることにより機能します。これによりASCIIのかわりに他のアルファベットを使うことが可能になります。GreekとRussianのインプットメソッドはこの方式で機能します。

より強力なテクニックがコンポジション (composition: 複合) です。これは文字のシーケンスを1つの文字に変換します。Europeanのインプットメソッドの多くが、アクセント文字を後 (または前) に続けた文字シーケンスから、1つの非ASCII文字を生成するためにコンポジションを使います。たとえば、インプットメソッドのいくつかは、o ^のシーケンスを1つのアクセントつき文字に変換します。これらのインプットメソッドは、それら自身では特別なコマンドをもちません。これらすべてが行うのは、文字シーケンスを複合して、プリント文字にすることです。

音節記号 (syllabic scripts) のためのインプットメソッドは通常、マッピングと、それに続けてコンポジションを使います。ThaiとKoreanのためのインプットメソッドは、この方式で機能します。最初に複数の文字が、特定の音や口調のためのシンボルにマッピングされます。次にこれらシンボルのシーケンスから音節全体を作り、それを1つの音節記号にマッピングします。

ChineseとJapaneseには、さらに複雑な方式が必要です。Chineseのインプットメソッドでは、最初にChineseの単語の音声スペルを入力するか (特にインプットメソッド chinese-py)、文字の一部をシーケンスとして入力します (インプットメソッド chinese-4corner、chinese-swなど)。通常1つの入力シーケンスは、多くのChinese文字に対応します。C-f、C-b、C-n、C-p(または矢印キー)、またはこの状況では特別な意味をもつ数字を指定することにより、意図するものを選択します。

文字の候補は、概念的には複数の行にアレンジされ、各行は10個の候補をもちます。通常Emacsは1度に1行をエコーエリアに表示します。行頭に (*i*/*j*)が表示され、これはトータル *j*行中、*i*番目の行かを示します。C-nまたはC-pをタイプすると、次または前の行を表示します。

C-fまたはC-bをタイプすると、カレント行の候補の間を前方または後方に移動します。これを行うとき、Emacsはカレント候補を特別な色でハイライトします。C-SPCとタイプすると、カレント候補を選択して、それを入力に使用します。各行の候補には番号も付けられています。この番号は各候補の前に表示されます。番号をタイプすると、カレント行の番号に関連付けられた候補を選択し、それを入力に使用します。

これらChineseのインプットメソッドでは、TABはすべての文字候補をバッファーに表示します。候補の1つを mouse-2でクリックすることにより選択します。C-f、C-b、C-n、C-p、および数字

キーは通常どおり機能しますが、それらはエコーエリアではなく、文字候補を表示したバッファーをハイライトします。

かわりに *pīnyīn* 変換メソッドに応じた文字を入力するには、入力メソッド `chinese-sisheng`を使用します。これは合成ベースのメソッドであり、`pi1`が 'pī'になります。

Japanese のインプットメソッドでは、最初に音声スペルを使って単語全体を入力します。つぎに単語がバッファーに入った後で、より大きな辞書を使って Emacs がそれを 1 つ以上の文字に変換します。1 つの音声スペルは、いくつかの Japanese の単語に対応します。これらの 1 つを選択するには、`C-n`または `C-p`を使って候補を巡回します。

インプットメソッドをオフにして、入力した文字シーケンスが複合されないようにするのが便利なときがあります。たとえばインプットメソッド `latin-1-postfix`では、シーケンス `o ^`は、アクセントつきの 'o'に複合されます。これらの文字を個別に入力したいときはどうすればよいでしょう?

1 つは、アクセントを 2 度タイプする方法です。これは文字とアクセントを個別に入力するための特別な機能です。たとえば `o ^ ^`により、2 つの文字 'o^'が得られます。他の方法としては `o`の後に別の文字 — 複合されない何か別の文字 — を入力してすぐにそれを削除する方法です。たとえば `o o DEL ^`とタイプすることにより、'o'と '^'を個別に得ることができます。もう 1 つは、より一般的ですがタイプが容易とは言えない方法で、2 つの文字が欠号されるのを防ぐために、文字の間で `C-\\ C-\\` を使用する方法です。これは、コマンド `C-\\`(`toggle-input-method`) を 2 回使用しています。

`C-\ C-\`は、インクリメンタル検索の中で使うのが特に便利です。なぜならこれは複合される文字が入力されるのを待つのを止めて、それまでに入力した文字で検索を開始するからです。

現在のインプットメソッドを使って、ポイント位置の後ろの文字を入力する方法を探すには、`C-u C-x =`をタイプします。Section 4.9 [Position Info], page 24 を参照してください。

変数 `input-method-highlight-flag`および `input-method-verbose-flag`は、インプットメソッドで何が起きているかを告げる方法を制御します。`input-method-highlight-flag`が非 `nil` の場合、部分的な入力シーケンスがバッファーでハイライトされます (この機能を無効にしているインプットメソッドもあります)。`input-method-verbose-flag`が非 `nil`の場合、次にタイプできる文字の一覧をエコーエリア (ただしミニバッファーにいるときは除く) に表示します。

フック変数 `quail-activate-hook`に関数で変更を行うことにより、入力メソッドが機能する方法を変更できます。Section 33.2.2 [Hooks], page 509 を参照してください。たとえば、関数 `quail-translation-keymap`によりリターンされるキーマップ内のキーバインディングを、`define-key`を使用して、定義することによりその入力メソッドのいくつかのキーを再定義できます。Section 33.3.6 [Init Rebinding], page 522 を参照してください。

何らかの理由でバッファーのテキストが読み取り専用の際には、入力メソッドは抑制されます。これは `read-only-mode`や `image-mode`のようなバッファーテキストやその一部が読み取り専用なモードにおいて、たとえば入力メソッドがアクティブでも単一文字のキーバインディングを機能させるためです。

キーボードにない文字をタイプする他の方法は、`C-x 8 RET` (`insert-char`) を使って、Unicode 名またはコードポイント (code-point) にもとづいて 1 つの文字を挿入する方法です。Section 4.1 [Inserting Text], page 17 を参照してください。

絵文字 (Emoji) の挿入用に特化したコマンドがあり、これらはキーマップ `C-x 8 e`で見つけることができます。`C-x 8 e e` (`emoji-insert`) は絵文字の異なるカテゴリー間を移動して 1 つのカテゴリーを選ぶためのコマンドです。`C-x 8 e l` (`emoji-list`) は新たなバッファーをポップアップしてすべての絵文字をリストします。Emoji 文字をクリック (または `RET`を使用) することによって、その絵文字がカレントバッファーに挿入されます。最後に `C-x 8 e s` (`emoji-search`) は名前にもとづいて絵文字の検索を行うコマンドです。

describe-charはポイントの下にある文字/グリフ (絵文字を含む) に関する多くの情報を表示します。名前の説明を手早く得られれば便利なときがあるでしょう。これを行うには C-x 8 e d (emoji-describe) コマンドを使用します。これは主として絵文字の異なるバリエーション (非常に似た外観をもつかもしれない) を区別する助けとなることを意図したコマンドですが、非 Emoji 文字の名前についても教えてくれるでしょう。

19.4 インプットメソッドの選択

C-\ 選択されたインプットメソッドを有効または無効にします (toggle-input-method)。

C-x RET C-\ *method* RET
カレントバッファーにたいして、新しいインプットメソッドを選択します (set-input-method)。

C-x \ *method* RET
選択された一時入力メソッド (transient input method) を一時的に有効にします (activate-transient-input-method)。これは単一文字の挿入後に無効になります

C-h I *method* RET
C-h C-\ *method* RET
インプットメソッド *method*の説明を表示します (describe-input-method)。デフォルトでは、(もしあれば) カレントのインプットメソッドを説明します。これは特定のインプットメソッドの使い方に関する、すべての詳細説明を表示します。

M-x list-input-methods
サポートされている、すべてのインプットメソッドのリストを表示します。

カレントバッファーにたいするインプットメソッドを選択するには、C-x RET C-\ (set-input-method) を使います。このコマンドはミニバッファーからインプットメソッドの名前を読み取ります。この名前は通常、それが使われることを意図した言語環境で開始されます。変数 current-input-methodは選択されたインプットメソッドを記録します。

インプットメソッドは非 ASCII文字を表すために、さまざまな ASCII文字のシーケンスを使います。インプットメソッドを一時的にオフにできると便利なときもあります。そのようなときは C-\ (toggle-input-method) をタイプします。インプットメソッドを再度有効にするには、もう 1 度 C-\をタイプします。

C-\をタイプしたときに、まだインプットメソッドが選択されていない場合、インプットメソッドを指定するように求めます。これはインプットメソッドを指定する C-x RET C-\を使ったときと同じ効果です。

C-u C-\のようにプレフィクス引数を指定した場合、toggle-input-methodは常にインプットメソッドを尋ねます。このときデフォルトとして提案されるのは、もっとも最近選択されたインプットメソッドです。

言語環境の選択により、さまざまなバッファーで使用するデフォルトのインプットメソッドが指定されます。デフォルトのインプットメソッドがある場合、C-\とタイプしてカレントバッファーでそれを選択できます。変数 default-input-methodはデフォルトのインプットメソッドを指定します (nilは、それが存在しないことを意味します)。

複数の異なるインプットメソッドをサポートする言語環境では、set-language-environmentで選択されるデフォルトとは違うインプットメソッドを使いたいときもあるでしょう。set-language-environment-hookを使って、特定の言語環境にたいして異なるデフォルトのインプットメソッドを

使うよう Emacs に指示できます (Section 19.2 [Language Environments], page 218 を参照してください)。たとえば:

```
(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)
```

これは言語環境を Chinese-GB language に選択したときは、常にデフォルトのインプットメソッドを `chinese-tonepy`にセットします。

特定のインプットメソッドを自動的にアクティブにするよう Emacs に指示できます。たとえば:

```
(add-hook 'text-mode-hook
  (lambda () (set-input-method "german-prefix")))
```

これは Text モードで自動的にインプットメソッド `german-prefix`をアクティブにします。

英文字スクリプトのためのいくつかのインプットメソッドは、それらのスクリプトで一般的に使用されているさまざまなキーボードエミュレートするために、(実質的には) 他のアルファベットに再マッピングすることにより機能します。この再マッピングがどのように正しく行われるかは、実際のキーボードレイアウトに依存します。キーボードがどのレイアウトなのかを指定するには、コマンド `M-x quail-set-keyboard-layout`を使います。

コマンド `M-x quail-show-key`を使って、ポイントの後ろにある文字を入力するために、選択されたキーボードレイアウトの、どのキー (またはキーシーケンス) をタイプすればよいのか表示できます。コマンド `C-u C-x =`もこの情報と、それに加えてその文字に関する他の情報を表示します。

`M-x list-input-methods`は、サポートされているすべてのインプットメソッドを一覧します。この一覧は各インプットメソッドの情報と、モードラインに表示される文字列を表示します。

文字を 1 つだけ挿入するために、入力メソッドを一時的 *(transiently)* に有効にできれば便利なときがあるかもしれません。`C-x \` (`activate-transient-input-method`) とタイプすることにより、入力メソッドのルールを使用して単一文字を入力して、その後に自動的に入力メソッドを無効にできます。まだ一時入力メソッド (transient input method) が選択されていなければ、`C-x \`は入力メソッドの入力を求めて、それ以降のコマンド呼び出しでは選択した一時入力メソッドが有効になります。違う一時入力メソッドを選択するには `C-u C-x \`とタイプしてください。`C-u C-\`を使用して選択した一時入力メソッドとは異なる一時入力メソッドを選択できます。

19.5 コーディングシステム

さまざまな言語のユーザーは、多少の差はあれ、それらを表示するための標準のコーディングシステムを確立しています。Emacs はこれらのコーディングシステムを、内部的に使用しません。データを読み込むときは、さまざまなコーディングシステムから Emacs 独自のコーディングシステムに変換し、データを書き込むときには、内部コーディングシステムから他のコーディングシステムに変換します。ファイルの読み書き、端末とのやりとり、サブプロセスとのデータ交換において、変換が可能です。

Emacs は各コーディングシステムに名前を割り当てます。ほとんどのコーディングシステムは、1 つの言語で使用され、コーディングシステムの名前は、言語の名前で始まります。複数の言語で使用されるコーディングシステムもあります。これらのコーディングシステムの名前は、通常 '`iso`'で始まります。`no-conversion`、`raw-text`、`emacs-internal`のような特別なコーディングシステムもあります。

まとめてコードページ *(codepages)* として知られる、特別なクラスのコーディングシステムは、MS-Windows および MS-DOS のソフトウェアーによりエンコードされたテキストをサポートするためにデザインされています。これらのコーディングシステムの名前は `cp`*nnnn*という形式で、*nnnn* は 3 桁から 4 桁のコードページ番号です。これらのコーディングもほかのコーディングシステムと同様に使うことができます。たとえばコードページ 850 でエンコードされたファイルを visit するには、`C-x RET c cp850 RET C-x C-f` *filename* `RET`とタイプします。

非 ASCII文字のさまざまな表現の変換に加えて、コーディングシステムは行末変換 (end-of-line conversion) も行います。Emacs は、ファイル内の行の区切り方として、3 つの異なる変換を扱います。つまり、改行 (Unix)、復帰改行 (DOS)、復帰 (Mac) です。

`C-h C` *coding* `RET`
: コーディングシステム *coding*の説明を表示します (`describe-coding-system`)。

`C-h C RET`
: カレントで使用しているコーディングシステム *coding*の説明を表示します (`describe-coding-system`)。

`M-x list-coding-systems`
: サポートされているすべてのコーディングシステムのリストを表示します。

コマンド `C-h C` (`describe-coding-system`) は、特定のコーディングシステムについて、それらのコーディングシステムで規定されている、行末変換も含めた情報を表示します。引数にコーディングシステム名を指定できます。引数が空のときには、さまざまな目的のために選択されている、現在のコーディングシステムの、カレントバッファにたいするものとデフォルトの両方について表示するとともに、コーディングシステムを認識するための優先順位表を表示します (Section 19.6 [Recognize Coding], page 225 を参照してください)。

サポートされているすべてのコーディングシステムのリストを表示するには、`M-x list-coding-systems`とタイプします。表示されるリストは、モードラインに表示される文字も含めて、各コーディングシステムの情報を提供します。

リストに表示される各コーディングシステム — ただし何の変換も行わない `no-conversion`は除く — は、プリントする文字をどのように変換するか、しないかを指定しますが、改行変換については、各ファイル内容にもどづいて決定するので選択をしません。たとえばファイルが行区切りに改行復帰文字を使っているように見えるときは、DOS の改行変換を使います。

リストされた各コーディングシステムは、改行変換を厳密に指定する 3 つの変種があります。

`...-unix`
: 何の改行変換も行いません。ファイルは行区切りに改行文字を使っていると仮定します (これは通常 Unix、GNU システム、macOS で使われている慣習です)。

`...-dos`
: ファイルが行区切りに改行復帰文字を使っていると仮定し、適切な変換を行います (これは通常 Microsoft システムで使われている慣習です [1])。

`...-mac`
: ファイルが行区切りに復帰文字を使っていると仮定し、適切な変換を行います (これはクラシックな Mac OS で使われていた慣習です)。

これらのコーディングシステムの変種は、それらが完全に予測可能なため、簡略化のために `list-coding-systems`の表示からは省略されています。たとえばコーディングシステム `iso-latin-1`は `iso-latin-1-unix`、`iso-latin-1-dos`、`iso-latin-1-mac`という変種をもちます。

[1] これは MIME の '`text/*`'の本体、および他のネットワーク転送のコンテキストでも指定されています。これは Emacs が直接サポートしない SGML リファレンス構文の record-start/record-end とは異なります。

コーディングシステム unix、dos、macは、それぞれ undecided-unix、undecided-dos、undecided-macの別名です。これらのコーディングシステムは改行変換だけを指定し、文字コード変換はテキスト字体から推論されるよう残します

コーディングシステム raw-textは、主にASCIIテキストのファイルに適していますが、ファイルには、非ASCII文字の符号を意味しない127を越えるバイト値が含まれるかもしれません。raw-textでは、Emacsはそれらのバイト値を変更せずにコピーし、カレントバッファーの enable-multibyte-charactersを nilにセットして、それらは適切に解釈されるます。raw-textは、出会ったデータに基づく通常の方法で行末変換を処理し、使用する行末変換を指定する変種も3つもちます。

対照的に、コーディングシステム no-conversionは、いかなる文字コード変換 — 非ASCIIバイト値や行末にたいしても — を行いません。これは、バイナリーファイル、tarファイル、そのまま処理する必要があるその他のファイルを読み書きするのに便利です。これも enable-multibyte-charactersを nilにセットします。

いかなる種類の変換もしないでファイルを編集するもっとも簡単な方法は、M-x find-file-literallyコマンドを使うことです。このコマンドは、no-conversionを使い、ファイルを見る前にファイルの内容を変換するかもしれない、Emacs のその他の機能を抑制します。Section 15.2 [Visiting], page 146 を参照してください。

コーディングシステム emacs-internal(または utf-8-emacs) は、Emacs内部エンコーディングのままで格納された、非ASCII文字を含むファイルであることを意味します。これは出会ったデータに基づいて行末変換を処理し、行末変換の種類を指定する通常の3つの変種を持ちます。

19.6 コーディングシステムの認識

Emacsはテキストを読み込むとき、どのコーディングシステムが使われているか認識しようと試みます。これはファイルの読み込み、サブプロセスからの出力、X選択からのテキストなど、さまざまです。Emacsは大抵の場合— 自分の好みを1度指定しておけば、自動的に正しいコーディングシステムを選択できます。

データにどのバイトシーケンスが出現するかにより、認識あるいは識別されるコーディングシステムもいくつかあります。しかし識別される可能性さえないコーディングシステムもあります。たとえばLatin-1とLatin-2を識別する方法はありません。これらは同じバイト値を異なる意味で使用します。

Emacsはこのようなシチュエーションを、コーディングシステムの優先リストにより処理します。Emacsがファイルを読み込むときは常に、それに使用するコーディングシステムを指定しなければ、Emacsはデータを各コーディングシステムに照らしてチェックし、それを優先順位の上から順に、データに適合するコーディングシステムが見つかるまで続けます。そして、そのコーディングシステムで、ファイル内容が表示できると仮定して変換を行います。

コーディングシステムの優先リストは、選択されている言語環境に依存します (Section 19.2 [Language Environments], page 218 を参照してください)。たとえばFrenchを使うのなら、おそらくEmacsにはLatin-2よりLatin-1を選んでほしいでしょう。Czechを使うなら、おそらくLatin-2のほうがよいでしょう。これが言語環境を指定する理由の1つです。

しかし、コマンド M-x prefer-coding-systemを使って、優先リストの詳細を変更できます。このコマンドはミニバッファーからコーディングシステムの名前を読み取り、それを優先リストの先頭に追加して、他のすべてのものより優先するようにします。このコマンドを数回使うと、使用するごとに優先リストの先頭に1つの要素が追加されます。

iso-8859-1-dosのような、行末変換を指定したコーディングシステムを使うと、iso-8859-1を優先して認識を試み、その際DOSの行末変換を使うことをEmacsに指示することになります。

ファイルにたいして使用するコーディングシステムをファイル名が示していることがあります。変数 file-coding-system-alistは、この対応関係を指定します。このリストに要素を追加する特別な関数は、modify-coding-system-alistです。たとえば、すべての‘.txt’の読み書きに、コーディングシステム chinese-iso-8bitを使用したいなら、つぎの Lisp 式を実行します:

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

1 つ目の引数は file、2 番目の引数はこれを適用するファイルを決定する正規表現、3 番目の引数は、これらのファイルに対して使用するコーディングシステムです。

Emacs はファイルの内容にもとづいて、使用する行末変換の種類を認識します。復帰のみ、あるいは復帰改行のシーケンスだけであれば、対応する行末変換を選択します。変数 inhibit-eol-conversionを非 nilにセットすることにより、行末変換の自動的な使用を抑止できます。これを行うと DOS スタイルのファイルは、バッファー内に可視の‘^M’という文字を表示します。モードラインの左端に目立たないように表示される改行タイプ指示‘(DOS)’より、こちらのほうを好む人もいます。

デフォルトでは、コーディングシステムの自動検知はエスケープシーケンスを検出します。文字シーケンスがエスケープ文字で開始されていて、そのシーケンスが有効な ISO-2022 であれば、それは Emacs にファイルをデコードするエンコーディングに、ISO-2022 を使うことを告げています。

しかし、ファイルの中のエスケープシーケンスを、そのまま読み取りたい場合もあるでしょう。そのような場合、変数 inhibit-iso-escape-detectionを非 nilにセットします。これにより、コード検知はエスケープシーケンスを無視するようになり、ISO-2022 エンコーディングは使用されません。この結果として、すべてのエスケープシーケンスがバッファー内で可視になります。

変数 inhibit-iso-escape-detectionのデフォルト値は nilです。わたしたちは特別な操作を除いて、これを変更しないことを推奨します。なぜなら、Emacs ディストリビューションの Emacs Lisp ソースファイルのいくつかは、コーディングシステム iso-2022-7bitでエンコードされた非 ASCII文字を含んでおり、エスケープシーケンス検知を抑止しているときにこれらのファイルを visit すると、正しくデコードされないからです。

変数 auto-coding-alistおよび auto-coding-regexp-alistは、それぞれファイル名に含まれる特定パターン、およびファイルに含まれる特定パターンによりコーディングシステムを指定する一番強い方法です。これらの変数は、ファイル自身に含まれる‘-*-coding:-*-’タグさえオーバーライドします。たとえば、Emacs は tar およびアーカイブファイルに、auto-coding-alistを使います。これはアーカイブのメンバーファイルに‘-*-coding:-*-’が含まれている場合、Emacs が混乱してそれをファイル全体に適用するのを防ぎます。

コーディングシステムを指定する他の方法は、変数 auto-coding-functionsを使う方法です。たとえばビルトインの 1 つ auto-coding-functionsは、XML ファイルにたいするエンコーディングを検知します。前の 2 つと異なり、この変数は‘-*-coding:-*-’タグをオーバーライドしません.

19.7 ファイルのコーディングシステムの指定

Emacs がファイルのエンコーディングを正しく認識しなかった場合、C-x RET r (revert-buffer-with-coding-system) で、正しいコーディングシステムでファイルを再読み込みできます。このコマンドは、使用するコーディングシステムの入力を求めます。ファイルのデコードに実際に使われているコーディングシステムを見るには、モードラインの左端の近くのコーディングシステムのニーモニック文字を見るか、C-h C (describe-coding-system) をタイプします。

特定のファイルのコーディングシステムを指定するのに、そのファイル自身の最初に‘-*-...-*-’構成を指定するか、ファイルの最後にローカル変数リスト (Section 33.2.4 [File Variables], page 512 を参照してください) を使用できます。これは codingという名前の “変数” に、値を定義することにより行われます。Emacs は実際には変数 codingをもっていません。かわりに変数を

セットして、特定のファイルにたいしてコーディングシステムを指定するのにこれを使います。たとえば '`-*-mode: C; coding: latin-1; -*-`' は、Latin-1 コーディングシステム、同様に C モードを指定することを指示します。ファイルの中でコーディングを明示的に指定した場合、これは`file-coding-system-alist`をオーバーライドします。

19.8 出力のためのコーディングシステムの選択

Emacs がバッファーにたいして 1 度コーディングシステムを選択すると、そのコーディングシステムは、`buffer-file-coding-system`に記録されます。これにより `save-buffer`や `write-region` などの、バッファーからファイルに書き込む際のデフォルトに、それを使用するようになります。`set-buffer-file-coding-system`を使って、バッファーのコーディングシステムとは異なるコーディングシステムで、ファイルに書き込むよう指定できます (Section 19.9 [Text Coding], page 227 を参照してください)。

Emacs がサポートする任意の文字を、任意の Emacs バッファーに挿入できますが、ほとんどのコーディングシステムは、それらの文字のサブセットしか処理することができません。したがって挿入した文字は、そのバッファーを保存するのに使われるコーディングシステムではエンコードできないかもしれません。たとえば、`iso-8859-2`でエンコードされた Polish のファイルを visit して、それに Russian の単語を追加することは可能です。このバッファーを保存するとき、Emacs は`buffer-file-coding-system`の現在の値を使用できません。なぜなら追加された文字が、そのコーディングシステムではエンコードできないからです。

これが発生した場合、Emacs は (`M-x prefer-coding-system`または `M-x set-language-environment`によりセットされた) もっとも適したコーディングシステムを試します。そのコーディングシステムがバッファーのすべての文字をエンコードできたら、Emacs はそれを使って、その値を `buffer-file-coding-system`に格納します。そうでなければ Emacs はバッファー内容をエンコードするのに適したコーディングシステムのリストを表示して、それらのコーディングシステムを 1 つ選ぶよう求めます。

メールメッセージに適さない文字を入力した場合、Emacs の振る舞いは若干異なります。この場合、追加で MIME メッセージに推奨されたもっとも適したコーディングシステムかをチェックします。もしそうでなければ、この事実を知らせ、他のコーディングシステムの入力を求めます。これにより、メール受信者のメールソフトがデコードするのが困難なエンコードで、無意識にメッセージを送るようなことがなくなります (入力をもとめられたときに、適さないコーディングシステムを選ぶ、という選択肢もまだ残っています)。

メールメッセージを送信するとき、Emacs はメッセージテキストのエンコーディングに使うコーディングシステムを決定する、4 つの異なる方法をもっています。最初にバッファー自身の `buffer-file-coding-system`が非 `nil`なら、それを使います。次に `sendmail-coding-system`が非 `nil`なら、それを使います。3 番目は `default-sendmail-coding-system`の値を使います。上述した値のすべてが `nil`の場合、Emacs は、新たなファイルに使用されるデフォルトコーディングシステム (選択された言語環境により制御される、`buffer-file-coding-system`の値) を使用して送信メールをエンコードします。

19.9 ファイルのテキストにたいするコーディングシステムの指定

Emacs がファイル内容にたいして、自動的に正しいコーディングシステムを選択しない場合、コーディングシステムを指定するために、以下のコマンドを使用できます。

`C-x RET f` *coding* `RET`
: カレントバッファーのファイルを、コーディングシステム *coding*を使って保存または再 visit します (`set-buffer-file-coding-system`)。

C-x RET c *coding* RET
: 直後に続くコマンドのコーディングシステムに *coding*を指定します (universal-coding-system-argument).

C-x RET r *coding* RET
: コーディングシステム *coding*を使って、現在のファイルを再 visit します (revert-buffer-with-coding-system)。

M-x recode-region RET *right* RET *wrong* RET
: コーディングシステム *wrong*を使ってデコードされたリージョンを、かわりにコーディングシステム *right*を使ってデコードします。

コマンド C-x RET f (set-buffer-file-coding-system) は、カレントバッファーのファイルのコーディングシステムをセットします (たとえばファイルを保存またはリバートするときに使うコーディングシステム)。これはミニバッファーを使ってコーディングシステムを指定します。モードラインのコーディングシステムインディケーターを mouse-3でクリックしても、このコマンドを呼び出すことができます。

バッファーのすべての文字を処理できないコーディングシステムを指定した場合、Emacs は問題となる文字について警告します。そしてそのバッファーを保存するときのコーディングシステムの選択を求めます。

このコマンドを、カレントバッファーのエンコーディングの際の改行変換の指示に使うこともできます (Section 19.5 [Coding Systems], page 223 を参照してください)。たとえば C-x RET f dos RETは、カレントバッファーを、DOS スタイル (行末が改行復帰文字) で保存します。

ファイルにたいしてコーディングシステムを指定する他の方法は、ファイルを visit するときに指定する方法です。最初にコマンド C-x RET c (universal-coding-system-argument) を使います。このコマンドはミニバッファーを使ってコーディングシステムを読み取ります。ミニバッファーを抜けた後、その直後に続くコマンドに、指定したコーディングシステムが使用されます。

たとえば直後に続くコマンドが C-x C-fの場合、そのコーディングシステムを使ってファイルを読み込みます (そして後で保存するときのために、そのコーディングシステムを記録します)。直後に続くコマンドが C-x C-wの場合、そのコーディングシステムを使ってファイルを書き込みます。C-x RET fのかわりに、この方法で保存するときのコーディングシステムを指定した場合、バッファーにそのコーディングシステムが処理できない文字が含まれていても警告はされません。

C-x iや C-x C-v、同様に C-x C-fの別ウィンドウ版 C-x RET cなど、その他のファイルコマンドも指定されたコーディングシステムに影響されます。そして M-x shell (Section 31.5 [Shell], page 457 を参照してください) を含む、サブプロセスを開始するコマンドも影響を受けます。直後に続くコマンドがコーディングシステムを使用しない場合、C-x RET cは何の影響も与えません。

変換をせずにファイルを visit する簡単な方法は、M-x find-file-literallyコマンドです。Section 15.2 [Visiting], page 146 を参照してください。

変数 buffer-file-coding-systemのデフォルト値は、新しいファイルを作成するときに選択されるコーディングシステムを指定します。これは新しいファイルを作成するときや、バッファーを作成してそれをファイルに保存するときに適用されます。言語環境の選択は、この変数を言語環境にたいして適した、デフォルトのコーディングシステムにセットします。

間違ったコーディングシステムでファイルを visit したときは、C-x RET r (revert-buffer-with-coding-system) でこれを正すことができます。これは指定したコーディングシステムを使って、現在のファイルを再 visit します。

テキストの一部が、すでに間違ったコーディングシステムでバッファーに挿入されてしまった場合、M-x recode-regionを使ってデコードしなおすことができます。これは正しいコーディングシ

ステムと、実際に使われた間違ったコーディングシステムの入力を求め、変換を行います。最初にリージョンを間違ったコーディングシステムでエンコードして、その後で正しいコーディングシステムでデコードします。

19.10 プロセス間通信にたいするコーディングシステム

このセクションでは、他のプロセスと通信するときに使うコーディングシステムを指定する方法を説明します。

`C-x RET x` *coding* `RET`
: 選択したテキストを、他のグラフィカルなアプリケーションと送受信するために、コーディングシステム *coding*を使用します (`set-selection-coding-system`)。

`C-x RET X` *coding* `RET`
: 次回に選択するテキストを、他のグラフィカルなアプリケーションと送受信するために、コーディングシステム *coding*を使用します (`set-next-selection-coding-system`)。

`C-x RET p` *input-coding* `RET` *output-coding* `RET`
: カレントバッファーでのサブプロセスの入出力に、コーディングシステム *input-coding* と *output-coding*を使用します (`set-buffer-process-coding-system`)。

コマンド `C-x RET x` (`set-selection-coding-system`) は、選択したテキストを他のウィンドウアプリケーションに送信するとき、および他のアプリケーションで選択されたテキストを受信するときのコーディングシステムを指定します。このコマンドは、このコマンドを再度使って設定をオーバーライドするまで、以降のすべての選択に適用されます。コマンド `C-x RET X` (`set-next-selection-coding-system`) は、Emacs で次に選択されるテキスト、または次に読み取られるテキストのためのコーディングシステムを指定します。

変数 `x-select-request-type`は、X ウィンドウシステムからのリクエストにより、他のアプリケーションで選択されたテキストを受信する際のデータタイプを指定します。値が `nil`(デフォルト) の場合、Emacs は `UTF8_STRING`、`COMPOUND_TEXT`の順に試み、さらにさまざまな経験則を用いて、2 つの結果からより適したものを選択します。どちらも成功しなかったとき、Emacs は `STRING`にフォールバックします。`x-select-request-type`の値が、`COMPOUND_TEXT`、`UTF8_STRING`、`STRING`、`TEXT`のうちのどれかであった場合、Emacs はリクエストされたタイプだけを使用します。値がこれらのシンボルのリストだった場合、Emacs はリストのリクエストタイプを順に試行し、どれかが成功するか、すべてを試みるまで続けます。

コマンド `C-x RET p` (`set-buffer-process-coding-system`) は、サブプロセスの入出力のコーディングシステムを指定します。このコマンドはカレントバッファーに適用されます。サブプロセスは通常、それぞれ自身のバッファーをもっています。したがってサブプロセスに対応するバッファーでこのコマンドを実行することにより、特定のサブプロセスとの送受信に使用するコーディングシステムを指定できます。

サブプロセスを開始するコマンドの直前に `C-x RET c` (`universal-coding-system-argument`) を使うことにより、そのプロセスとの通信で使用するコーディングシステムを指定することもできます。Section 19.9 [Text Coding], page 227 を参照してください。

デフォルトでは、プロセス通信の入出力は現在の言語環境に依存します。

変数 `locale-coding-system`は、システムのエラーメッセージや、`format-time-string`のフォーマットやタイムスタンプなどの、システム文字列のエンコードおよびデコードで使用するコーディングシステムを指定します。このコーディングシステムは X ウィンドウシステムでの非 ASCII キー

ボードによる入力のデコードに使用されるかもしれず、バッチモードにおいて標準出力とエラーストリームに送るテキストのエンコードにも使用されるでしょう。通常は環境変数 LC_ALL、LC_CTYPE、LANGのうちの 1 つで指定される、背景にあるシステムのテキスト表現 (text representation) と互換性のあるコーディングシステムを選択するべきです (上記の順番で最初の環境変数の値が空でない場合、それはテキスト表現を決定します)。

19.11 ファイル名にたいするコーディングシステム

C-x RET F *coding* RET
: ファイル名のエンコードおよびデコードに、コーディングシステム *coding*を使用します (set-file-name-coding-system)。

コマンド C-x RET F (set-file-name-coding-system) は、ファイルの名前に使用するコーディングシステムを指定します。ファイルの内容の読み込みと書き込みには影響がありません。

実際にこのコマンドが行うのは、変数 file-name-coding-systemに値をセットすることだけです。変数にコーディングシステムの名前 (Lisp シンボルか文字列) をセットすると、Emacs はすべてのファイル操作において、ファイル名のエンコードにそのコーディングシステムを使用します。これによりファイル名に非 ASCII文字— または少なくとも指定されたコーディングシステムではエンコードできる非 ASCII文字 — を使うことが可能になります。

file-name-coding-systemが nilの場合、Emacs は言語環境により選択され、変数 default-file-name-coding-systemに格納される、デフォルトのコーディングシステム (通常は UTF-8) を使用します。

Emacs が、MS-Windows の NT ファミリーの子孫 (Windows 2000、XP、および以降すべてのバージョン) にあたるバージョンで実行されている場合、file-name-coding-systemの値は大部分が無視されます。これは Emacs がデフォルトで Unicode ファイル名を直接渡せる API を使用するからです。一方、Windows 9X では、ファイル名は変数 file-name-coding-systemを使ってエンコードされており、この変数にはカレントのシステムロケールにたいして適切なコードページ (Section 19.5 [Coding Systems], page 223 を参照してください) がセットされている必要があります。変数 w32-unicode-filenamesの値は、Emacs がファイル名を引数とする OS 関数を呼び出す Unicode API を使うかどうかを制御します。この変数はスタートアップコードにより、Windows 9X では nil、新しいバージョンの MS-Windows では tにセットされます。

警告: Emacs セッションの途中で file-name-coding-system(または言語環境) を変更した場合、すでに visit しているファイルの名前が、古いコーディングシステムを使えばエンコードできるが、新しいコーディングシステムではエンコードされない (または違ってエンコードされる) という問題が発生します。このようなバッファーを visit したファイル名で保存を試みると、間違ったファイル名で保存するか、エラーが発生します。このような問題が発生したときは C-x C-wを使って、そのバッファーにたいして新しいファイル名を指定してください。

ファイル名をエンコードするとき間違いが発生した場合、コマンド M-x recode-file-nameを使って、ファイル名のコーディングシステムを変更します。このコマンドは古いコーディングシステムでの既存のファイル名と、変換したいコーディングシステムの入力を求めます。

19.12 X キーボード入力にたいするコーディングシステム

X ウィンドウシステムのインプットメソッドは自身のコーディングシステムを指定します。キーボード入力のデコードには、このコーディングシステムが使用されなければなりません。デフォルトではインプットメソッドそれぞれに使用するコーディングシステムは、インプットメソッドサーバーへの接続を確立する際に Emacs によって自動的に決定されて、キーボード入力のデコードにはその特定のコー

ディングシステムが使用されます。ただしコーディングシステムの決定が失敗する可能性もあり、その場合にはかわりに locale のコーディングシステムが使用されます (Section 19.10 [Communication Coding], page 229 を参照)。

インプットメソッドがテキストのエンコードに使用するコーディングシステムを正しく通知しない場合には、インプットメソッドからのテキストをデコードするために Emacs が使用するコーディングシステムを手作業で指定しなければなりません。変数 `x-input-coding-system`の値をシンボルにセットすると、それがインプットメソッドからのキーボード入力のデコードに使用するコーディングシステムとして無条件に使用されます。

19.13 端末入出力にたいするコーディングシステム

`C-x RET t` *coding* `RET`
: 端末の出力に、コーディングシステム *coding*を使用します (`set-terminal-coding-system`)。

`C-x RET k` *coding* `RET`
: キーボード入力に、コーディングシステム *coding*を使用します (`set-keyboard-coding-system`)。

コマンド `C-x RET t` (`set-terminal-coding-system`) は、端末出力のためのコーディングシステムを指定します。端末出力の文字コードを指定した場合、端末へのすべての文字出力は、指定したコーディングシステムに変換されます。

この機能は、特定の言語または文字セットをサポートするようビルドされた、特定の文字端末で有用です — たとえば European 端末は、ISO Latin 文字セットの 1 つをサポートします。マルチバイトテキストを使う場合は、端末のコーディングシステムを指定する必要があります。これにより、Emacs は端末が実際にどの文字を処理できるのか知ることができます。

デフォルトでは、Emacs が端末タイプまたは locale 指定により、正しいコーディングシステムを推論できない場合、端末への出力は変換されません。

コマンド `C-x RET k` (`set-keyboard-coding-system`)、または変数 `keyboard-coding-system`は、キーボード入力のためのコーディングシステムを指定します。キーボード入力の文字コード変換は、非 ASCIIのグラフィック文字を送信するキーをもつ端末で有用です— たとえば、いくつかの端末は ISO Latin-1、またはそれのサブセットのためにデザインされています。

デフォルトでは、キーボード入力はシステムの locale 設定にもとづいて変換されます。端末が locale により暗に指定されるエンコードを実際にはサポートしない場合 (たとえば、`M-i`をタイプしたときに非 ASCII文字が挿入されるのに気づいたとき)、エンコーディングをオフにするために `keyboard-coding-system`を `nil`にセットする必要があるでしょう。これは、

```
(set-keyboard-coding-system nil)
```

を init ファイルに記述することにより、行うことができます。

MS-Windows では古い Windows 9X システムを除き `keyboard-coding-system`をセットしても効果はないので、エンコードは MS-Windows コンソールのカレントコードページにマッチしなければなりませんが、これは `w32-set-console-codepage`の呼び出しによって変更できます。

キーボード入力にたいするコーディングシステムを使用した変換と、インプットメソッドの使用は似通った点があります。これらは両方ともキーボード入力シーケンスを 1 つの文字に変換します。しかし、インプットメソッドは人間により対話的に使用されることが便利なようにデザインされており、通常は ASCIIのプリント文字のシーケンスが、変換されたシーケンスになります。通常、コーディングシステムは非グラフィック文字のシーケンスを変換します。

19.14 フォントセット

フォントは通常、1 つのアルファベットまたはスクリプトの形状を定義します。したがって Emacs がサポートするスクリプトの全範囲を表示するには、多くのフォントのコレクションが要求されます。Emacs ではこのようなコレクションのことをフォントセット *(fontset)* と呼びます。フォントセットはフォント仕様のリストとして定義され、それぞれが文字コードのある範囲を処理し、指定されたフォントでカバーしない文字にたいしては他のフォントセットにフォールバックします。

それぞれのフォントセットは、フォントと同様に名前をもちます。しかしフォントはシステムに格納されていて、利用可能なフォント名はシステムで定義されていますが、フォントセットは Emacs 自身で定義されます。1 度フォントセットを定義したら、1 つのフォントを使える場所ならどこでも、フォントセットを名前で指定して使用することができます。もちろん Emacs のフォントセットに使用できるのは、システムがサポートするフォントだけです。もしある文字がスクリーン上で空のボックスや 16 進コードで表示される場合、それは使用しているフォントセットがその文字にたいするフォントをもっていないことを意味します。このような場合や、文字は表示されるが、それが意図したものとは異なる場合、多分追加のフォントをインストールするか、システムにすでにインストールされた特定のフォントを使用するようにフォントセットを修正する必要があるでしょう (以下参照)。オペレーティングシステムにはインストールできるオプションのフォントがあるはずです。またはサポートされたスクリプトのほとんどのフォントを含む GNU Intlfonts パッケージをインストールすることもできます。[2]

Emacs は 3 つのフォントセットを自動的に作成します。それはスタンダードフォントセット *(standard fontset)*、スタートアップフォントセット *(startup fontset)*、デフォルトフォントセット *(default fontset)* の 3 つです。デフォルトフォントセットは、さまざまな非 ASCII文字のフォントをもち、他の 2 つのフォントセットのデフォルトのフォールバック先です (デフォルトフォントをセットしたときは、デフォルトフォントセットではなくデフォルトフォント)。しかしこれはフォントのファミリー名を指定しないので、これを直接使うと、結果は少しランダムに思えるかもしれません。Emacs を '`-fn`'オプションで実行することにより、特定のフォントセットを使用するように指示できます。たとえば、

```
emacs -fn fontset-standard
```

'`Font`'でフォントセットを指定することもできます (Appendix D [X Resources], page 591 を参照してください)。

使用するフォントセットが何も指定されていない場合、Emacs は ASCIIフォントを使用し、そのフォントがカバーしない文字にたいするフォールバックに '`fontset-default`'を使用します。名前とは裏腹にスタンダードフォントセットは、明示的に要求されたときだけ使用されます。

特定のフォントセットの情報を表示するためには `M-x describe-fontset` コマンドを使用します。このコマンドはフォントセットの名前 (デフォルトはカレントフレームで使用されているフォントセット) を尋ねて文字のすべての部分範囲 (subrange) と、フォントセット内でそれらに割り当てられたフォントを表示します。特定のフォントセットなしで開始されたセッション (通常発生し得る) で Emacs が使用するフォントを確認するには、プロンプトで `fontset-default RET`をタイプするか、あるいはカレントフレームで使用中のフォントを表示するために単に `RET`をタイプしてください。

フォントセットはすべての文字コードにたいしてフォントを指定する必要はありません。フォントセットが特定の文字にたいしてフォントを指定していない、または指定したフォントがシステムに

[2] Emacs を X で実行している場合には、以下のようにして新しくインストールしたフォントの場所を X server に指示する必要があるでしょう:

```
xset fp+ /usr/local/share/emacs/fonts
xset fp rehash
```

存在しなければ、フォントセットは文字を正しく表示できません。この場合には、その文字にたいして 16 進コード、細いスペース、または空のボックスがかわりに表示されます (詳細は Section 11.20 [glyphless characters], page 98 を参照)。フォントセットが何らかの文字範囲を指定するかもしれないものの、その視覚的な外観が好みではないかもしれません。これが発生した場合にはフォントセットを修正したくなるかもしれません。これを行う方法については Section 19.16 [Modifying Fontsets], page 234 を参照してください。

19.15 フォントセットの定義

X で Emacs を実行している場合、Emacs は `standard-fontset-spec`の値により、スタンダードフォントセットを作成します。このフォントセットの名前は、

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

または単に短く '`fontset-standard`'です。

GNUstep、および macOS ではスタンダードフォントセットは、`ns-standard-fontset-spec` の値を使って作成され、MS Windows では `w32-standard-fontset-spec`の値を使って作成されます。

スタンダードフォントセットのボールド、イタリック、ボールドイタリックなどの変種も自動的に作成されます。これらの変種の名前には '`medium`'のかわりに '`bold`'、または '`r`'のかわりに '`i`'、またはその両方が使われます。

Emacs は '`Font`'リソース、または '`-fn`'引数で指定した任意のデフォルト ASCIIフォント、または Emacs が起動時に見つけたデフォルトフォントにもとづいて、フォントセットを自動的に作成します。これがスタートアップフォントセット *(startup fontset)* で、名前は `fontset-startup`です。Emacs は *charset_registry* フィールドを '`fontset`'、*charset_encoding* フィールドを '`startup`'で置き換えてフォントセットを生成して、その置き換えた文字列をフォントセットの指定に用います。

たとえば以下の形式で Emacs を起動した場合、

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

Emacs は以下のフォントセットを生成して、それを X ウィンドウの初期フレームに使用します:

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

スタートアップフォントセットは、そのフォントでサポートされているすべての文字にたいして指定したフォントか、異なる registry または encoding のフォントを使用し、それ以外の文字は '`fontset-default`'にフォールバックして表示するでしょう。

X リソースの '`Emacs.Font`'では、フォントセット名を実際のフォント名のように指定できます。しかし '`Emacs*Font`'のようなワイルドカードを使ったリソースにフォントセット名を指定しないように注意してください— ワイルドカードを使った指定は、メニューのようなフォントセットを処理できないものも含めて、 他のさまざまな目的にも適用されます。Appendix D [X Resources], page 591 を参照してください。

'`Fontset-`*n*'という名前の X リソースを使って、追加のフォントセットを指定できます。ここで *n*は 0 から始まる整数です。リソースの値はつぎのような形式です:

fontpattern, [*charset*:*font*]...

ここで *fontpattern*は、最後の 2 つのフィールドを除いて、標準の X フォント名の形式です (前の fontset-startup の例を参照)。最後の 2 つのフィールドは、'`fontset-`*alias*'の形式をもつべきです。

すべてのフォントセットには 2 つの名前、長い名前と短い名前があります。長い名前は *fontpattern* です。短い名前は '`fontset-`*alias*'で、これは長い名前の最後の 2 つのフィールドです (たとえば、

スタートアップ時に自動的に作成されるフォントセットは‘`fontset-startup`’)。どちらの名前でもフォントセットを参照できます。

‘*`charset`*:*`font`*’という構成は、ある文字セットにたいして、(このフォントセットでは) どのフォントを使用するかを指定します。ここで *charset*は、文字セットの名前で、*font*はその文字セットに使用するフォントです。1 つのフォントセットの定義の中で、この構成を何度でも使用できます。

他の文字セットにたいしては、Emacs は *fontpattern*にもとづいて選択します。これは文字セットを記述する値で ‘`fontset-`*`alias`*’を置き換えます。ASCII文字フォントにたいしては、‘`fontset-`*`alias`*’を ‘`ISO8859-1`’で置き換えます。

これに加えて、複数の連続するフィールドがワイルドカードの場合、Emacs はそれらを 1 つのワイルドカードにまとめます。これは、オートスケールされたフォントの使用を避けるためです。大きいフォントをスケーリングしたフォントは編集に適しておらず、小さいフォントをスケーリングしたフォントも同様です。なぜなら Emacs がそうするように、もともと小さなフォントを使うほうがよいからです。

したがって、*fontpattern*が以下の場合、

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

ASCII文字にたいするフォント指定は、以下のようになるでしょう:

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

そして Chinese GB2312 文字にたいするフォント指定は、以下のようになるでしょう:

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

上記のフォント指定に一致する Chinese フォントがないかもしれません。ほとんどの X ディストリビューションには、*family*フィールドが ‘`song ti`’か ‘`fangsong ti`’の Chinese フォントだけが含まれています。そのような場合、‘`Fontset-`*`n`*’をつぎのように指定します:

```
Emacs.Fontset-0: -*-fixed-medium-r-normal-*-24-*-*-*-*-*-fontset-24,\
        chinese-gb2312:-*-*-medium-r-normal-*-24-*-gb2312*-*
```

そうすると Chinese GB2312 の文字を除くフォント指定では、*family*フィールドが ‘`fixed`’となり、Chinese GB2312 の文字に対するフォント指定では、*family*フィールドが ‘`*`’となります。

フォントセットのリソース値を処理してフォントセットを作る関数は、`create-fontset-from-fontset-spec`と呼ばれます。フォントセットを作るために、この関数を明示的に呼ぶこともできます。

フォントの命名についての詳細は、Section 18.8 [Fonts], page 201 を参照してください。

19.16 フォントセットの修正

常にフォントセットをスクラッチから作成する必要はありません。軽微な変更だけが必要なときは、通常は既存のフォントセット ‘`fontset-default`’を修正するのが簡単な方法でしょう。‘`fontset-default`’の修正は、それをフォールバックに使用する他のフォントセットにも影響するので、特定のスクリプトのために Emacs が選択するフォントに関する問題を解決する効果的な方法になり得ます。

フォントセットは関数`set-fontset-font`を使って、文字、文字セット、スクリプトフォントを修正する文字範囲、使用されるフォントの指定を修正することができます。以下は例です:

```
;; Prefer a big5 font for han characters.
(set-fontset-font "fontset-default"
                  'han (font-spec :registry "big5")
                  nil 'prepend)
```

```
;; Use MyPrivateFont for the Unicode private use area.
(set-fontset-font "fontset-default"  '(#xe000 . #xf8ff)
                  "MyPrivateFont")

;; Use Liberation Mono for latin-3 charset.
(set-fontset-font "fontset-default" 'iso-8859-3
                  "Liberation Mono")

;; 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)
```

`set-fontset-font`関数のより詳細な使用方法については Section “Fontsets” in *GNU Emacs Lisp Reference Manual* を参照してください。

文字のコードポイントや文字が属するスクリプトが不明なら Emacs に尋ねることができます。文字にポイントを移動して `C-u C-x =` (`what-cursor-position`) とタイプすれば、Emacs はポップアップする`*Help*`により多くの情報とともにこの情報が表示されます。Section 4.9 [Position Info], page 24 を参照してください。たとえば ‘`kana`’スクリプトに属する日本語文字が中国語文字とも混合された日本語テキストでは、以下のように日本語テキストに ‘`Kochi Gothic`’フォントを使用するように Emacs をセットアップして、‘`han`’スクリプトを使用します:

```
(set-fontset-font "fontset-default" 'han "Kochi Gothic")
```

(Emacs の ‘`han`’スクリプトは中国語だけではなく中国語、韓国語、日本語、いわゆる CJKのすべての文字をサポートするようにセットアップされています。)

既知のスクリプトのリストは変数`script-representative-chars`を確認してください。

上記のようなフォントセットのセッティングはデフォルトフォントがサポートしない文字のみに影響するので、‘`Kochi Gothic`’がラテン文字をカバーしていても、Emacs が使用するデフォルトフォントは通常は基本的なラテン文字をカバーするので使用されることはありません。

システムにインストールされているフォントが壊れていたり、使用されている文字にたいして好ましくない結果を生成するフォントがあるかもしれません。そのような場合、文字を表示するのに必要となる適切なフォントの検索に、それらのフォントを完全に無視するように、Emacs に指示したいと思うかもしれません。`face-ignored-fonts`の値 (リスト変数です) に、不適切なフォントを追加することによりこれを行なうことができます。以下は、そのような設定を`~/.emacs`に記述する例です:

```
(add-to-list 'face-ignored-fonts "Some Bad Font")
```

19.17 表示できない文字

あなたの端末では表示できない非 ASCII文字が、いくつか存在するかもしれません。ほとんどのテキスト端末は、1 つの文字セットだけをサポートします (Emacs に何を使うか指示するには、変数`default-terminal-coding-system`を使用します。Section 19.13 [Terminal Coding], page 231 を参照してください)。そのコーディングシステムではエンコードできない文字は、デフォルトでは ‘`?`’ と表示されます。

グラフィカルなディスプレイでは、より広範囲の文字を表示できますが、それらすべてのフォントがインストールされていないかもしれません。フォントがない文字は、中空のボックスで表示されます。

Latin-1 文字を使用するとき、端末が Latin-1 を表示できない場合、かわりにニーモニック ASCII シーケンスを表示できます。たとえば o-umlaut のかわりに '"o'が表示されます。これを行うには `iso-ascii`をロードします。

端末が Latin-1 を表示できる場合、Latin-1 と等しい文字と ASCIIニーモニックを混交して、他の European 文字セットを表示できます。これは変数 `latin1-display`をカスタマイズすることにより有効になります。ニーモニック ASCIIシーケンスは、ほとんどがインプットメソッドのプレフィクスに対応します。

19.18 Unibyte 編集モード

ISO 8859 Latin-*n*文字セットは、さまざまな European 言語で必要とされるアクセント文字と区切り文字を扱うために、8 進の 0240 から 0377(10 進の 160 から 250) の範囲の文字コードを定義しています。Emacs はこの範囲のバイトを、たとえ unibyte バッファー (たとえばマルチバイト文字を無効にしている場合) でも、それらを文字としてではなく、raw バイトとみなします。しかし、それでも Emacs はこれらの文字コードを、あたかも *1* つも 1 バイト文字セットに属するかのように 1 度に扱うことができます。これらのコードのどれを使うかを指定するには、`M-x set-language-environment` を呼び出して、'`Latin-n`'のような適切な言語環境を指定します。Section "Disabling Multibyte Characters" in *GNU Emacs Lisp Reference Manual* を参照してください。

端末や使っているフォントがこれらの文字をサポートしている場合、Emacs は 160 から 255 の文字を読み取り可能な文字として表示できます。これは自動的に行われます。グラフィカルなディスプレイでは、Emacs はフォントセットを通じて 1 バイト文字として表示できます。これは現在の言語環境で、それらに対応するマルチバイト文字を表示することにより行われます。これを行うには、変数 `unibyte-display-via-language-environment`に非 `nil`値を設定します。このセッティングは、これらのバイトを表示する方法だけに影響しますが、Emacs がそれらを文字としてではなく raw バイトとして扱うという基礎事実は変わらないことに注意して下さい。

端末で Latin-1 文字セットを表示できない場合、Emacs はこれらの文字をその文字が少なくとも何であるかを明確に理解できるような、ASCIIシーケンスとして表示できます。これを行うには、ライブラリー `iso-ascii`をロードします。他の Latin-*n*文字セットに対しても似たようなライブラリを実装できますが、これはまだ行われていません。

通常、非 ISO 8859 文字セット (10 進文字の 128 から 159 のコードも含む) は、8 進でエスケープ表示されます。ライブラリー `disp-table`の関数 `standard-display-8bit`を使うことにより、非標準の拡張バージョンの ISO 8859 文字セットに変更できます。

1 バイトの非 ASCII文字を入力する 2 つの方法があります:

- 選択した言語環境のインプットメソッドを使用することができます。Section 19.3 [Input Methods], page 220 を参照してください。unibyte バッファーでインプットメソッドを使用した場合、入力した非 ASCII文字は、ユニバイトに変換されます。
- キーボードが、非 ASCII文字を表現する (10 進の)128 以上の文字コードを生成できるならば、それらの文字コードを直接タイプすることができます。

  グラフィカルなディスプレイでは、これらのキーを使うのに特別なことをする必要はありません。それらは単純に機能するでしょう。テキスト端末では、コマンド `M-x set-keyboard-coding-system`を使うか、変数 `keyboard-coding-system`をカスタマイズして、キーボードが使用するコーディングシステムを指定します (Section 19.13 [Terminal Coding], page 231 を参照してください)。この機能を有効にすることにより、おそらく Meta 文字を入力するために `ESC`を使う必要が生じるでしょう。しかし、コンソール端末、または `xterm`のような端末エミュレータでは、Meta を `ESC`にアレンジすることが可能です。また 8 ビット文字を直接キーボードから入力し

たり、Composeキーや AltGrキーを使うこともできます。Section 2.1 [User Input], page 12 を参照してください。

モダンなシステムの多くが独自にキーボードのキーを割り当てられていない文字をもつ多くの言語にたいして、ネイティブインプットメソッド: *native input methods* を提供しています。これらのネイティブインプットメソッドのサポート付きで Emacs がビルドされていれば、そのようなインプットメソッドをアクティブにして、それらがサポートしている文字をタイプできます。これらのインプットメソッドをどのようにアクティブにして使用するかはシステムとそのインプットメソッドに依存する問題であり、ここでは説明しません。あなたのシステムのドキュメントを参照してください。ここではネイティブインプットメソッドの使用を制御するための、Emacs の機能について説明します。

GTK ツールキットとともにビルドされた Emacs ではかな、Emacs が GTK インプットメソッドが生成した文字を受け取るかどうかを変数 x-gtk-use-native-inputが制御します。Emacs はこの変数が nil(デフォルト) なら X インプットメソッド、それ以外なら GTK インプットメソッドを使用します。X リソース useXIMは XIMを使うかどうか、X リソース inputStyleはネイティブインプットメソッドが生成したテキストプレビューの表示を制御します。Section D.2 [Table of Resources], page 592 を参照してください。

MS-Windows の Emacs では IMM(Input Method Manager)が提供するネイティブインプットメソッドがサポートされていますが、必要であればオフに切り替えることができます。Section H.6 [Windows Keyboard], page 612 を参照してください。

- 非 ASCIIの Latin-1、および他のプリント文字の合成文字 (compose-character) プレフィックスとして C-x 8を使用できます。C-x 8は、(ミニバッファーや他のバッファーでの) 挿入、検索、キーシーケンスが許される他のコンテキストなどで使用できます。

  ライブラリー iso-translをロードすることにより C-x 8は機能します。1 度ライブラリーをロードすると、Alt修飾キーがある場合は、C-x 8と同じ目的で使用できます。後続の文字を修飾するには、アクセント文字と一緒に Altを使用します。さらに Latin-1 の専用アクセント文字キー (dead accent characters) があると、1 度 iso-translをロードした後は、それらのキーも後続の文字を合成するように定義されます。

  C-x 8 C-hを使用すると、利用可能なすべての C-x 8翻訳をリストします。

19.19 文字セット

Emacs では “文字セット (character set)” を縮めて、*charset* と呼びます。Emacs は、ほとんどの有名な charsets(ascii、iso-8859-1、cp1250、big5、unicodeなど) に加えて、Emacs 自身の charsets(emacs、unicode-bmp、eight-bitなど) をサポートします。すべてのサポートされた文字は、1 つ以上の charsets に属します。

Emacs は通常、charsets にたいして正しいことを行う (does the right thing) ので、あなたはそれらを心配する必要はありません。しかし、charsets の背景の詳細を知ることが助けになる場合もあります。

1 つの例はフォント選択です (Section 18.8 [Fonts], page 201 を参照してください)。それぞれの言語環境 (Section 19.2 [Language Environments], page 218 を参照してください) は、さまざまな文字にたいする優先リスト (priority list) を定義します。フォントを検索するとき、Emacs は最初に一番優先度の高い charsets を表示できるものを探すことを試みます。たとえば Japanese 言語環境では、charsets japanese-jisx0208は一番高い優先度をもっているので、Emacs は registryプロパティが ‘JISX0208.1983-0’のフォントの使用を試みます。

charsets に関する情報を得るのに使うことができるコマンドが 2 つあります。コマンド M-x list-charset-charsは charset 名の入力を求め、その文字セットのすべての文字を表示します。

コマンド `M-x describe-character-set`は charset 名の入力を求め、Emacs での内部表現も含めたその charset に関する情報を表示します。

`M-x list-character-sets`は、すべてのサポートされた charsets を表示します。このリストは charsets の名前と、各 charset を識別する追加の情報を与えます。詳細については、Information Processing Society of Japan/Information Technology Standards Commission of Japan (IPSJ/ITSCJ) (`https://www.itscj.ipsj.or.jp/itscj_english/`) により保守されている、ISO International Register of Coded Character Sets to be Used with Escape Sequences (ISO-IR) (`https://www.itscj.ipsj.or.jp/itscj_english/iso-ir/ISO-IR.pdf`) を参照してください。このリストでは、charsets は 2 つのカテゴリーに分かれています。通常の *charsets(normal charsets)* が最初にリストされ、その後に追加の *charsets(supplementary charsets)* が続きます。追加の charset は他の charset を定義するのに (サブセットの親として) 使用されるか、古いバージョンの Emacs との互換性のために提供されます。

バッファーの文字がどの charset に属するか探すには、ポイントをその文字の前において、`C-u C-x =`をタイプします (Section 19.1 [International Chars], page 216 を参照してください)。

19.20 双方向の編集

Emacs は Arabic や Farsi、Hebrew のような、テキストを水平方向の右から左に記述するスクリプトで書かれたテキストの編集をサポートします。しかし数字やそれらのスクリプトに埋め込まれた Latin テキストは、左から右に表示されます。Latin 文書の中に少量の Arabic や Hebrew のテキスト部分が含まれている場合も、稀ではありません (例: プログラムソース内のコメントや文字列)。これらの理由により、これらのスクリプトを使うテキストは、実際には双方向 *(bidirectional)*、つまりそれらは left-to-right(左から右) の文字と right-to-left(右から左) 文字の混交されたものになります。

このセクションでは、双方向テキストを編集するために Emacs が提供する機能とオプションを説明します。

Emacs は right-to-left および双方向テキストを、いわゆる *logical* 順 (または *reading* 順) で格納します。バッファーまたは文字列の最初の文字の位置は、次に読む文字の前になります。双方向テキストを *visual* 順に再配置するには表示時間が発生します。結果として文字の位置は、それらが表示される位置にたいして単調に増加しなくなります。Emacs は表示のための双方向テキストの再配置を、Unicode Standard Annex #9 (`https://unicode.org/reports/tr9/`) で説明されている UBA(Unicode Bidirectional Algorithm) で実装しています。right-to-left パラグラフ中に長い英文テキストが出現するような、基本となるパラグラフと継続行が逆方向のときに表示する方法だけが、UBA と異なります。

バッファーローカルな変数 `bidi-display-reordering`は、表示用にバッファーのテキストを再配置するかどうかを制御します。この変数の値が非 `nil`の場合、Emacs は右から左の方向に表示される文字を再配置します。デフォルト値は `t`です。

双方向テキストの各パラグラフは、それ自身の *base direction(*基本方向*)* をもっており、それは right-to-left または left-to-right です。left-to-right のパラグラフはスクリーンの左端から開始し、右端に到達すると切り詰め、または継続されます。対照的に right-to-left のパラグラフのテキストは右端から開始し、左端で継続、または切り詰められて表示されます。デフォルトでは、パラグラフの境界は空行 (たとえば行全体が空白文字からなる行) です。これを変更するために、2 つの変数 `bidi-paragraph-start-re`と `bidi-paragraph-separate-re`をカスタマイズできます。これらの値には正規表現 (文字列) を指定します。たとえば単一の改行を新たなパラグラフの開始とする場合は、両方の変数に`\"^\"`をセットします。これら 2 つの変数はバッファーローカルです (Section 33.2.3 [Locals], page 511 を参照)。

Emacs は、パラグラフを開始するテキストにもとづいて、各パラグラフの基本方向を動的に決定します。しかし、バッファーのパラグラフにたいして特定の基本方向を強制する必要もあるでしょう。変数 `bidi-paragraph-direction`が非 `nil`の場合、これは基本方向の動的な決定を無効にして、バッファーのすべてのパラグラフの方向を、このバッファーローカルな値で指定された方向に強制します。値には `right-to-left`と `left-to-right`が指定できます。これ以外の値は `nil`と解釈されます。

かわりにパラグラフの先頭に特別な文字を挿入することにより、パラグラフの基本方向を制御できます。特別な文字 `RIGHT-TO-LEFT MARK`または RLM は、以降に続くパラグラフを right-to-left 方向に強制します。その効果は `LEFT-TO-RIGHT MARK`または LRM により left-to-right 方向に再強制されるまで続きます (`C-x 8 RET`を使ってこれらの文字を挿入できます)。GUI セッションでは LRM 文字および RLM 文字は、極端に細いスペースで表示されます。テキスト端末では、それらはスペースで表示されます。

文字は表示用に再配置されるので、logical 順で処理を行う Emacs コマンドやバッファーの拡大は、普通とは異なる効果を生みます。たとえばコマンド `C-f`および `C-b`はポイントを logical 順で移動するので、再配置された双方向テキストではポイントがジャンプすることがあります。同様に隣接する文字位置の範囲をカバーするハイライトされたリージョンは、リージョンが再配置されたテキストにかかる場合には不連続に見える場合があります。これは双方向テキストをサポートする他のプログラムの振る舞いとしては普通であり、似通っています。

`LEFT`や `C-RIGHT`のように、矢印キーにバインドされたカーソル移動コマンドは、カレントパラグラフの基本方向にしたがいます。left-to-right パラグラフでは、修飾キーの有無に関わらず、`RIGHT`にバインドされるコマンドは、バッファーテキスト内を前方 *(forward)* に移動しますが、right-to-left パラグラフではかわりに後方 *(backward)* に移動することになります。これは、right-to-left パラグラフのバッファー位置は、ディスプレイ上を左に移動することにより大部分は増加するという事実を反映しています。

パラグラフ外に移動した際、先行または後続するパラグラフの基本方向が、移動する前のパラグラフの方向と異なる場合は、矢印キーのもつ意味は変化するでしょう。これが発生したときは、新たな基本方向に合わせて矢印キーを押下する必要があります。

デフォルトでは、`LEFT`および `RIGHT`は logical 方向に移動しますが、`visual-order-cursor-movement`が非 `nil`の場合、これらのコマンドはそれぞれスクリーン位置を左または右に、必要ならスクリーン行の次行または前行へと移動します。これは周辺の双方向コンテキストに依存して、多くのバッファー位置が移動される可能性を秘めることに注意してください。

双方向テキストは表示にたいするテキストの再並べ替えのために、特殊なフォーマット文字を使うことがあります。上述した LRM および RLM という 2 つの文字もそのような文字の一部ですが、他にもそのような文字は存在します。このような文字はデフォルトでは GUI フレームなら細いスペースのグリフ (thin space glyph)、テキストモードのフレームなら単純にスペースとして表示されます。このような特殊な制御文字がもたらす表示効果に驚かされないように識別可能にしたければ、`glyphless-display-mode`をオンに切り替えることができます (Section 11.20 [Text Display], page 98 を参照)。これはこれらのフォーマット用文字を、内部に制御文字の頭文字をもつ小さなボックスとして表示することで容易に理解できるようにするためのマイナーモードです。

20 メジャーモードとマイナーモード

Emacs には多くの編集用モード *(editing modes)* が含まれており、これは基本的な振る舞いを、編集に便利な方法に変更します。これらの編集用モードはメジャーモード *(major modes)* とマイナーモード *(minor modes)* に分けられます。

メジャーモードは、C ソースファイル (Chapter 23 [Programs], page 285 を参照してください) などの特定のファイルタイプや、shell バッファー (Section 31.5 [Shell], page 457 を参照してください) などの、特別なタイプの非ファイルバッファーにたいして作業するための特別な機能を提供します。メジャーモードは互いに排他であり、各バッファーは常に 1 つのメジャーモードをもちます。

マイナーモードはオンとオフを切り替えることができるオプションの機能で、ファイルやバッファーのタイプに特定する必要はありません。たとえば Auto Fill モードは、単語の間にタイプした `SPC`で行を区切るマイナーモードです (Section 22.6.1 [Auto Fill], page 256 を参照してください)。マイナーモードは互いに独立していて、選択されたメジャーモードからも独立しています。

20.1 メジャーモード

すべてのバッファーはメジャーモードをもっており、そのバッファーがカレントである間の編集の動作を決定します。モードラインには通常カレントのメジャーモード名がカッコ内に表示されます (Section 1.3 [Mode Line], page 9 を参照してください)。

もっとも特殊化されていないメジャーモードは、*Fundamental(*基本*)* モードと呼ばれます。このモードには、モード独自の再定義や変数設定がないので、各 Emacs コマンドはもっとも一般的な振る舞いをし、各ユーザーオプションはデフォルトの状態になっています。

Lisp や英文テキストのように、Emacs が認識できる特定のタイプのテキスト編集には、Lisp モードや Text モードのような、より特殊化されたメジャーモードを通常は使用します。ほとんどのメジャーモードは 3 つのグループに分けられます。最初のグループはプレインまたはマークアップされた通常テキストのためのモードを含みます。これには Text モード、HTML モード、SGML モード、TEX モードや Outline モードなどが含まれます。2 番目のグループはプログラミング言語特有のモードです。これらは、Lisp モード (いくつかの変種を有する)、C モード、Fortran モードなどが含まれます。3 番目のグループはファイルに直接関連付けられていないメジャーモードが含まれます。これらは Emacs が特別の目的のために作るバッファーで使用されるものです。例としては、Dired が作成するバッファーのための Dired モード (Chapter 27 [Dired], page 380 を参照)、`C-x m`で作成されるバッファーのための Message モード (Chapter 29 [Sending Mail], page 420 を参照)、下位のシェルプロセスとの通信用のバッファーのための Shell モード (Section 31.5.2 [Interactive Shell], page 459 を参照) などが含まれます。

通常、メジャーモードは最初にファイルを visit したとき、またはバッファーを作成したときに、Emacs により自動的にセットされます。`M-x`コマンドを使うことにより、新しいメジャーモードを明示的に選択することができます。モードの名前に`-mode`を追加することにより、モードを選択するコマンド名を得ることができます (たとえば、Lisp モードを選択する場合は `M-x lisp-mode`)。すべてのバッファーは厳密に 1 つのメジャーモードをもつので、メジャーモードを “オフ” にする方法はなく、かわりに他のメジャーモードに切り替えなければなりません。

バッファーローカルな変数 `major-mode`の値は、メジャーモードコマンドと同じ名前のシンボル (たとえば `lisp-mode`) です。この変数は自動的にセットされます。あなた自身が変更するべきではありません。

`major-mode`のデフォルト値は、メジャーモードが指定されていないファイルを使うときや、`C-x b`で作成した新しいバッファーのメジャーモードを決定します。通常、デフォルト値は Fundamental

モードを指定する、シンボル `fundamental-mode`です。Customization インターフェースを通じて、このデフォルト値を変更できます (Section 33.1 [Easy Customization], page 499 を参照してください)。init ファイルに以下のような行を追加しても変更できます (Section 33.4 [Init File], page 528 を参照してください):

```
(setq-default major-mode 'text-mode)
```

`major-mode`のデフォルト値が `nil`の場合、メジャーモードは前のカレントバッファーから引き継がれます。

特殊化されたメジャーモードは、特定のキーにたいして、そのモードにより適した何かを行うよう、意味づけが変更される場合があります。たとえばプログラミングに関連するモードでは、`TAB`には、カレント行をその言語のルールにしたがってインデントする機能がバインドされます (Chapter 21 [Indentation], page 247 を参照してください)。一般的に変更されるキーは `TAB`、`DEL`、`C-j`です。多くのモードがモード自身の特別なコマンドを定義しており、それらは通常、プレフィクスキーが `C-c`であるようなキーシーケンスにバインドされます。メジャーモードはユーザーオプションと変数も変更できます。たとえばプログラミングに間するモードは通常、変数 `comment-start`にバッファーローカルな値をセットします。これはソースコードのコメントがどのように区切られるかを決定します (Section 23.5 [Comments], page 295 を参照してください)。

カレントメジャーモードのキーバインディング一覧も含めたドキュメントを閲覧するには、`C-h m` (`describe-mode`) とタイプします。Section 7.7 [Misc Help], page 49 を参照してください。

Fundamental モード以外のすべてのメジャーモードは、モードフック *(mode hook)* を定義します。これはバッファーでそのモードが有効になるたびに実行される、カスタマイズ可能な Lisp 関数のリストです。フックに間する詳細は、Section 33.2.2 [Hooks], page 509 を参照してください。各モードフックはメジャーモード名の後に名前がつけられます。たとえば Fortran モードのモードフックは、`fortran-mode-hook`です。さらに、すべてのテキストベースのメジャーモードは、`text-mode-hook`を実行し、多くの、プログラミング言語のモード (Emacs とともに配布されるものを含む) は、その言語モード自身のモードフックの前に、`prog-mode-hook`を実行します [1]。フック関数は変数 `major-mode`の値を調べて、どのモードに入ろうとしているか調べることができます。

モードフックは、一般的にマイナーモードを有効にするために使用されます (Section 20.2 [Minor Modes], page 241 を参照)。たとえば以下の行を init ファイルに記述すると、すべてのテキストベースのメジャーモードで Flyspell マイナーモード (Section 13.4 [Spelling], page 134 を参照) を、Emacs Lisp モードで ElDoc マイナーモード (Section 23.6.3 [Programming Language Doc], page 299 を参照) を有効にすることができます:

```
(add-hook 'text-mode-hook 'flyspell-mode)
(add-hook 'emacs-lisp-mode-hook 'eldoc-mode)
```

20.2 マイナーモード

マイナーモードは明確な方法で Emacs の動作を変更する、オプションの編集用モードです。メジャーモードとは異なり、いつでも任意の数のマイナーモードを有効にできます。いくつかのマイナーモードはバッファーローカル *(buffer-local)* で、特定のバッファーにたいしてオン (有効) にして、他のバッファーではオフ (無効) に切り替えることができます。それ以外のマイナーモードはグローバル *(global)* で、それが有効な間は Emacs セッションのすべてのバッファーで行う、すべての操作に影響します。ほとんどのマイナーモードはデフォルトで無効ですが、デフォルトで有効なものもいくつかあります。

[1] より具体的には、そのモードは `prog-mode`から “派生” したモードです (Section “Derived Modes” in *The Emacs Lisp Reference Manual* を参照)。

ほとんどのバッファーローカルなマイナーモードは、モードラインのメジャーモード標識のすぐ後ろに有効であることを示します。たとえばモードラインに‘Fill’と表示されているとき、それはAuto Fillモードが有効であることを意味します。Section 1.3 [Mode Line], page 9 を参照してください。

メジャーモードと同様に、各マイナーモードはモードコマンド *(mode command)* に関連付けられていて、それはモード名の後ろに‘-mode’を付けた名前です。たとえばAuto Fillモードのモードコマンドは`auto-fill-mode`です。しかしメジャーモードのコマンドは、単純にそのモードを有効にするだけですが、マイナーモードのモードコマンドは、モードを有効または無効にすることができます。

- `M-x`を通じて、またはバインドしたキー (Section 33.3 [Key Bindings], page 519 を参照してください) をタイプすることにより、モードコマンドをプレフィクスキーなしで直接呼び出すと、それはマイナーモードを切り替え *(toggles)* ます。つまり、マイナーモードがオフのときはオンに、オンのときはオフに切り替えます。
- プレフィクス引数を指定してモードコマンドを呼び出すと、引数が0または負のときは無条件にマイナーモードをオフにし、それ以外のときは無条件にオンに切り替えます。
- Lispからモードコマンドが呼び出された場合、引数が省略されているか`nil`のとき、マイナーモードは無条件にオンになります。これはメジャーモードのモードフックからマイナーモードをオンに切り替えるのを簡単にします (Section 20.1 [Major Modes], page 240 を参照してください)。非`nil`の引数は、上で説明したインタラクティブなプレフィクス引数と同様に処理されます。

ほとんどのマイナーモードは、モードコマンドと同じ名前のモード変数 *(mode variable)* をもっています。変数の値が非`nil`のときはモードが有効で、`nil`なら無効です。一般的に、Lispから直接モード変数を変更して、モードを有効または無効にするべきではありません。かわりにモードコマンドを使うべきです。しかしCustomizeインターフェース (Section 33.1 [Easy Customization], page 499 を参照してください) を通じてのノード変数のセットは、Customizeが自動的にモードコマンドを実行するので、常に正しくモードを有効または無効にします。

以下にいくつかのバッファーローカルなマイナーモードのリストを示します:

- Abbrevモードは、事前に定義された省略形 (abbreviation) の定義にもとづいて、テキストを自動的に展開します。Chapter 26 [Abbrevs], page 373 を参照してください。
- Auto Fillモードは、行が長くなりすぎるのを防ぐため、タイプされた文字にしたがって改行を挿入します。Section 22.6 [Filling], page 256 を参照してください。
- Auto Saveモードはバッファー内容を定期的に保存して、クラッシュした場合等に失われる作業量を減らします。Section 15.6 [Auto Save], page 159 を参照してください。
- Electric Quoteモードは、クォーテーションマークを自動的に変換します。たとえば、‘`like this'`とタイプするとこれは、`` `like this' ``のように再クォートされます。どうのような種類のテキストを処理するかを制御できます。また特定のバッファーにたいしてこれを完全に無効にできます。Section 22.5 [Quotation Marks], page 255 を参照してください。
- Enrichedモードは、書式つきのテキストの編集と保存を可能にします。Section 22.14 [Enriched Text], page 275 を参照してください。
- Flyspellモードは、自動的に間違ったスペルの単語をハイライトします。Section 13.4 [Spelling], page 134 を参照してください。
- Font-Lockモードは、プログラム内で見つかった特定のテキスト単位を自動的にハイライトします。このモードはデフォルトでグローバルに有効になっていますが、個別のバッファーで無効にすることができます。Section 11.8 [Faces], page 82 を参照してください。

- Display Line Numbers モードは `display-line-numbers`の便利なラッパーであり、`display-line-numbers-type`の値を使用してセッティングします。Section 11.24 [Display Custom], page 102 を参照してください。
- Outline minor モードは、Outline モードと呼ばれるメジャーモードと同様な機能を提供します。Section 22.9 [Outline Mode], page 261 を参照してください。
- Overwrite モードは、通常のプリント文字の挿入により、後の文字をずらすかわりに、既存のテキストを置き換えます。たとえば、ポイントが‘`FOOBAR`’の‘`B`’の前にある場合、`G`をタイプすると通常は‘`FOOGBAR`’となりますが、Overwrite モードでは‘`FOOGAR`’になります。Overwrite モードでは、コマンド `C-q`は次の文字が何であれ、たとえそれが数字であってもその文字を挿入します— これにより既存のテキストを置き換える代わりに文字を挿入する方法が与えられます。モードコマンド `overwrite-mode`は、`Insert`キーにバインドされています。
- Binary Overwrite モードは、バイナリーファイルを編集するための、Overwrite モードの変種です。このモードは改行とタブを他の文字と同じように扱うので、他の文字を上書きしたり、他の文字で上書きさせたりすることができます。Binary Overwrite モードでは `C-q`の後の数字は、通常どおり 8 進文字コードを指定します。
- Visual Line モードは、 単語単位の折り返し (word wrapping) を処理します。これにより長い行は単語境界で折り返されます。Section 11.23 [Visual Line Mode], page 101 を参照してください。

以下に便利なグローバルマイナーモードをいくつか示します:

- Column Number モードは、現在の列番号をモードラインに表示します。Section 1.3 [Mode Line], page 9 を参照してください。
- Delete Selection モードでは、リージョンがアクティブの場合、最初にリージョンのテキストを削除してからテキストを挿入します。Section 8.3 [Using Region], page 55 を参照してください。
- Icomplete モードは、ミニバッファーで補完がアクティブのとき、利用可能な候補を表示します。Section 16.7.2 [Icomplete], page 184 を参照してください。
- Line Number モードは、現在の行番号をモードラインに表示します。このモードはデフォルトで有効です。Section 1.3 [Mode Line], page 9 を参照してください。
- Menu Bar モードは、各フレームにメニューバーを表示します。このモードはデフォルトで有効です。Section 18.15 [Menu Bars], page 209 を参照してください。
- Scroll Bar モードは、各ウィンドウにスクロールバーを表示します。このモードはデフォルトで有効ですが、スクロールバーが表示されるのはグラフィカルな端末だけです。Section 18.12 [Scroll Bars], page 206 を参照してください。
- Tool Bar モードは、各フレームにツールバーを表示します。このモードはデフォルトで有効ですが、ツールバーが表示されるのはグラフィカルな端末だけです。Section 18.16 [Tool Bars], page 209 を参照してください。
- Tab Bar モードは各フレームにタブバーを表示します。Section 18.17 [Tab Bars], page 210 を参照してください。
- Tab Line モードは各ウィンドウにタブラインを表示します。Section 17.8 [Tab Line], page 193 を参照してください。
- Transient Mark モードはリージョンをハイライトして、マークがアクティブなときは Emacs の多くのコマンドがリージョンにたいして操作を行うようになります。このモードはデフォルトで有効です。Chapter 8 [Mark], page 52 を参照してください。

20.3 ファイルのモードを選択する

ファイルを visit したとき、Emacs は自動的にメジャーモードを選択します。これは通常、ファイル名にもとづいて選択されます—たとえば名前が '.c'で終わるファイルは通常、C モードで編集されます— が、ファイル内の特別なテキストにもとづいてメジャーモードが選択されるときもあります。この特別なテキストは、バッファーローカルなマイナーモードを有効にするためにも使用されます。

以下は、これの正確な手順です:

最初に Emacs は、ファイルにファイルローカル (file-local) なモード変数が含まれているかチェックします。Section 33.2.4 [File Variables], page 512 を参照してください。メジャーモードを指定するファイルローカル変数が存在する場合、Emacs は他の条件をすべて無視してそのメジャーモードを使用します。ファイルローカル変数を使用してメジャーモードを指定する方法はいくつかあります。一番単純なのは、空行でない最初の行に、そのモードの名前と、モード名の前後に '-*-'を記述する方法です。他のテキストがその行にあっても問題はありません。たとえば、

```
; -*-Lisp-*-
```

これは Emacs に Lisp モードを使用するよう指示します。Lisp がこの行をコメントとして扱うように、セミコロンがどのように使われているか注意してください。以下のように書くこともできます

```
; -*- mode: Lisp;-*-
```

ファイルローカル変数を使用して、バッファーローカルなマイナーモードを指定することもできます。これは `eval`指定を使用して行います。たとえば、空行でない最初の行に以下を記述すると、これはバッファーを Lisp モードにして、Auto-Fill モードを有効にします。

```
; -*- mode: Lisp; eval: (auto-fill-mode 1); -*-
```

しかし、ほとんどのマイナーモードはユーザー個人の好みが表れるものなので、この方法でマイナーモードを有効にするのは、通常は不適切です。特定のファイルタイプにたいしてマイナーモードを個人的に使用したい場合は、メジャーモードフックを通じてマイナーモードを有効にするのが、より良い方法です。

2 番目に Emacs はファイルの拡張子がディレクトリーローカル `auto-mode-alist`のいずれかにマッチするかどうかをチェックします。これらは`.dir-locals.el`の手法により発見されます (Section 33.2.5 [Directory Variables], page 516 を参照)。

3 番目に、メジャーモードを指定するファイル変数が存在しない場合、Emacs は最初の行が '`#!`' で開始されていないかチェックします。もし該当したら、それはそのファイルがファイルの最初の行に記述された名前のインタープリターを実行する (ファイルの残りはインタープリターへの入力として使用されます) ことにより機能する、実行可能なシェルコマンドであることを示します。したがって Emacs はインタープリター名を使って、モードの選択を試みます。たとえば、'`#!/usr/bin/perl`' で始まるファイルは、Perl モードで開かれます。変数 `interpreter-mode-alist`は、インタープリター名とメジャーモードの対応を指定します。

最初の行が '`#!`'で開始されている場合、最初の行で '`-*-`'の機能は通常は使用できません。なぜならインタープリターを実行するときにシステムが混乱するからです。そのため、Emacs はそのようなファイルにたいしては、1 行目と同じように 2 行目の '`-*-`'を探します。これはマジック文字列 '`'\"`' で開始される man page で、troff プリプロセッサーのリストを指定することと同じです。

4 番目に、Emacs は変数 `magic-mode-alist`にもとづいて、バッファーの先頭のテキストを調べて、メジャーモードの決定を試みます。デフォルトではこの変数は `nil`(空のリスト) なので、Emacs はこのステップをスキップしますが、init ファイルでこれをカスタマイズできます (Section 33.4 [Init File], page 528 を参照してください)。変数の値には、以下の形式のリスト要素を指定します

```
(regexp . mode-function)
```

ここで *regexp*は正規表現 (Section 12.6 [Regexps], page 115 を参照してください) で、*mode-function*はメジャーモード関数です。ファイルの先頭のテキストが *regexp*にマッチした場合、Emacs は *mode-function*で指定したメジャーモードを選択します。

`magic-mode-alist`の要素は、以下の形式でも指定できます

```
(match-function . mode-function)
```

ここで *match-function*は、バッファーの先頭で呼び出される Lisp 関数です。この関数が非 `nil`を返した場合、Emacs はメジャーモードを *mode-function*にセットします。

5 番目に —Emacs が適正なメジャーモードをまだ見つけていない場合 — 今度はファイル名を調べます。ファイル名とメジャーモードの対応付けは、変数 `auto-mode-alist`で制御されます。この変数の値は、各要素が以下の形式のリストです。

```
(regexp . mode-function)
```

または以下の形式です

```
(regexp mode-function flag)
```

たとえば、通常見られるようなリストの要素は、`("\\.c\\'" . c-mode)`のような形式で、これは名前が`.c`で終わるファイルにたいして、C モードを選択する役目を負っています (正規表現で '`.`'がもつ特別な意味を打ち消すために、文字列に '`\`'を含めなければなりません。これは Lisp 構文では '`\\`'と記述する必要があります)。

要素が (*regexp mode-function flag*) の形式をもち、*flag*が非 `nil`の場合には、(`nil`でなければ)*mode-function*を呼び出した後に、Emacs は *regexp*とマッチした接尾辞を捨ててほかのマッチを検索します。この "再帰的な拡張子除去 (recursive extension stripping)" は拡張子を複数もち、実際に正しいモードを指定する "内側" の拡張子を、"外側" の拡張子が隠しているようなファイルに使用されます。たとえばバックアップファイルや拡張子`.gpg`をもつ GPG 暗号化ファイルには、この機能が使用されます。

GNU/Linux および、ファイル名の大文字小文字を区別する他のシステムでは、Emacs は `auto-mode-alist`の検索で大文字小文字を区別します。この検索が失敗すると、2 回目は大文字小文字を区別せずに alist を再検索します。2 回目の検索を行わないようにするには、変数 `auto-mode-case-fold`を `nil`に変更します。Microsoft Windows のような、ファイル名の大文字小文字を区別しないシステムでは、Emacs は大文字小文字を区別しないで `auto-mode-alist`を検索します。

最後に、Emacs がまだ使用するメジャーモードを見つけられない場合、Emacs はバッファーの先頭のテキストと、変数 `magic-fallback-mode-alist`を比較します。この変数は上述した `magic-mode-alist`と同じように機能しますが、`auto-mode-alist`の後に調べられるという点が異なります。デフォルトでは `magic-fallback-mode-alist`にはイメージファイル、HTML/XML/SGML ファイル、PostScript ファイル、および Unix スタイルの Conf ファイルをチェックする形式が含まれています。

一旦メジャーモードが見つかれば、Emacs はそのモードが `major-mode-remap-alist`によってリマップ (remap: 再マップ) されているか最終チェックを行います (リマップされていればそのモードを使用する)。リマップは同一のファイルタイプに使用できる複数の異なるモードが存在する際に、好みのモードを指定できるようにするために使用されます。

バッファーのメジャーモードを変更したとき、`M-x normal-mode`とタイプすると、Emacs が自動的に選択するメジャーモードに戻ることができます。この関数は `find-file`がメジャーモードを選択するために呼び出す関数と同じです。そのバッファーがファイルを visit している場合には、このコマンドは (もしあれば) ファイルの '`-*-`'行とファイルローカル変数も処理します。Section 33.2.4 [File

Variables], page 512 を参照してください。そのバッファーがファイルを visit していなければ、このコマンドは (もしあれば)‘-*-’内のモード指定とファイルローカル変数リストだけを処理します。

コマンド C-x C-wおよび set-visited-file-nameは、新しいファイル名が何らかのモードに関係がある場合は、新しいメジャーコードに変更します (そのバッファーがファイルを visit していない場合、C-x C-sも同じことを行います)。しかしバッファーの内容がメジャーモードを指定していて、ある種の特別なメジャーモードはモードの変更を許しません。このモード変更機能をオフにするには、change-major-mode-with-file-nameを nilにセットして下さい。

21 インデント

インデント *(Indentation)* とは、空白文字 (スペースやタブ文字) を行のテキストの先頭に挿入したり調整することを指します。このチャプターでは、インデントコマンドと、Text モードやそれに関連するモード、同様にプログラミング言語のモードでの一般的なオプションを記します。プログラミング用のモードでのインデントに関する追加のドキュメントは、Section 23.3 [Program Indent], page 288 を参照してください。

インデントを行うもっとも簡単な方法は、TABキーです。ほとんどのメジャーモードでは、これによりコマンド `indent-for-tab-command`が実行されます (C およびそれに関連するモードでは、TABは同じように振る舞うコマンド `c-indent-line-or-region`を実行する。Section 23.3.4 [C Indent], page 290 を参照されたい)。

TAB　空白文字を挿入、またはモードに適した方法でカレント行をインデントします (`indent-for-tab-command`)。リージョンがアクティブのときは、リージョンのすべての行をインデントします。

TABの正確な動作は、メジャーモードに依存します。Text モードおよびそれに関連するモードでは、TABは通常、空白とタブ文字を組み合わせたものを挿入して、ポイントを次のタブストップに進めます。このために、先行する行の最初の空白以外の文字の位置は、追加のタブストップとして扱われるので、TABを使って先行する行にポイントを揃えることができます。リージョンがアクティブ (Section 8.3 [Using Region], page 55 を参照してください) な場合、TABは特別な動作をします。これはリージョンの各行をインデントするので、各行の最初の空白以外の文字は、先行する行に揃えられます。

プログラミング関連のモードでは、TABはコードのカレント行にたいして、前の行のコードの意味を理解してインデントします。リージョンがアクティブな場合、リージョンのすべての行はこの方法でインデントされます。ポイントが最初にカレント行のインデント領域にあった場合、その行の最初の空白以外の文字に再配置されます。

単にタブ文字をバッファーに挿入したいときは、`C-q` TABとタイプします (Section 4.1 [Inserting Text], page 17 を参照してください)。

21.1 インデントコマンド

TAB (`indent-for-tab-command`) コマンドとは別に、Emacs は他の方法でインデントを処理する、さまざまなコマンドを提供します。

C-M-o　カレント行をポイント位置で分割します (`split-line`)。その行のポイントの後にあるテキストは新しい行となり、ポイントがあった位置と同じ列にインデントされます。このコマンドは、最初にポイントを複数のスペースまたはタブを飛び越えて移動させます。その後、ポイントは挿入された行の前に配されます。

M-m　カレント行の最初の空白以外の文字に移動 (前方または後方) します (`back-to-indentation`)。その行に空白文字しかない場合は、その行の行末に移動します。

M-i　ポイント位置の空白文字を次のタブストップまでインデントします (`tab-to-tab-stop`)。Section 21.2 [Tab Stops], page 248 を参照してください。

M-x indent-relative
　ポイントが前の行 (実際には最後の空行以外の行) の、最初の空白以外の文字に揃うように、ポイント位置に空白文字を挿入します。ポイントがすでにその位置より右にある場合は、かわりに `tab-to-tab-stop`を実行します— ただし数引数を指定した場合は何もしません。

M-^ 前の行とカレント行をマージします (delete-indentation)。このコマンドはカレント行の前にある任意のインデントと、行の境界を 1 つのスペースで置き換えて、2 つの行を明解に結合 (join) します。

特別なケースとして、結合された文字が連続する開きカッコまたは閉じカッコの場合、または他の改行が続く場合、1 つのスペースは省略されます (Lisp コードにたいして有用)。

fill プレフィクスが存在して、それが改行の後ろにあった場合、M-^は fill プレフィクスを削除します。Section 22.6.3 [Fill Prefix], page 258 を参照してください。

プレフィクス引数を与えると、カレント行を後続行に結合します。リージョンがアクティブでプレフィクス引数が与えられなければ、かわりにリージョン内の全行を結合します。

C-M-\ リージョンのすべての行にたいして、各行の先頭で TABをタイプしたかのようにインデントします (indent-region)。

数引数が与えられた場合、その列番号までリージョンのすべての行をインデントします。

C-x TAB リージョン内で開始されるすべての行をインデントします。影響を受ける行は厳格な (rigid) 単位とによって移動されます (indent-rigidly)。

引数を指定しないで呼び出されたとき、このコマンドは影響を受ける行のインデントを対話的に調整するために、transient モードをアクティブにします。transient モードがアクティブの間は、LEFTまたは RIGHTにより、スペース 1 文字単位で左または右にインデントします。S-LEFTまたは S-RIGHTとタイプすることにより、タブストップ単位で左または右にインデントすることもできます。他のキーをタイプすると transient モードは無効になり、その後はこのキーは通知のように動作します。

プレフィクス引数 *n*を指定して呼び出すと、このコマンドは行を *n*個のスペースでインデントします (transient モードは有効になりません)。*n*に値を与えると後方にインデントするので、リージョン内の行のすべてのインデントを除去したいときは、以下のように充分に大きい負の引数を与えます。

```
C-u -999 C-x TAB
```

21.2 タブストップ

Emacs は、特定の列番号がタブストップになるよう定義しています。これらは Text モードおよびそれに関連するモードで、TABおよび M-iのようなコマンドにより、空白文字が挿入されるときのストップポイントとして使用されます。変数 tab-stop-listは、これらの位置を制御します。デフォルト値は nilで、これはタブストップが 8 列ごとという意味です。値には、タブストップの列番号を、0 基準の列番号のリスト (昇順) で指定することもできます。Emacs は最後の要素と、その 1 つ前の要素の差分を繰り返すことにより、リストを無限に拡張します。

変数 tab-stop-listを直接カスタマイズするかわりに、コマンド M-x edit-tab-stopsを通じて、タブストップを視認しながらセットする便利な方法があります。このコマンドは、以下のようなタブストップの説明を含んだバッファーに切り替えます。

```
          :       :       :       :       :       :
0         1         2         3         4
0123456789012345678901234567890123456789012345678
To install changes, type C-c C-c
```

最初の行は各タブストップ位置にコロンを示します。その後の 2 行はコロンが示す位置を表示します。tab-stop-listの値が nil(デフォルト) の場合、最初はコロンが表示されません。

異なるタブストップを置きたい列に、コロンを配して指定するために、このバッファーを編集できます。このバッファーは Overwrite モードを使用します (Section 20.2 [Minor Modes], page 241

を参照してください)。Emacs は、最後に明示した 2 つのストップの差分を繰り返すことにより、タブストップのリストを無限に拡張することを思い出してください。編集が終わったら、新しいタブストップを有効にするために、C-c C-cとタイプします。通常、新しいタブストップのセッティングは、すべてのバッファーに適用されます。しかし M-x edit-tab-stopsを呼び出したバッファーにたいして、ローカルな変数 tab-stop-listを作成していた場合 (Section 33.2.3 [Locals], page 511 を参照してください)、新しいタブストップのセッティングは、そのバッファーだけに適用されます。将来の Emacs セッションのためにタブストップのセッティングを保存するには、Customize インターフェースを使用して tab-stop-listの値を保存します (Section 33.1 [Easy Customization], page 499 を参照してください)。

このセクションで議論したタブストップは、バッファーでタブ文字が表示される方法には影響しないことに注意してください。タブ文字は常に次のタブストップ表示 *(display tab stop)* まで拡張される、空のスペースとして表示されます。Section 11.20 [Text Display], page 98 を参照してください。

21.3 タブ vs. スペース

インデントコマンドは通常、最小になるような一連のタブ文字とスペースを挿入 (または削除) することにより、望む列に行を揃えます。タブ文字は次のタブストップ表示 *(display tab stop)* まで伸長された、空のスペースとして表示されます。デフォルトでは、tab-width列ごと (デフォルトは 8) に、1 つのタブストップ表示があります。Section 11.20 [Text Display], page 98 を参照してください。

もし望むなら、すべてのインデントをスペースだけで行うこともできます。これを要求するには、バッファーローカルな変数 indent-tabs-modeを nilにセットします。バッファーローカルな変数についての情報は、Section 33.2.3 [Locals], page 511 を参照してください。しかし C-q TABは、indent-tabs-modeの値とは無関係に、常にタブ文字を挿入することに注意してください。

indent-tabs-modeを nilにセットする 1 つの理由は、すべてのエディターがタブ文字を同じ方法で表示するわけではないという理由です。Emacs ユーザーも同様で、彼らはカスタマイズされた tab-widthにより、異なる値をもつかもしれません。スペースだけを使うことにより、ファイルが常に同じように見えることを保証できます。Emacs でどのように見えるかだけに関心がある場合、この問題に取り組む他の方法は、ファイルローカルな変数 tab-widthをセットする方法です (Section 33.2.4 [File Variables], page 512 を参照してください)。

空白以外の文字の列を常に保ちながら、スペースをタブに、またはその逆の変換を行うコマンドも存在します。M-x tabifyは、リージョンの連続するスペースをスキャンして、インデントを変更せずに、少なくとも 2 文字の連続するスペースをタブに変換します。M-x untabifyは、リージョンのすべてのタブを適正な数のスペースに変更します。

21.4 インデントの便利な機能

変数 tab-always-indentは、TAB (indent-for-tab-command) コマンドの動作を微調整します。デフォルト値は tで、これは Chapter 21 [Indentation], page 247 で説明した動作を与えます。値をシンボル completeに変更すると、TABは最初にカレント行のインデントを試みます、すでにその行がインデントされている場合、このコマンドはポイント位置のテキストの補完を試みます (Section 23.8 [Symbol Completion], page 302 を参照してください)。値が nilの場合、TABはポイントが行の左端かインデント位置にあるときだけインデントを行い、それ以外はタブ文字を挿入します。

tab-always-indentが completeの場合には、tab-first-completion変数を通じて展開するか、それともインデントするかを更にカスタマイズできます。たとえばこの変数が eolなら、ポイントが行末にある場合のみ補完します。詳細は Section “Mode-Specific Indent” in *The Emacs Lisp Reference Manual* を参照してください。

Electric Indent モードはグローバルなマイナーモードで、これは RETをタイプしたあと自動的にインデントを行います。このモードはデフォルトで有効です。このマイナーモードを切り替えるには、M-x electric-indent-modeとタイプします。1 つのバッファーにたいしてモードを切り替えるには、M-x electric-indent-local-modeを使用してください。

22 人間の言語のためのコマンド

このチャプターではテキスト *(text)* — 人間の言語 (対照的なものとしてコンピューターのプログラム言語があります) における、文字シーケンスという意味 — にたいして動作する、Emacs コマンドを説明します。これらコマンドは、人間の言語の構文および文体の規則に配慮した方法で動作します。単語、センテンス (文)、パラグラフ (段落)、大文字に関する規則があります。フィル *(filling)* — パラグラフの行をおおよそ同じ長さに再配置するという意味 — を行うコマンドもあります。これらのコマンドは主にテキストの編集を意図しており、プログラムの編集にも便利なときがあります。

Emacs には、人間言語のテキストを編集するためのメジャーモードがいくつかあります。ファイルが普通のテキストを含む場合は、Text モードを使います。これは Emacs を、テキストの構文規則にたいして少しカスタマイズしたものです。Outline モードは。アウトライン構造でテキストを操作する特別なコマンドを提供します。Section 22.9 [Outline Mode], page 261 を参照してください。

Org モードは、Outline モードを拡張して Emacs を本格的なオーガナイザーに変えます。これにより TODO リストを管理したり、ノートを保存して、それを多くのフォーマットで公開することができます。Emacs と共に配布される Org の Info マニュアルを参照してください。

Emacs には TEX や LATEX (Section 22.11 [TeX Mode], page 268 を参照してください)、HTML や SGML (Section 22.12 [HTML Mode], page 273 を参照してください)、XML (Emacs と共に配布される nXLM モードのマニュアルを参照してください)、Groff や Nroff (Section 22.13 [Nroff Mode], page 274 を参照してください) のような、コマンドが埋め込まれたテキストのための、他のメジャーモードがあります。

テキスト文字で作られた AA(ASCII art) のピクチャーを編集する必要がある場合は、そのような絵を編集するための特別なメジャーモードの Picture モードを使用します Section “Picture Mode” in *Specialized Emacs Features* を参照してください。

22.1 単語

Emacs は、単語単位で移動するコマンドをいくつか定義しています:

`M-f` 1 単語前方に移動します (`forward-word`)。

`M-b` 1 単語後方に移動します (`backward-word`)。

`M-d` 単語の最後まで kill します (`kill-word`)。

`M-DEL` 単語の先頭まで kill します (`backward-kill-word`)。

`M-@` 次の単語の最後にマークをセットします (`mark-word`)。

`M-t` 2 つの単語を入れ替える、または他の単語を飛び越えて単語をドラッグします (`transpose-words`)。

これらのキーは文字単位のコマンド `C-f`、`C-b`、`C-d`、`DEL`、`C-t`に類似していることに注目してください。`M-@`は `C-@`(これは `C-SPC`の別名です) が由来です。

コマンド `M-f` (`forward-word`) および `M-b` (`backward-word`) は、単語単位で前方または後方に移動します。これらの `Meta`ベースのキーシーケンスは、1 文字単位で移動するキーシーケンス `C-f`および `C-b`に類似しています。類似点は数引数にも拡張されます。これらは繰り返し回数を指定します。`M-f`に負の引数を与えると後方に移動し、`M-b`に負の引数を与えると前方に移動します。前方への移動は単語の最後の文字の直後で停止し、後方への移動は、最初の文字の直前で停止します。

`M-d` (`kill-word`) は、ポイントの後ろの単語を kill します。正確に言うと、ポイントから、`M-f` で移動する位置までのすべてを kill します。したがってポイントが単語の途中にある場合、`M-d`はポ

イントの後だけを kill します。ポイントと次の単語の間に区切り文字がある場合、それは単語と一緒に kill されます（次の単語だけを kill して、その前にある区切り文字は kill したくないときは、M-f で最後まで移動して、M-DELで単語を後方に kill します)。M-dは M-fと同様の引数を取ります。

M-DEL (backward-kill-word) は、ポイントの前の単語を kill します。これはポイントから、M-bで移動する位置までのすべてを kill します。たとえばポイントが 'FOO, BAR' のスペースの後ろにある場合、これは 'FOO, ' を kill します。'FOO'だけを kill して、カンマとスペースは残したい場合は、M-DELのかわりに M-b M-dを使用します。

M-t (transpose-words) は、ポイントの前またはポイントを含む単語を、次の単語と交換します。単語間の区切り文字は移動されません。たとえば 'FOO, BAR' は、'BAR FOO,'ではなく、'BAR, FOO' に入れ替えられます。入れ替えについては、Section 13.2 [Transpose], page 132 を参照してください。

リージョンにたいして操作を行う際、単語単位で操作したいときは、コマンド M-@ (mark-word) を使用します。このコマンドは M-fで移動する位置にマークをセットします。このコマンドにつての詳しい詳細は、Section 8.2 [Marking Objects], page 54 を参照してください。

単語にたいするコマンドが理解する単語境界は、構文テーブル (syntax table) で制御されます。単語の区切り文字に、(たとえば) 任意の文字を使うことができます。Section "Syntax Tables" in *The Emacs Lisp Reference Manual* を参照してください。

以上に加えて、リージョンまたはバッファーに含まれる単語数のカウントとリポートを行う M-= (count-words-region) および M-x count-wordsコマンドについては、Section 4.9 [Position Info], page 24 を参照してください。

22.2 センテンス

センテンス (文) とパラグラフ (段落) を操作する Emacs コマンドは、単語を処理するコマンドと同様、ほとんどが Meta キーに割り当てられています。

M-a　センテンスの先頭に、後方へ移動します (backward-sentence)。

M-e　センテンスの最後に、前方へ移動します (forward-sentence)。

M-k　センテンスの最後まで、前方に kill します (kill-sentence)。

C-x DEL　センテンスの最初まで、後方に kill します (backward-kill-sentence)。

コマンド M-a (backward-sentence) および M-e (forward-sentence) は、センテンスの先頭または最後に移動します。これらのコマンドのバインディングは、行の先頭と最後に移動する C-aと C-eに似せて選ばれました。似ていない点は、M-aまたは M-eを繰り返すと、センテンス単位で連続して移動することです。

センテンスにたいして後方に移動すると、ポイントはセンテンスの最初の文字の直前に移動します。前方に移動したときは、センテンスを終了させる区切り文字の直後に移動します。どちらもセンテンスの境界にある空白文字にポイントを移動することはありません。

C-aと C-eが kill コマンド C-kをもつように、M-aと M-eにも対応する kill コマンドがあります。M-k (kill-sentence) は、ポイントからセンテンスの最後までを kill します。正の数引数 nを指定すると、次の nセンテンスを kill します。負の数引数 $-n$を指定すると、後方に n番目のセンテンスの先頭までを kill します。

C-x DEL (backward-kill-sentence) は、センテンスの先頭までを後方に kill します。

センテンスのコマンドは、センテンスの最後に 2 つのスペースを置くという American のタイピストの慣習に、ユーザーが従うと仮定します。つまりセンテンスの最後は '.'、'?'、'!'の後ろに改行

または 2 つのスペースがあると仮定し、その間に任意の個数の ‘)’、‘]’、‘'’、‘"’が許されます。パラグラフの先頭および最後は、センテンスの先頭および最後でもあります。この慣習にしたがうことにより、Emacs のセンテンスにたいするコマンドがセンテンスの最後のピリオドと、省略形を示すピリオドを区別できるので便利になります。

センテンスの間を 1 つのスペースにしたい場合は、`sentence-end-double-space`を `nil`にセットすることにより、センテンスのコマンドが 1 つのスペースで止まるようになります。しかし、これにはセンテンスの終わりのピリオドと、省略形を示すピリオドを区別できなくなるという欠点があります。したがって便利で信頼できる編集のために、2 つのスペースの慣習にしたがうことを推奨します。変数 `sentence-end-double-space`はフィルにも影響します (Section 22.6.2 [Fill Commands], page 256 を参照してください)。

変数 `sentence-end`は、センテンスの終了を認識する方法を制御します。非 `nil`の場合、その値は正規表現にすべきで、それはセンテンスの最後の数文字 (センテンスの後ろのスペースも含む) とのマッチに使用されます。値が `nil`(デフォルト) の場合、Emacs は `sentence-end-double-space`の値のような、変数条件に照らしてセンテンスの最後を計算します。

Thai のようないくつかの言語は、センテンスの最後を示すのにピリオドを使用しません。そのような場合は変数 `sentence-end-without-period`に `t`をセットします。

22.3 パラグラフ

パラグラフ (段落) を操作する Emacs コマンドも Meta キーに割り当てられています。

`M-{` 前のパラグラフの先頭に、後方へ移動します (`backward-paragraph`)。

`M-}` 次のパラグラフの最後に、前方へ移動します (`forward-paragraph`)。

`M-h` そのパラグラフ、または次のパラグラフの周囲にポイントとマークを配します (`mark-paragraph`)。

`M-{` (`backward-paragraph`) は、カレントのパラグラフ、または前のパラグラフの先頭に移動します (呼び出し時にポイントがある場所に依存する。パラグラフの定義は以下を参照)。同様に、`M-}` (`forward-paragraph`) は、カレントのパラグラフ、または次のパラグラフの最後に移動します。パラグラフの前に空行がある場合、`M-{`はその空行に移動します。

パラグラフにたいして操作をしたい場合、`M-h` (`mark-paragraph`) とタイプすることにより、パラグラフの周囲にリージョンがセットされます。たとえば `M-h C-w`は、パラグラフの周囲またはポイントの後ろを kill します。`M-h`は、ポイント位置のパラグラフの先頭にポイントを、最後にマークを配します。ポイントがパラグラフの間 (連続する空行の中、または境界) にある場合、`M-h`はポイントの後ろに続くパラグラフの周囲にリージョンをセットします。パラグラフの最初の行の前に空行がある場合、その空行はリージョンに含まれます。リージョンがすでにアクティブの場合、このコマンドはポイントを変更せずにマークをセットします。連続する `M-h`は、マークをパラグラフ単位で先にセットします。

パラグラフの定義はメジャーモードに依存します。Fundamental モード、同様に Text モードとそれに関連するモードでは、パラグラフと、その隣接するパラグラフを分けるのは 1 行以上の空行—空の行、スペースとタブだけからなる行、およびそれらに改ページ文字がついた行 — です。プログラミング言語に関するモードでは、通常パラグラフは同じような方法で定義されるので、パラグラフがないプログラムなどにたいしても、パラグラフのコマンドを使うことができます。

Text モードでは、インデントされた行自体では、パラグラフの区切りにはならないことに注意してください。インデントされた行をパラグラフの分割に使用したい場合は、かわりに Paragraph-Indent Text モードを使用します。Section 22.8 [Text Mode], page 261 を参照してください。

フィルプレフィクスをセットしている場合、フィルプレフィクスで始まっていないすべての行は、パラグラフの区切りとなります。Section 22.6 [Filling], page 256 を参照してください。

パラグラフ境界の正確な定義は、変数 paragraph-separateおよび paragraph-startにより制御されます。paragraph-startの値には、パラグラフを開始または分割する行にマッチする正規表現を指定します。paragraph-separateの値には、パラグラフの一部とならないような、パラグラフを分割する行にマッチする正規表現を指定します。新しいパラグラフを開始するパラグラフに含まれる行は、paragraph-startにはマッチしますが、paragraph-separateにはマッチしません。たとえば Fundamental モードでは、paragraph-startは"\f\\|[\t]*$"、paragraph-separateは"[\t\f]*$"です。

paragraph-startと paragraph-separateはテキストの左マージンにマッチしますが、行頭である必要はないので、パラグラフ関数がマージンのセッティングに応じてテキストリージョン内で同じように確実に機能させるために、これらの regexp に '^'を使用するべきでないことに注意してください。

22.4 ページ

改ページ文字 (formfeed character: ASCIIコード 12 で、'control-L'とも表示されます) でページに分割されているものがあり、Emacs ではそれらがエスケープシーケンス '^L'で表示されます (Section 11.20 [Text Display], page 98 を参照してください)。伝統的には、そのようなテキストファイルのハードコピーを印刷する場合、各改ページ文字で改ページされます。ほとんどの Emacs コマンドは、この文字をほかの文字と同様に扱うので、C-q C-lで挿入して、DELで削除などができます。それに加えて、Emacs はページ単位の移動や操作を行うコマンドを提供します。

M-x what-page
: ポイント位置のページ番号と、ページ内での行番号を表示します。

C-x [
: 前のページ境界にポイントを移動します (backward-page)。

C-x]
: 次のページ境界にポイントを移動します (forward-page)。

C-x C-p
: 現在のページ (または他のページ) の周囲に、ポイントとマークを配します (mark-page)。

C-x l
: 現在のページの行数を数えます (count-lines-page)。

M-x what-pageは、ファイルの先頭から数えたページ数と、そのページの行数をエコーエリアに表示します。

C-x [(backward-page) コマンドは、前のページ区切りの直後にポイントを移動します。ポイントがすでにページ区切りの直後にある場合は、そのページ区切りをスキップして、その前のページ区切りに移動します。数引数は繰り返し回数を指定します。C-x] (forward-page) コマンドは、次のページ区切りに前方に移動します。

C-x C-p (mark-page) コマンドは、ポイントを現在のページの先頭 (ページの先頭に隣接するページ区切りの後ろ)、マークをページの最後 (ページの最後に隣接するページ区切りの後ろ) に配します。

C-x C-p C-wは、ページを他の場所に移動するために kill する便利な方法です。C-x [または C-x]で他のページへ移動して、そこに kill されたページを yank すれば、すべてのページは再び適切に区切られます。C-x C-pのリージョンに後続のページ区切りだけが含まれるのは、これが期待したよう確実に機能させるためです。

C-x C-pに数引数を指定すると、現在のページから数えて何ページ目に移動するかを指定します。0 は現在のページ、1 は次のページ、−1 は前のページを意味します。

`C-x l` (`count-lines-page`) コマンドは、ページをどこで 2 つに分割するかの良い指標になります。これは現在のページの全行数と、カレント行の前にある行数と、後ろにある行数を以下のように表示します。

```
Page has 96 (72+25) lines
```

合計が 1 少ないのに注意してください。これはポイントが行の先頭にない場合は正しくなります。

変数 `page-delimiter`は、ページがどこで始まるかを制御します。この値にはページを分割する行の先頭にマッチする正規表現を指定します (Section 12.6 [Regexps], page 115 を参照してください)。この変数の値は通常`"^\f"`で、これは行の先頭の改ページ文字にマッチします。

22.5 クォーテーションマーク

テキストをクォートするための一般的な方法の 1 つは、typewriter convention(タイプライター方式) です。これは‘`''like this''`’のような straight apostrophes によるクォートや、‘`"like this"`’のようなダブルクォートを使用します。別の一般的な方法としては curved quote convention(曲クォート方式) があります。これは‘ `like this` ’や“ `like this` ”のように、左右のシングルクォーテーションマークやダブルクォーテーションマークを使用します。[1] テキストファイルにおいては、typewriter quotes はシンプルで可搬性があり、curved quotes はあいまいさが少なく見栄えもよくなります。

Electric Quote モードにより、curved quotes をタイプするのが簡単になります。オプションでタイプした文字`を‘ 、'を ’、``を“ 、''を ”に変換します。このデフォルトのクォートリストは、変数 `electric-quote-chars`をカスタマイズして変更できます。値には左シングルクォート、右シングルクォート、左ダブルクォート、右ダブルクォートに対応する、4 つの文字からなるリストを指定します。デフォルト値は`'(?‘ ?’ ?“ ?”)`です。

Electric Quote モードは、このモードをどこでアクティブにするかを制御する変数を通じて、カスタマイズできます。`electric-quote-paragraph`が非 `nil`の場合、テキストパラグラフ内でアクティブになります。`electric-quote-comment`が非 `nil`の場合、プログラミング言語のコメント内でアクティブになります。`electric-quote-string`が非 `nil`の場合、プログラミング言語の文字列定数内でアクティブになります。デフォルトでは、`electric-quote-string`が `nil`で、それ以外の変数が `t`です。

オプション `electric-quote-replace-double`を非 `nil`値にセットすることもできます。その場合には"とタイプすることにより、コンテキストに応じて適切な curved double quote が挿入されます。バッファー先頭や行ブレーク、空白、開カッコ、クォート文字の後には“ 、それ以外なら ”が挿入されます。

Electric Quote モードはデフォルトで無効になっています。これを単一のバッファーで切り替えるには `M-x electric-quote-local-mode`、グローバルに切り替えるには `M-x electric-quote-mode`とタイプします。1 回だけこれを抑制したいときは、`や'のかわりに、`C-q ` `と `C-q '`を使用します。Electric Quote モードが無効または非アクティブのときに curved quote を挿入する場合、‘ は `C-x 8 [`、 ’は `C-x 8 ]`、“ は `C-x 8 {`、 ”は `C-x 8 }`とタイプします。Section 4.1 [Inserting Text], page 17 を参照してください。`electric-quote-chars`の値は、これらのキーバインドには影響を与えないことに注意してください。これらの値は`electric-quote-mode`のキーバインドではなく、`global-map`のバインドのためのものです。

[1] curved single quote 文字、U+2018 LEFT SINGLE QUOTATION MARK と U+2019 RIGHT SINGLE QUOTATION MARK です。curved double quote 文字は U+201C LEFT DOUBLE QUOTATION MARK と U+201D RIGHT DOUBLE QUOTATION MARK です。これらの文字を表示できないテキスト端末では、Info リーダーはそれらを typewriter ASCII quote 文字として表示するかもしれません。

22.6 テキストのフィル

テキストをフィル *(fill)* するとは、指定した幅に収まるように行を分割することを意味します。Emacs はフィルを 2 つの方法で行います。Auto Fill モードでは、自己挿入文字によるテキストの挿入で、自動的にフィルされます。テキストを編集するときに使用できる、明示的なフィルコマンドもあります。

22.6.1 Auto Fill モード

Auto Fill モードは行が長くなりすぎたときに SPCや RETをタイプすると行を自動的に分割する、バッファーローカルなマイナーモード (Section 20.2 [Minor Modes], page 241 を参照) です。

`M-x auto-fill-mode`
: Auto Fill モードを有効または無効にします。

`SPC`
`RET`
: Auto Fill モード中は、適切なときに行を分割します。

モードコマンド `M-x auto-fill-mode`は、カレントバッファーの Auto Fill モードを切り替えます。他のマイナーモードと同様、正の数引数は Auto Fill モードを有効にし、負の引数は無効にします。特定のメジャーモードで Auto Fill モードを自動的に有効にするには、モードフックに `auto-fill-mode`を追加します (Section 20.1 [Major Modes], page 240 を参照してください)。Auto Fill モードが有効なときは、モードラインにモード指標の '`Fill`'が表示されます (Section 1.3 [Mode Line], page 9 を参照してください)。

Auto Fill モードは行が望ましい幅より長くなったときに、適切な位置で自動的に行を分割します。行分割は SPCか RETをタイプしたときだけ発生します。行の分割をさせずにスペースまたは改行を挿入したいときは、`C-q` SPCまたは `C-q C-j`とタイプします。`C-o`も行の分割を発生させずに改行を挿入します。

Auto Fill が行ブレークする位置は、その行にある文字に依存します。ASCIIに由来する文字、ラテン文字、および他のほとんどのスクリプトでは、単語を損なわないように Emacs はスペース文字で行を分割します。しかし CJK スクリプトでは、行は任意の 2 文字間での分割があり得ます (`kinsoku` ライブラリーをロードしていれば、Emacs は特別な規則が行ブレークを抑止するような位置における、特定の CJK 文字ペアー間での行分割を避けるだろう)。

Auto Fill モードが行を分割するときは、適応型フィルプレフィクス *(adaptive fill prefix)* にしたがうよう試みます。フィルプレフィクスがカレントパラグラフの最初の 1 行目、および/または 2 行目で推論できる場合、そのフィルプレフィクスは新しい行に挿入されます (Section 22.6.4 [Adaptive Fill], page 259 を参照してください)。そうでない場合、新しい行はその行で TABをタイプしたかのようにインデントされます (Chapter 21 [Indentation], page 247 を参照してください)。プログラミング言語に関するモードでは、行がコメントの途中で改行された場合、新しいコメント区切りが適切に挿入されてコメントが分割されます。

Auto Fill モードはパラグラフ全体を再フィルしません。これは行の分割はしますが、行のマージはしません。したがって、パラグラフの途中を編集しているときは、適切にフィルされていないパラグラフがあり得ます。これをフィルするには、フィルコマンドを明示的に呼び出してください。これは次のセクションで説明します。

長い行を表示時に折り返す類似機能として Visual Line モードがあります (Section 11.23 [Visual Line Mode], page 101 を参照)。

22.6.2 明示的なフィルコマンド

`M-q`
: カレントパラグラフをフィルします (`fill-paragraph`)。

`C-x f` フィルの列幅をセットします (`set-fill-column`)。

`M-x fill-region`
リージョンの各パラグラフをフィルします (`fill-region`)。

`M-x fill-region-as-paragraph`
リージョンを 1 つのパラグラフとしてフィルします。

`M-x center-line`
行を中央に揃えます。

コマンド `M-q` (`fill-paragraph`) は、カレントのパラグラフをフィルします。このコマンドは特定の最大列幅に適合するような方法で、パラグラフの改行を再配分して、パラグラフの余分なスペースやタブ文字を削除します。Auto Fill モードのように、このコマンドや他のフィルコマンドは通常はスペース文字で行ブレークしますが、CJK 文字にたいしてはこれらのコマンドはほとんどすべての 2 文字間で行を分割でき、禁則処理にしたがうこともできます。Section 22.6.1 [Auto Fill], page 256 を参照してください。

`M-q`は通常、ポイントがあるパラグラフに作用しますが、ポイントがパラグラフの間にあるときは、ポイントの後ろのパラグラフに作用します。リージョンがアクティブの場合、かわりにリージョンのテキストに作用します。`M-x fill-region`を呼び出して、リージョンのテキストを明確にフィルすることもできます。

`M-q`および `fill-region`は、パラグラフの境界を探すために、通常の Emacs の条件式を使用します。より細かく制御するには、ポイントとマークの間にあるすべてを 1 つのパラグラフとして再フィルする、`M-x fill-region-as-paragraph`を使用することができます。このコマンドはリージョンの空行を削除するので、分割されたテキストブロックは、1 つのブロックに結合されます。

`M-q`に数引数を指定した場合、これはテキストのフィルと同時に割付 *(justify)* も指示します。これは行の右端が正確にフィル列になるように、余分なスペースが挿入されることを意味します。余分なスペースを削除するには、引数を指定せずに `M-q`を使用します (`fill-region`と同じ)。

フィルのための行の最大幅は、バッファーローカルな変数 `fill-column`により指定されます。デフォルト値は 70 です (Section 33.2.3 [Locals], page 511 を参照してください)。カレントバッファーの `fill-column`をセットする一番簡単な方法は、コマンド `C-x f` (`set-fill-column`) を使用する方法です。数引数を指定すると、それを新しいフィル列として使用します。`C-u`だけを指定すると、このコマンドは現在のポイントの水平位置を `fill-column`にセットします。`fill-column`はその性質上、列単位で数えられることに注意してください。その列の実際の位置は、グラフィカルなディスプレイでは使用されているフォントに依存します。特に可変幅フォントを使用することによって、行が異なれば `fill-column`がディスプレイ上で異なる水平位置を占めることが起こり得ます。

コマンド `M-x center-line`は、現在のフィル列でカレント行を中央に揃えます。数引数 *n*を指定すると、*n*を中央に揃えた後、ポイントを先に移動します。このバインディングは Text モードのためのもので、利用可能なのは Text モードとそれに関連するモードだけです (Section 22.8 [Text Mode], page 261 を参照してください)。

デフォルトでは、Emacs は 2 つのスペースまたは改行が後にあるピリオドを、センテンスの終わりと判断します。後に 1 つのスペースしかないピリオドは、センテンスの終わりではなく、省略形を示します。それに合わせて、フィルコマンドも後に 1 つのスペースしかないピリオドでは、行を分割しません。変数 `sentence-end-double-space`を `nil`にセットした場合、フィルコマンドは後に 1 つのスペースしかないピリオドでも行を分割するようになり、各ピリオドの後に 1 つのスペースを配すようになります。他の効果とこの方法の欠点については、Section 22.2 [Sentences], page 252 を参照してください。

変数 colon-double-spaceが非 nilの場合、フィルコマンドはコロンの後に 2 つのスペースを配します。

行分割を許さない場所の条件を追加で指定するには、アブノーマルフック変数 fill-nobreak-predicateをカスタマイズします (Section 33.2.2 [Hooks], page 509 を参照してください)。このフックの各関数は Emacs が行を分割すべきと判断する位置で引数なしで呼び出されます。関数が非 nil値を返した場合には、Emacs はその位置で行を分割しません。このフックに指定できる関数には fill-single-word-nobreak-p (センテンスの最初の単語の後ろ、および最後の単語の前では分割しません)、fill-single-char-nobreak-p (空白文字が前置された 1 文字の単語の後ろでは分割しません)、fill-french-nobreak-p (‘(’の後ろ、および ‘)’、‘:’、‘?’の前では分割しません)、fill-polish-nobreak-p (非空白文字が前置された 1 文字の単語の後ろでも分割しません) が含まれます。

Emacs は Display fill column indicator モード (Section 11.16 [Displaying Boundaries], page 94 を参照) を使用することによって fill-columnの位置にインジケーターを表示できます。

22.6.3 フィルプレフィクス

フィルプレフィクス *(fill prefix)* 機能により、フィルされるパラグラフの各行が、特別な文字列 (インデントされたパラグラフの行頭につける複数のスペースなど) で開始されるようになります。フィルプレフィクスを明示的に指定することができます。明示的に指定しない場合、Emacs は自動的にそれを推測することを試みます (Section 22.6.4 [Adaptive Fill], page 259 を参照してください)。

C-x .　　フィルプレフィクスをセットします (set-fill-prefix)。

M-q　　現在のフィルプレフィクスで、パラグラフをフィルします (fill-paragraph)。

M-x fill-individual-paragraphs
　　リージョンをフィルします。インデントの変化は、新しいパラグラフの開始と判断されます。

M-x fill-nonuniform-paragraphs
　　リージョンをフィルします。パラグラフを分割する行だけを、新しいパラグラフの開始と判断します。

カレントバッファーのフィルプレフィクスを指定するには、使用したいフィルプレフィクスで開始される行に移動して、プレフィクスの最後にポイントを配し、C-x . (set-fill-prefix) とタイプします (C-xの後にピリオドです)。フィルプレフィクスをオフに切り替えるには、ポイントを行の先頭に移動して C-x . とタイプして、空のフィルプレフィクスを指定します。

フィルプレフィクスの効果がある間、フィルコマンドはフィルを行う前にパラグラフの各行のフィルプレフィクスを削除して、フィルを行った後で各行にフィルプレフィクスを挿入します (パラグラフの最初の行は変更されずに残ります。これは他の行と異なるように意図されていることがよくあるからです)。Auto Fill モードは、新しい行を作成するときにフィルプレフィクスの自動的な挿入も行います (Section 22.6.1 [Auto Fill], page 256 を参照してください)。C-oコマンドは、行の先頭で使用したときは、新しい行を作成してフィルプレフィクスを挿入します。コマンド M-^は逆に、削除する改行の後ろにフィルプレフィクスがあれば、プレフィクスを削除します。

たとえば fill-columnが 50 で、フィルプレフィクスが ‘;; ’の場合、以下のテキストにたいして M-qを行うと、

```
;; This is an
;; example of a paragraph
;; inside a Lisp-style comment.
```

結果は以下のようになります:

```
;; This is an example of a paragraph
;; inside a Lisp-style comment.
```

M-qおよびパラグラフのコマンドは、フィルプレフィクスで開始されない行を、パラグラフの開始と判断します。これは、最初の行を除く各行がインデントされている形式のパラグラフにたいして、良い結果を与えます。空行、またはプレフィクスが削除されたインデントされた行も、パラグラフを分割または開始します。これは各行にコメント区切りがある、複数パラグラフにわたるコメントを記述するときに望ましいものです。

M-x fill-individual-paragraphsを使用して、各パラグラフに自動的にフィルプレフィクスをセットできます。このコマンドはリージョンをパラグラフに分割して、インデントの変化を新しいパラグラフの開始として扱い、それらのパラグラフをフィルします。したがって 1 つのパラグラフのすべての行は、同じ量のインデントをもちます。このインデントは、そのパラグラフにたいするフィルプレフィクスの役目を果たします。

M-x fill-nonuniform-paragraphsも同様なコマンドですが、別の方法でリージョンをパラグラフに分割します。このコマンドは、(paragraph-separateで定義される) パラグラフ開始行だけを、新しいパラグラフの開始と判断します。これは 1 つのパラグラフの各行は異なる量のインデントをもつことを想定しており、パラグラフの中で最小のインデント量をフィルプレフィクスに使用します。このコマンドは最初の行のインデントが、パラグラフの他の行より少なくても多くても、良い結果をもたらします。

フィルプレフィクスは、変数fill-prefixに保存されます。変数の値は文字列で、フィルプレフィクスがないときはnilです。これはバッファーごとの変数です。変数の変更はカレントバッファーだけに影響しますが、変更できるデフォルト値も存在します。Section 33.2.3 [Locals], page 511 を参照してください。

テキストプロパティindentationは、パラグラフのインデント量を制御する別の方法を提供します。Section 22.14.5 [Enriched Indentation], page 277 を参照してください。

22.6.4 適応型フィル

フィルコマンドは、特定なケースでは自動的に適切なフィルプレフィクスを推測できます。空白文字または特定の区切り文字が行の先頭にあり、それがパラグラフの全行に適用されている場合です。

パラグラフに 2 行以上の行がある場合は、2 行目からフィルプレフィクスが取得されますが、それは 1 行目にもそれが存在する場合だけです。

パラグラフに 1 行しかない場合、フィルコマンドは、その行からフィルプレフィクスを取得するかもしれません。この決定は複雑です。なぜならそのような場合に妥当な、3 つの選択肢があるからです。

- 最初の行のプレフィクスを、パラグラフのすべての行に適用します。
- 後続の行を空白文字でインデントします。これにより最初の行のプレフィクスの下にテキストが並びますが、実際には最初の行からプレフィクスをコピーしません。
- 2 行目以降の行に特別なことは行いません。

これら 3 つの書式スタイルは一般的に使用されているものです。そのためフィルコマンドは、そのメジャーモードに出現するプレフィクスにもとづいて、望ましいスタイルを決定しようと試みます。判断基準は以下のとおりです。

最初の行で見つかったプレフィクスがadaptive-fill-first-line-regexpにマッチするか、それがコメント開始シーケンス (メジャーモードに依存します) の場合、見つかったプレフィクスをパ

ラグラフのフィルに使用しますが、それは後続の行でそれらがパラグラフの開始として振る舞わない場合に限られます。

上記以外の場合、見つかったプレフィクスは同じ量のスペースに変換され、それらのスペースは残りの行のフィルプレフィクスとして使用されますが、それは後続の行でそれらがパラグラフの開始として振る舞わない場合に限られます。

Text モード、および空行またはページ区切りだけがパラグラフを分割するモードでは、適応型フィルにより選択されるプレフィクスは、パラグラフの開始として振る舞わないので、常にフィルに使用できます。

変数 adaptive-fill-regexpは、どのような行開始がフィルプレフィクスの役目を果たすかを決定します。行がこの正規表現にマッチする文字で開始されるとき、プレフィクスとして使用されます。変数 adaptive-fill-modeを nilにセットした場合、フィルプレフィクスは自動的に選択されません。

変数 adaptive-fill-functionに関数をセットすることにより、より複雑な方法で自動的にフィルプレフィクスを選択する方法を指定できます。この関数は行の左端の直後のポイントで呼び出され、その行にもとづいた適切なフィルプレフィクスを返すべきです。この関数が nilを返した場合、adaptive-fill-regexpがフィルプレフィクスを見つける機会を与えられます。

22.7 大文字小文字変換コマンド

Emacs には、1 つの単語または任意の範囲のテキストを、大文字または小文字に変換するコマンドがあります。

M-l 後続の単語を小文字に変換します (downcase-word)。

M-u 後続の単語を大文字に変換します (upcase-word)。

M-c 後続の単語の先頭の文字を大文字に変換します (capitalize-word)。

C-x C-l リージョンを小文字に変換します (downcase-region)。

C-x C-u リージョンを大文字に変換します (upcase-region)。

M-l (downcase-word) は、ポイントの後ろの単語を小文字に変換して、その先にポイントを移動します。したがって、M-lを繰り返すと、単語を連続して変換します。M-c (capitalize-word) は、単語の最初の文字を大文字にして残りを小文字にしますが、M-u (upcase-word) はすべての文字を大文字に変換します。これらのコマンドはすべて、引数を与えると複数の単語を変換します。これらのコマンドはすべてが大文字の大きなテキストを、これらが混成されたものに変換するとき、特に有用です、なぜなら適切に M-l、M-u、M-cを使い分けて単語を変換しながら移動できるからです (変換が不要なときは M-fを使ってその単語をスキップします)。

負の引数を与えると。ポイントの前の指定した数の単語にたいして、大文字小文字の変換を適用しますが、ポイントは移動しません。これは大文字小文字を間違えてタイプしたとき有用です。大文字小文字の変換コマンドを実行した後、そのまま編集を続けられるからです。

単語の途中で単語の大文字小文字を変換するコマンドが与えられた場合、ポイントの後ろに続く単語の一部だけに変換が適用されます (これは M-d (kill-word) と互換性があります)。負の引数の場合は、ポイントの前にある単語の一部が変換されます。

他の大文字小文字の変換コマンドは C-x C-u (upcase-region) と C-x C-l (downcase-region) で、これらはポイントとマークの間にあるすべてのテキストを大文字または小文字に変換します。ポイントとマークは移動しません。

リージョンにたいする大文字小文字の変換コマンド upcase-regionおよび downcase-region は、通常では無効になっています。これは、これらを使おうと試みたとき、確認を求められることを意味します。確認にたいして同意するとコマンドが有効になり、それ以降は確認を求められなくなることを意味します。Section 33.3.11 [Disabling], page 527 を参照してください。

22.8 Text モード

Text モードは、人間の言語のテキストファイルを編集するためのメジャーモードです。.txtという拡張子で終わる名前のファイルは、通常は Text モードで開かれます。明示的に Text モードに切り替えるには、M-x text-modeとタイプしてください。

Text モードでは、改行とページ区切りだけがパラグラフを分割します。結果として、パラグラフはインデントすることができ、適応型フィルはパラグラフをフィルするときに、どのインデントが使用されているか決定します。Section 22.6.4 [Adaptive Fill], page 259 を参照してください。

Text モードでは TAB (indent-for-tab-command) コマンドは通常、カレント行をインデントするかわりに、次のタブストップまで空白文字を挿入します。詳細については、Chapter 21 [Indentation], page 247 を参照してください。

Text モードは、明示的にそれらを呼び出した場合をのぞき、コメントに間する機能をオフに切り替えます。これは構文テーブル (syntax table) を変更するので、アポストロフィーは単語の一部と判断されます (たとえば‘don't’は 1 つの単語と判断されます))。しかし単語がアポストロフィーで開始される場合、それは先頭の文字を大文字にするためのプレフィクスとして扱われます (たとえば M-cにより、‘''hello''’は‘''Hello''’に変換されます)。

パラグラフの最初の行をインデントしている場合は、Text モードではなく Paragraph-Indent Text モード (M-x paragraph-indent-text-mode) を使用するべきでしょう。このモーででは、パラグラフの間に空行を入れる必要はありません。最初の行のインデントだけでパラグラフの開始を判断するのに充分だからです。しかしパラグラフのすべての行がインデントされている場合は、サポートされません。メジャーモードを変更したくないとき (たとえばメール作成時など) は、M-x paragraph-indent-minor-modeを使って、等価なマイナーモードを有効にしてください。

Text モードでは、M-TABが ispell-complete-wordにバインドされます。このコマンドはバッファーのポイントの前にある単語を、スペル辞書を使用して補完します。Section 13.4 [Spelling], page 134 を参照してください。ウィンドウマネージャーが M-TABをウィンドウの切り替えに定義している場合は、かわりに ESC TABまたは C-M-iとタイプすることができます。

Text モードに入ると、モードフック text-mode-hookが実行されます (Section 20.1 [Major Modes], page 240 を参照してください)。

以下のセクションでは、Text モードから派生したいくつかのメジャーモードを説明します。派生したモードは、上記で説明した Text モードのほとんどの機能を継承します。とくに、Text モードから派生したモードは、それら自身のモードフックの前に、text-mode-hookを実行します。

22.9 Outline モード

Outline モードは Text モードから派生したメジャーモードで、アウトライン (outline: 概略、概要、要綱) を編集するために特化したモードです。このモードはアウトライン構造のエントリー間を操作したり、バッファーの一部を一時的に非表示にするコマンドを提供するので、アウトライン構造をより簡単に閲覧することができるでしょう。M-x outline-modeとタイプすることにより、Outline モードに切り替わります。Outline モードに入ることにより、フック text-mode-hook、およびそれに続けてフック outline-mode-hookが実行されます (Section 33.2.2 [Hooks], page 509 を参照してください)。

Outline モードで行を非表示にするコマンド (Section 22.9.4 [Outline Visibility], page 264 を参照してください) を使用した場合、それらの行は画面に表示されなくなります。非表示行の前にある表示された行の後ろには省略記号 (3 つのピリオド) が表示され、そこに隠れたテキストがあることを示します。連続する複数の行を非表示にした場合も、省略記号は 1 つだけです。

C-nや C-pのような、複数の行にたいして操作を行う編集コマンドは、非表示の行のテキストを、その前にある表示された行の一部として扱います。表示された行の最後にある省略記号を kill すると、省略記号に対応する後続のすべての非表示テキストを実際に kill します。

22.9.1 Outline マイナーモード

Outline minor モードは、メジャーモードである Outline モードと同じコマンドを提供する、バッファーローカルなマイナーモードで、他のメジャーモードと合わせて使うことができます。M-x outline-minor-modeとタイプして、カレントバッファーにたいして Outline minor モードを切り替えることができます。ファイルローカル変数のセッティングを使用して、特定のファイルにたいして有効にすることもできます (Section 33.2.4 [File Variables], page 512 を参照してください)。

メジャーモードの Outline モードは、C-cプレフィクスで特別なキーバインディングを提供します。Outline minor モードは、C-c @プレフィクスで同様なバインディングを提供します。これは他のメジャーモードの特別なコマンドとの競合を減らすためです (変数 outline-minor-mode-prefix は、使用するプレフィクスを制御します)。

outline-minor-mode-use-buttonsが非 nilなら、Outline マイナーモードは隠されたセクションを表示するために、省略記号に加えてヘッダーライン先頭のボタンも使用します。このボタンをマウスでクリックすれば、そのセクションの表示を切り替えることができます。この変数の値が insertなら、ボタンはそのバッファーのテキストに直接挿入されます。これによりボタン上で RETをタイプすれば、マウスで行うのと同じようにセクションの表示を切り替えることができます。値が in-marginsなら、Outline マイナーモードはセクションが隠されていることを示すためにウィンドウマージンを使用します。ボタンはアイコンとしてカスタマイズできます (Section 11.11 [Icons], page 87 を参照)。

ユーザーオプション outline-minor-mode-cycleが非 nilなら、アウトラインのヘッダー行で表示の可視性を巡回するキー TABと S-TABが有効になります (Section 22.9.4 [Outline Visibility], page 264 を参照)。TABではサブヘッダーの非表示、表示、すべてのカレントセクションの表示を巡回します。S-TABはバッファー全体にたいして同じことを行います。

22.9.2 アウトラインのフォーマット

Outline モードは、バッファーの行には 2 つのタイプがあると仮定します。それはヘッダー行 *(heading lines)* とボディー行 *(body lines)* です。ヘッダー行はアウトラインのトピックを表します。ヘッダー行は 1 つ以上のアスタリスク文字 (‘*’) で開始されます。アスタリスクの数はアウトライン構造でのヘッダーの深さを決定します。したがって 1 つの ‘*’がついたトピックは、メジャーなトピックになります。このヘッダーと、次の ‘*’が 1 つのヘッダーの間にある、‘*’が 2 つのヘッダー行は、そのヘッダーのサブトピックです。ヘッダー行ではないすべての行はボディー行です。ボディー行は、それの前にあるヘッダー行に属します。以下は例です:

```
* Food
This is the body,
which says something about the topic of food.

** Delicious Food
This is the body of the second-level header.
```

```
** Distasteful Food
This could have
a body too, with
several lines.

*** Dormitory Food

* Shelter
Another first-level topic with its header line.
```

ヘッダー行とそれに属するすべてのボディー行を合わせて、エントリー *(entry)* と呼びます。ヘッダー行と、その配下にあるすべてのヘッダー行、それらすべてのボディー行を、サブツリー *(subtree)* と呼びます。

ヘッダー行と判別する条件は、変数 `outline-regexp`をセットすることによりカスタマイズできます (これを行う推奨された方法は、メジャーモードの関数を使うか、ファイルローカル変数を使う方法です)。行の開始がこの regexp にマッチする行は、ヘッダー行と判断されます。(行の左端ではなく) 行の途中でのマッチは勘定に入りません。

テキストにたいするマッチの長さは、ヘッダーのレベルを決定します。長いマッチは、より深くネストされたレベルとなります。'`@chapter`'、'`@section`'、'`@subsection`'というコマンドが、ドキュメントをチャプター、セクション、サブセクションに分割するようなテキストフォーマッターがある場合 (訳注: texinfo 形式のこと)、`outline-regexp`に'`"@chap\\|@\\(sub\\)*section"`'をセットすることにより、それらの行をヘッダー行と判断させることができます。ここでトリックに注意してください。2 つの単語'`chapter`'と'`section`'は同じ長さです。しかし regexp'`chap`'だけにマッチするように定義することによって、チャプターのヘッダーにマッチするテキストの長さを短くすることができるので、Outline モードはセクションがチャプターの配下であることを知ることができます。これは'`@chap`'で始まるコマンドが他にない場合に限り機能します。

変数 `outline-level`をセットすることにより、ヘッダー行のレベルを計算するためのルールを明示できに指定できます。`outline-level`の値には、引数をとらないカレントヘッダーのレベルを返す関数を指定します。この変数をセットする推奨された方法は、メジャーモードのコマンドを使うか、ファイルローカル変数を使用する方法です。

22.9.3 アウトライン移動コマンド

Outline モードはヘッダー行を後方または前方に移動する、特別な移動コマンドを提供します。

`C-c C-n` ポイントを次の可視なヘッダー行に移動します (`outline-next-visible-heading`)。

`C-c C-p` ポイントを前の可視なヘッダー行に移動します (`outline-previous-visible-heading`)。

`C-c C-f` ポイントを同じレベルの、次の可視なヘッダー行に移動します (`outline-forward-same-level`)。

`C-c C-b` ポイントを同じレベルの、前の可視なヘッダー行に移動します (`outline-backward-same-level`)。

`C-c C-u` ポイントを、より低い (より包括的な) レベルの可視のヘッダー行に移動します (`outline-up-heading`)。

上記のコマンドはすべて、繰り返し回数として数引数を受け取ります。たとえば、`C-c C-f`に引数を与えると、その数だけ前方の可視なヘッダー行と同じレベルに移動し、`C-c C-u`に引数を与えると、ネストされたレベルをその数のレベル抜け出して移動します。

22.9.4 アウトライン表示コマンド

Outline モードには、アウトライン構造にもとづいてバッファーの一部を一時的に非表示にしたり可視にするコマンドが、いくつかあります。これらのコマンドはアンドゥ可能ではありません。コマンドの効果は単純にアンドゥメカニズムに記録される訳ではありませんが、それらを実行した直後に限りアンドゥできます (Section 13.1 [Undo], page 131 を参照してください)。

これらのコマンドは、カレントのヘッダー行に作用します。ポイントがヘッダー行にある場合、それがカレントのヘッダー行です。ポイントがボディー行にある場合、カレントヘッダー行は一番近くの前にあるヘッダー行です。

`C-c C-c` カレントヘッダー行のボディーを非表示にします (`outline-hide-entry`)。

`C-c C-e` カレントヘッダー行のボディーを可視にします (`outline-show-entry`)。

`C-c C-d` カレントヘッダー行の配下のすべて (ヘッダー行自身は含まれません) を非表示にします (`outline-hide-subtree`)。

`C-c C-s` カレントヘッダー行の配下のすべて (ボディー、サブヘッダーおよびそのボディーを含む) を可視にします (`outline-show-subtree`)。

`C-c C-l` カレントヘッダー行のボディーと、すべてのサブヘッダーを非表示にします (`outline-hide-leaves`)。

`C-c C-k` カレントヘッダー行のサブヘッダーをすべてのレベルで可視にします (`outline-show-branches`)。

`C-c C-i` カレントヘッダー行の一番近いサブヘッダー (1 レベル下) を可視にします (`outline-show-children`)。

`C-c C-t` バッファーのすべてのボディー行を非表示にします (`outline-hide-body`)。

`C-c C-a` バッファーのすべての行を可視にします (`outline-show-all`)。

`C-c C-q` 上位 *n*レベルのヘッダー行を除き、すべてを非表示にします (`outline-hide-sublevels`)。

`C-c C-o` ポイントのあるヘッダー行またはボディー行と、そこから最上位までのレベルにあるヘッダーを除き、すべてを非表示にします (`outline-hide-other`)。

これらのコマンドのうち、カレントヘッダー行に直接続くボディー行を非表示にする `C-c C-c` (`outline-hide-entry`) と、それらを可視にする `C-c C-e` (`outline-show-entry`) が一番簡単なコマンドです。サブヘッダーとそれらのボディーは影響を受けません。

コマンド `C-c C-d` (`outline-hide-subtree`) および `C-c C-s` (`outline-show-subtree`) は、より強力です。これらはカレントヘッダー行のサブツリー — つまりカレントヘッダー行のボディーと、すべての直接または間接のサブヘッダーとそのボディー — に適用されます。

コマンド `C-c C-l` (`outline-hide-leaves`) は、カレントヘッダー行のボディーと、サブツリーのボディーをすべて非表示にします (サブヘッダー自身は表示されます)。コマンド `C-c C-k` (`outline-show-branches`) は、(たとえば `C-c C-d`) などで) 前に非表示にされたサブヘッダーを可視にします。コマンド `C-c C-i` (`outline-show-children`) はこれの穏やかなバージョンで、直接的なサブヘッダー (たとえば 1 レベル下のサブヘッダー) を可視にします。

コマンド `C-c C-o` (`outline-hide-other`) は、ポイントのあるエントリーと、その祖先 (アウトライン構造上でそこから最上位レベルに至るまでのヘッダー)、およびトップレベルのヘッダーを除き、すべてを非表示にします。このコマンドは、そのバッファーの最初のヘッダーの前のボディー行も表示します。

残りのコマンドは、バッファー全体に作用するコマンドです。C-c C-t (outline-hide-body) はすべてのボディー行を非表示にするので、アウトライン構造だけを見ることができます (特別な例外として、ファイルの最初にあるヘッダー行より前の行は、技術的に言うとボディー行ですが、これらは非表示になりません)。C-c C-a (outline-show-all) はすべての行を可視にします。C-c C-q (outline-hide-sublevels) は、カレントのヘッダー行 (ポイントがヘッダー行にないときのデフォルトは 1 行目) の位置、またはその上にある最上位のヘッダーを除き、すべてを非表示にします。数引数 *n*を指定すると、上位 *n*レベルのヘッダー行を除き、すべてを非表示にします。この場合、上位 *n*レベルと、最初のヘッダーの前のボディー行が表示されることに注意してください。

Outline は各セクションおよびバッファー全体の可視性を循環させるために便利なコマンドも提供しています。ヘッダーで TAB (outline-cycle) とタイプすることでカレントセクションにたいして “hide all”、“subheadings”、“show all” の状態間を巡回できます。S-TAB (outline-cycle-buffer) とタイプすれば “only top-level headings”、“all headings and subheadings”、“show all” の状態間を巡回できます。

インクリメンタル検索が、Outline モードにより非表示にされたテキストを検索したとき、検索された部分のバッファーは可視になります。その位置で検索を抜けると、テキストは可視のまま残ります。アクティブなインクリメンタル検索が、非表示のテキストにマッチするかを切り替えるには、M-s iとタイプします。将来の検索にたいしてデフォルトを変更するには、オプション search-invisibleをカスタマイズします (このオプションは query-replace、および関連する機能が非表示のテキストを扱う方法に影響を与えます。Section 12.10.4 [Query Replace], page 124 を参照してください)。バッファーローカルなマイナーモード Reveal モード (M-x reveal-mode) を使用して、操作にしたがって自動的にテキストを可視にもできます。

outline-default-stateは Outline モードがオンに切り替わった後にどのヘッダーを可視にするかを制御する変数です。非 nilなら一部のヘッダーはアウトライン化されます。数値ならそれに相当するレベルを含めて、そのレベルヘッダーだけを表示、outline-show-allならバッファーのすべてのテキストを表示、outline-show-only-headingsならレベルとは無関係にヘッダーだけを表示、値が lambda 関数か関数名の場合には、モードが有効になった後にヘッダーの可視性の切り替えを担う関数として、その関数が引数なしで呼び出されることになります。

22.9.5 複数ビューによるアウトラインの閲覧

1 つのアウトラインを同時に 2 つのビューで、別のウィンドウで表示することができます。これを行うには、M-x make-indirect-bufferを使ってインダイレクトバッファーを作らなければなりません。このコマンドの最初の引数は既存のアウトラインバッファーの名前で、2 番目の引数は新しくインダイレクトバッファーとして使用する名前です。Section 16.6 [Indirect Buffers], page 183 を参照してください。

1 度インダイレクトバッファーが作成されると、C-x 4 bや他の Emacs コマンドで、通常のやり方でウィンドウにそれを表示できます。テキストの一部を可視にしたり非表示にする Outline モードのコマンドは、それぞれのバッファーで独立したエントリーを操作し、それぞれのバッファーは独自のビューをもつことができます。同じアウトラインにたいして 2 つ以上のビューが欲しいときは、追加でインダイレクトバッファーを作成します。

22.9.6 折り畳み編集

Foldout パッケージは、Outline モードおよび Outline minor モードを、フォールディング (folding: 折り畳み) コマンドで拡張します。フォールディングのアイデアは、アウトラインのネストされた部分にたいして、それに関連する部分と、より高いレベルが非表示のときに、それにズームインするというものです。

すべてのテキストと、レベル 1 より下のレベルが非表示になっている Outline モードのバッファーを考えてください。それらのヘッダーの下に何が隠れているか見るには、C-c C-e (M-x outline-show-entry) でボディーを可視にするか、C-c C-iでその子ヘッダー (レベル 2) を可視にすることができます。

Foldout では、C-c C-z (M-x foldout-zoom-subtree) を使います。これはそのヘッダーのボディーと、その子サブヘッダーを可視にして、バッファーをナローイングするので、レベル 1 にヘッダーとボディー、それにレベル 2 のヘッダーだけが可視になります。そのうちの 1 つのレベル 2 ヘッダーの下を見るには、カーソル位置をそのヘッダーに移動して、もう一度 C-c C-zを使用します。これにより、そのレベル 2 ヘッダーのボディーと、レベル 3 の子サブヘッダーが可視になり、バッファーが再びナローイングされます。連続して好きなだけサブヘッダーをズームインすることができます。モードラインの文字列には、今どの深さにいるのかが表示されます。

ヘッダーにズームインしているときに、それらの子サブヘッダーを見るには C-u C-c C-zのように数引数を指定します。何レベル下の子かを指定することもできます (M-x outline-show-children に相当します)。たとえば M-2 C-c C-z は 2 レベルの子サブヘッダーを可視にします。ボディーを指定するには M-- C-c C-z のように負の引数を指定します。C-c C-s (M-x outline-show-subtree) のようにサブツリー全体を展開するには、M-0 C-c C-z のように引数に 0 を指定します。

ズームインしている時でも、Outline モードの表示および非表示の関数は、Foldout に干渉せずに使用できます。バッファーがナローイングされているので、グローバルな編集操作は、ズームインされたヘッダーのテキストだけに影響します、これは特定のチャプターやセクションに変更を限定するのに便利です。

フォールドのズームを取り消す (抜ける) には、C-c C-x (M-x foldout-exit-fold) を使用します。これは最上位レベルのヘッダーの配下のすべてのテキストと、サブヘッダーを非表示にして、バッファーの以前のビューに戻ります。数引数はフォールドを何レベル抜けるかを指定します。0 を指定するとすべてのフォールドを抜けます。

テキストとサブヘッダーを非表示にせずに、フォールドのナローイングを取り消すには、負の引数を指定します。たとえば、M--2 C-c C-x は 2 つのフォールドを抜け出し、テキストとサブヘッダーは表示されたままにします。

Foldout モードはフォールドへの出入りと、非表示のテキストを表示するためのマウスコマンドも提供します:

C-M-mouse-1でヘッダーをクリックするとズームインします。

シングルクリック: ボディーを可視にします。

ダブルクリック: サブヘッダーを可視にします。

トリプルクリック: ボディーとサブヘッダーを可視にします。

4 連クリック: サブツリー全体を可視にします。

C-M-mouse-2でヘッダーをクリックしたときは可視になります

シングルクリック: ボディーを可視にします。

ダブルクリック: サブヘッダーを可視にします。

トリプルクリック: ボディーとサブヘッダーを可視にします。

4 連クリック: サブツリー全体を可視にします。

C-M-mouse-3でヘッダーをクリックすると、ヘッダーの配下のテキストを非表示にするか、フォールドを抜けます

シングルクリック: サブツリーを非表示にします。

ダブルクリック: フォールドを抜けてテキストを非表示にします。

トリプルクリック: テキストを非表示にせずにフォールドを抜けます
4 連クリック: すべてのフォールドを抜けてテキストを非表示にします。

`foldout-mouse-modifiers`をセットすることにより、(`Ctrl-Meta-`のかわりに) 他の修飾キーを指定できます。すでに `foldout.el`ライブラリーがロードされている場合、これが効果をもつために再ロードしなければなりません。

Foldout パッケージを使用するには、`M-x load-library RET foldout RET`とタイプするか、init ファイルに以下の行を記述して自動的にこれを行うことができます。

```
(with-eval-after-load "outline"
  (require 'foldout))
```

22.10 Org モード

Org モードは、Emacs をオーガナイザーおよび／またはオーサリングツールとして使用するための、Outline モードの変種です。名前が拡張子`.org`で終わるファイルは、Org モードで開かれます (Section 20.3 [Choosing Modes], page 244 を参照してください)。明示的に Org モードに切り替えるには、`M-x org-mode`とタイプしてください。

Org モードでは、Outline モードのように、各エントリーは 1 つ以上の '`*`'文字で始まるヘッダー行をもちます。Section 22.9.2 [Outline Format], page 262 を参照してください。それに加えて、'`#`'文字で始まる行は、コメントとして扱われます。

Org モードは、簡単にアウトライン構造を眺めたり操作するためのコマンドを提供します。それらのコマンドのうち、一番簡単なのは `TAB` (`org-cycle`) です。ヘッダー行でこのコマンドを呼び出すと、サブツリーを — (i) ヘッダー行だけを表示 (ii) ヘッダー行と、(あれば) 直接の子ヘッダー行だけを表示 (iii) サブツリー全体を表示 — の 3 つの異なる視点で巡回します。ボディー行にたいして呼び出された場合、`TAB`にグローバルにバインドされたコマンドが実行されます。

Org モードのバッファーの任意の場所で、`S-TAB` (`org-shifttab`) とタイプすることにより、アウトライン構造全体を — (i) 最上位^^e3^^87^^be ベルのヘッダー行だけを表示 (ii) ボディー行を除いたすべてのヘッダー行を表示 (iii) すべてを表示 — の 3 つの異なる視点で巡回します。

ヘッダー行で `M-<UP>` (`org-metaup`) または `M-<DOWN>` (`org-metadown`) とタイプすることにより、ボディー行と (あれば) サブツリーを含むバッファー内のエントリー全体を、上または下に移動できます。同様に `M-<LEFT>` (`org-metaleft`) および `M-<RIGHT>` (`org-metaright`) で、ヘッダー行を昇格または降格できます。ボディー行で呼び出された場合は、それらのキーにグローバルにバインドされたコマンドが実行されます。

以下のサブセクションでは、オーガナイザーやオーサリングシステムとして Org モードを使用するための基本的な操作方法を解説します。詳細については、Section "Introduction" in *The Org Manual* を参照してください。

22.10.1 オーガナイザーとしての Org

エントリー内のどこかで `C-c C-t` (`org-todo`) とタイプすることにより、その Org エントリーを *TODO* アイテムとして、タグ付けすることができます。これはヘッダー行にキーワード '`TODO`'を追加します。もう一度`C-c C-t`をタイプするとキーワードは'`DONE`'に切り替わり、さらに`C-c C-t`とタイプするとキーワード全体を除去します。`C-c C-t`で使用されるキーワードは、変数 `org-todo-keywords` を通じてカスタマイズすることができます。

エントリーを TODO として作成する他に、エントリー内で `C-c C-s` (`org-schedule`) とタイプすることにより、日付を割り当てることができます。これは Emacs Calender(Chapter 28 [Calendar/Diary], page 401 を参照してください) をポップアップして日付の入力を求め、選択された日付

と一緒に、ヘッダー行の下にタグ'SCHEDULED'を追加します。コマンド C-c C-d (org-deadline) も、タグ DEADLINEを使う以外は同じ効果をもちます。

1 度 Org ファイルに計画された TODO アイテムをもつと、C-c [(org-agenda-file-to-front) とタイプすることにより、アジェンダファイル *(agenda files)* のリストにそのファイルを追加できます。Org モードは、たとえばあなたの生活のさまざまな側面をオーガナイズするために、複数のアジェンダファイルを簡単に保守できるようにデザインされています。アジェンダファイルのリストは、変数 org-agenda-filesに格納されています。

アジェンダファイルのアイテムを閲覧するには、M-x org-agendaとタイプします。このコマンドは何を見たいのか、入力を求めます (今週に行なうすべての事項のリストか、特定のキーワードにマッチする TODO アイテムのリストか、など)。

22.10.2 オーサリングシステムとしての Org

Org ノートを見映えよくフォーマットしたり、エクスポートして公表したいと思うかもしれません。カレントバッファーをエクスポートするには、Org バッファーのどこかで C-c C-e (org-export-dispatch) とタイプします。このコマンドはエクスポートするフォーマットの入力を求めます。現在サポートされているフォーマットは、HTML、LaTeX、Texinfo、OpenDocument(.odt)、iCalendar、Markdown、man-page、PDF です。PDF のようないくつかのフォーマットは、システムに特定のツールがインストールされている必要があります。

1 度に複数のファイルを特定のディレクトリー (ローカルまたはネットワーク越し) にエクスポートするには、変数 org-publish-project-alistにプロジェクトのリストを定義しなければなりません。詳細は Org のドキュメントを参照してください。

Org はエクスポートされたドキュメントのテキストのフォーマットに適用する、シンプルなマークアップの仕組みをサポートします:

```
- This text is /emphasized/
- This text is *in bold*
- This text is _underlined_
- This text uses =a teletype font=

#+begin_quote
``This is a quote.''
#+end_quote

#+begin_example
This is an example.
#+end_example
```

詳細は、Section "Exporting" in *The Org Manual*、および Section "Publishing" in *The Org Manual* を参照してください。

22.11 TeX モード

TeX は Donald Knuth により記述された強力なテキストフォーマッターで、GNU Emacs と同様、フリーなソフトウェアーです。TeX フォーマットにはいくつかの変種があります。LaTeX は TeX のための簡略化された入力フォーマットです。DocTeX は LaTeX ソースが記述された特別なファイルフォーマットであり、ソースとドキュメントが合成されています。SliTeX は、時代遅れの特別な LaTeX 形式です。[2]

[2] これは LaTeX 由来の'slides'ドキュメントクラスにより置き換えられました。

Emacs は、それらの変種用に TEX のメジャーモードを提供します。それは Plain TEX モード、LATEX モード、DocTEX モード、SliTEX モードです。Emacs はバッファーの内容を調べて、適切なモードを選択します (通常これは TEX 様式のファイルを visit したときに自動的に呼び出される `tex-mode`コマンドにより行われる。Section 20.3 [Choosing Modes], page 244 を参照されたい)。ファイル内容がこれを決定するのに不十分な場合、Emacs は `tex-default-mode`に指定されたモードを選択します。デフォルト値は `latex-mode`です。Emacs が間違った選択をしたときは、コマンド `M-x plain-tex-mode`、`M-x latex-mode`、`M-x slitex-mode`、`doctex-mode`を使用して、正しい TEX モードの変種を選択できます。

以下のセクションでは、TEX モードとその変種の機能について記述しています。TEX に関連したモードは他にもいくつかありますが、このマニュアルには記述されていません。

- BibTEX モードは BibTEX ファイルのためのメジャーモードで、これは一般的には LATEX 文書の図書目録リファレンスを維持するのに使用されます。詳細については、コマンド `bibtex-mode` のドキュメント文字列を参照してください。
- RefTEX パッケージは、図書目録リファレンスを管理する LATEX モードとして使用されるマイナーモードを提供します。詳細については、Emacs とともに配布されている RefTEX の Info マニュアルを参照してください。
- AUCTEX パッケージは、TEX およびそれに関連するフォーマットを編集するための。より先進的な機能を提供します。これには Emacs バッファーで TEX をプレビューする機能も含まれます。BibTEX モードや RefTEX パッケージとは異なり、AUCTEX はデフォルトでは Emacs とともには配布されません。これは Package メニュー (Chapter 32 [Packages], page 490 を参照してください) を通じてダウンロードできます。インストールされた後に、パッケージに含まれる AUCTEX マニュアルを参照してください。

22.11.1 TEX 編集コマンド

`"` コンテキストに応じて ‘``’、‘"’、‘''’のどれかを挿入します (`tex-insert-quote`)。

`C-j` パラグラフの区切り (2 つの改行) を挿入して、前のパラグラフの釣り合いの取れていない大カッコ (braces) やドル記号をチェックします (`tex-terminate-paragraph`)。

`M-x tex-validate-region`
リージョン内のパラグラフの、釣り合いのとれていない大カッコやドル記号をチェックします。

`C-c {` ‘{}’を挿入して、ポイントをその間に配します (`tex-insert-braces`)。

`C-c }` 対応が取れていない、次の閉じ大カッコの後ろに、前方に移動します (`up-list`)。

TEX では文字 ‘"’は通常使用されません。かわりに ‘``’で始まり ‘''’で終わる引用が使用されます。したがって TEX モードは`"`キーを `tex-insert-quote`コマンドにバインドしています。これは空白文字または開き大カッコの後ろに ‘``’、バックスラッシュの後に ‘"’、それ以外の文字の場合は ‘''’を挿入します。

特別な例外として、ポイントの前のテキストが ‘``’か ‘''’のときに`"`をタイプすると、Emacs は前のテキストを 1 つの`"`で置き換えます。したがって、必要がある時は`""`とタイプして ‘"’を挿入できます (`C-q "`を使用してこの文字を挿入することもできます)。

TEX モードでは、‘$’は特別な構文コードを持っていて、それは TEX の数式モードの区切りを理解しようと試みます。数式モードを抜けるために ‘$’を入力した場合、数式モードに入るための対応する ‘$’の位置が 1 秒間表示されます。これは閉じ大カッコが挿入されたとき、それに対応する開き大カッコが表示されるのと同じき機能です。しかし ‘$’が数式モードに入るためなのか、それとも抜けるため

なのかを指示する方法はありません。したがって、もし対応するものがある場合、実際にはそれが関係なくても、前の '$'の位置が表示されます。

TEX は大カッコを、対応が取れていなければならない区切りとして使用します。これを 1 つずつ挿入するより、つねに大カッコの対応が取られている方を好むユーザーもいます。C-c { (tex-insert-braces) を使うと、対になった大カッコを挿入します。これはポイントを 2 つの大カッコの間に配すので、中のテキストを挿入することができます。その後でコマンド C-c } (up-list) を使用すると、前方の閉じ大カッコの先に移動します。あるテキストをマークした後に C-c {を呼び出すこともでき、その場合このコマンドはマークされたテキストを大カッコで括ります。

対応の取れていない大カッコをチェックするコマンドが 2 つあります。C-j (tex-terminate-paragraph) は、ポイントの前のパラグラフをチェックして、新しいパラグラフを開始するための 2 つの改行を挿入します。対応が取れていないものが見つかった場合、エコーエリアにメッセージを出力します。M-x tex-validate-regionはリージョンを、パラグラフごとにチェックします。エラーは *Occur*バッファーにリストされます。そのバッファーでは、特定のミスマッチを visit する C-c C-c などの、通常の Occur モードのコマンドを使用できます (Section 12.11 [Other Repeating Search], page 126 を参照してください)。

TEX の Emacs コマンドは大カッコだけではなく、角カッコ (square brackets) やカッコ (parentheses) などもカウントすることに注意してください。これは TEX 構文をチェックする目的としては、厳密に正しいとは言えません。しかしカッコと角カッコはテキストの中で、同じような対応の取れた区切りとして使用され、さまざまな移動コマンドや、対応する区切りの表示が、それらにたいして機能するのは便利なのです。

22.11.2 LATEX 編集コマンド

LATEX モードは、plain TEX には適用できない特別な機能をいくつか提供します:

C-c C-o　LATEX ブロックのための '\begin'と '\end'を挿入して、ポイントをそれらの間の行に配します (latex-insert-block)。

C-c C-e　まだ閉じていない一番内側の LATEX ブロックを閉じます (latex-close-block)。

LATEX 入力では、テキストをブロック化するのに '\begin'と '\end'のタグが使用されます。ブロックを挿入するには C-c C-o (latex-insert-block) とタイプします。これはブロックタイプの入力を求め、適切な対応する '\begin'と '\end'を挿入し、その 2 行の間に空行を残してポイントをそこに移動します。

C-c C-oにたいしてブロックタイプを入力するとき、通常の補完コマンドを使用できます (Section 5.4 [Completion], page 31 を参照してください)。デフォルトの補完リストには、標準的な LATEX のブロックタイプが含まれています。補完にブロックタイプを追加したい場合は、リスト変数 latex-block-namesをカスタマイズしてください。

LATEX 入力では、'\begin'と '\end'のタグは対応が取れていなければなりません。C-c C-e (latex-close-block) により、対応が取れていない最後の '\begin'に対応する '\end'を挿入することができます。これは対応する '\begin'に調和するように '\end'をインデントし、ポイントが行の先頭にあるときは '\end'タグの後ろに改行を挿入します。マイナーモード latex-electric-env-pair-modeは、'\end'か '\begin'の一方をタイプしたとき、自動的に対応する '\end'または '\begin'を挿入します。

22.11.3 TEX 印刷コマンド

バッファーの全内容、または一部 (たとえば大きなドキュメントの 1 つのチャプター) にたいして、TEX を Emacs のサブプロセスとして呼び出すことができます。

C-c C-b　カレントバッファー全体にたいして TEX を呼び出します (tex-buffer)。

`C-c C-r` バッファーのヘッダーとともに、カレントリージョンにたいして TEX を呼び出します (`tex-region`)。

`C-c C-f` カレントファイルにたいして TEX を呼び出します (`tex-file`)。

`C-c C-v` 最後の `C-c C-b`、`C-c C-r`、`C-c C-f`コマンドの出力をプレビューします (`tex-view`)。

`C-c C-p` 最後の `C-c C-b`、`C-c C-r`、`C-c C-f`コマンドの出力を印刷します (`tex-print`)。

`C-c TAB` カレントファイルにたいして BibTEX を呼び出します (`tex-bibtex-file`)。

`C-c C-l` TEX 出力を表示するウィンドウを再センタリングして、最後の行が見えるようにします (`tex-recenter-output-buffer`)。

`C-c C-k` TEX サブプロセスを kill します (`tex-kill-job`)。

`C-c C-c` カレントバッファー全体にたいして、他のコンパイルコマンドを呼び出します (`tex-compile`)。

カレントバッファーを TEX に渡すには、`C-c C-b` (`tex-buffer`) とタイプします。フォーマットされた出力は、通常`.dvi`という一時ファイルに出力されます。その後で `C-c C-v` (`tex-view`) とタイプして、`xdvi`のような外部プログラムを起動して、出力ファイルを閲覧することができます。`C-c C-p` (`tex-print`) とタイプして出力ファイルのハードコピーを印刷することもできます。

デフォルトでは `C-c C-b`は、カレントディレクトリーで TEX を実行します。TEX の出力もこのディレクトリーに作成されます。TEX を違うディレクトリーで実行するには、変数 `tex-directory` を望むディレクトリー名に変更します。環境変数 `TEXINPUTS`に相対名が含まれていたり、ファイルに含まれる '`\input`'コマンドが相対ファイル名の場合、`tex-directory`を`"."`にしないと、望ましくない結果となるでしょう。そうでない場合は、`"/tmp"`などの他のディレクトリーを指定しても安全です。

そのバッファーにたいする TEX の変種は、`C-c C-b`で実際に実行されるシェルコマンドを決定します。Plain TEX モードでは、これは変数 `tex-run-command`で指定され、デフォルトは`"tex"`です。LATEX モードでは、これは `latex-run-command`で指定され、デフォルトは`"latex"`です。`.dvi`を閲覧するために `C-c C-v`で実行されるシェルコマンドは、TEX の種類に関係なく、変数 `tex-dvi-view-command`で決定されます。出力を印刷するために `C-c C-p`で実行されるシェルコマンドは、変数 `tex-dvi-print-command`で決定されます。TEX でコンパイルされたファイルを閲覧および印刷するために要求されるファイル拡張子は、変数 `tex-print-file-extension`でセットできます。たとえば、この変数に`.pdf`をセットして、それに合わせて `tex-dvi-view-command`と `tex-dvi-print-command`、同様に `latex-run-command`や `tex-run-command`を更新できます。

Emacs は出力ファイル名に通常、前のパラグラフで説明したシェルコマンド文字列を自動的に追加します。たとえば `tex-dvi-view-command`が`"xdvi"`のとき、`C-c C-v`は `xdvi` *`output-file-name`*を実行します。しかし、ファイル名がコマンドに埋め込まれている必要があるケース、たとえばあるコマンドの引数にファイル名を与えて、そのコマンドの出力をパイプで他のコマンドに渡さなければいけない場合があります。コマンド文字列の '`*`'で、ファイル名をどこに置くか指定することができます。以下は例です

```
(setq tex-dvi-print-command "dvips -f * | lpr")
```

エラーメッセージを含む TEX からの端末出力は、`*tex-shell*`という名前のバッファーに表示されます。TEX でエラーが発生した場合、このバッファーに切り替えて、適切な入力を与えることができます (これは Shell モードで機能します。Section 31.5.2 [Interactive Shell], page 459 を参照してください)。このバッファーに切り替えなくても、これをスクロールできるので、`C-c C-l`とタイプして最後の行を表示することができます。

これ以上の出力が有用でないと判断した時は、C-c C-k (tex-kill-job) とタイプして TEX プロセスを kill します。C-c C-bおよび C-c C-rの使用するときも、TEX がまだ実行中の時は kill します。

C-c C-r (tex-region) とタイプして、任意のリージョンを TEX に渡すことができます。しかし、これはトリッキーです。なぜならほとんどの TEX 入力ファイルは、先頭にパラメーターをセットしたりマクロを定義するコマンドを含んでいるからです。この問題を解決するために、C-c C-rでは、必須のマクロを含むファイル部分を指定することができます。指定したリージョンの前にそれを含めて、TEX の入力の一部とします。ファイルの必須と指定された部分をヘッダーと呼びます。

Plain TEX モードでヘッダーの境界を示すには、ファイルに 2 つの特別な文字列を挿入します。これはヘッダーの前に '%**start of header'、ヘッダーの後ろに '%**end of header'を記述します。これらの文字列は 1 行に全体を記述しなければなりませんが、文字列の前または後ろに他のテキストがあっても構いません。この 2 つの文字列を含む行はヘッダーの中に含まれます。'%**start of header'がバッファーの先頭 100 行にない場合、C-c C-rはヘッダーがないとみなします。

LATEX モードでは、ヘッダーは '\documentclass'または '\documentstyle'で始まり、'\begin{document}'で終わります。いかなる場合でも LATEX がこれらを要求するので、ヘッダーを識別するのに特別なことをする必要はありません。

コマンド (tex-buffer) および (tex-region) は、すべてを一時ディレクトリーで処理します。そして TEX でクロスリファレンスのために必要となる補助的なファイルは利用不可です。一般的にこれらのコマンドは、すべてのクロスリファレンスが正しい必要がある最終コピーのために実行するのには適していません。

クロスリファレンスのための補助的なファイルを使いたいときは、C-c C-f (tex-file) を使用します。これはカレントバッファーのファイルにたいして、そのファイルのディレクトリーで TEX を実行します。TEX を実行する前に、変更されたバッファーを保存するか確認が求められます。一般的には、クロスリファレンスを正しく取得するために、2 回 (tex-file) を使用する必要があります。

変数 tex-start-optionsの値は、TEX を実行するためのオプションを指定します。

変数 tex-start-commandsの値は、TEX を開始する TEX コマンドを指定します。デフォルト値は TEX をノンストップモードで実行します。対話的に TEX を実行するときは、変数に""をセットします。

大きなサイズの TEX ドキュメントは複数のファイル — 1 つはメインファイルで、他はサブファイル — に分割されているときがあります。サブファイルにたいして TEX を実行しても通常は動作しません。メインファイルにたいして実行する必要があるのです。サブファイルを編集するのに tex-file を使えるようにするには、変数 tex-main-fileにメインファイルの名前をセットします。その後は tex-fileはそのファイルにたいして TEX を実行します。

tex-main-fileを使用する一番簡単な方法は、各サブファイルのローカル変数リストにそれを指定する方法です。Section 33.2.4 [File Variables], page 512 を参照してください。

LATEX ファイルにたいしては、BibTEX を使用してカレントバッファーのファイルのための補助的なファイルを処理できます。BibTEX はデータベースの図書目録の引用 (bibliographic citations) を探して、図書目録のセクション (bibliography section) のための引用文献 (cited references) を準備します。コマンド C-c TAB (tex-bibtex-file) は、カレントバッファーのファイルにたいする '.bbl'ファイルを生成するためにシェルコマンド (tex-bibtex-command) を実行します。一般的に、'.aux'ファイルを生成するために一度 C-c C-f (tex-file) を行う必要があります。その後 C-c TAB (tex-bibtex-file) を行ってから、さらなるクロスリファレンスを正しく取得するために C-c C-f (tex-file) を 2 回行います。

カレントの TEX バッファーで、他のコンパイルプログラムを呼び出すには、C-c C-c (tex-compile) をタイプします。このコマンドは pdflatex、yap、xdvi、dvipsを含む、多くの

一般的なプログラムに渡す引数を知っています。標準の補完キーを使用して望ましいコンパイルプログラムを選択できます (Section 5.4 [Completion], page 31 を参照してください)。

22.11.4 TEX モード、その

TEX モードの変種に入ると、フック `text-mode-hook`および `tex-mode-hook`を実行します。その後、`plain-tex-mode-hook`、`doctex-mode-hook`、`latex-mode-hook`、`slitex-mode-hook`のうち、適切なものを実行します。TEX シェルを開始すると、フック `tex-shell-hook`を実行します。Section 33.2.2 [Hooks], page 509 を参照してください。

コマンド `M-x iso-iso2tex`、`M-x iso-tex2iso`、`M-x iso-iso2gtex`、`M-x iso-gtex2iso` は Latin-1 でエンコードされたファイルと、TEX でエンコードされた等価なファイルの変換に使用できます。

22.12 SGML モードと HTML モード

SGML および HTML のためのメジャーモードは、インデントのサポートとタグを操作するコマンドを提供します。

HTML は 2 つのモードから構成されます — 1 つ目は `html-mode`と呼ばれる基本的なモードで、これは若干カスタマイズされた SGML モードの変種です。もう 1 つはデフォルトで HTML ファイルに使用されるもので `mhtml-mode`モードと呼ばれ、`<script>`要素で囲まれた Javascript、および `<style>`要素内に埋め込まれた CSS を正しく処理することを意図しています。

`C-c C-n` 対話的に特殊文字を指定して、その文字に対応する SGML の '`&`'コマンドを挿入します (`sgml-name-char`)。

`C-c C-t` 対話的にタグとタグの属性を指定します (`sgml-tag`)。このコマンドはタグ名と属性値を尋ね、開始タグと終了タグの両方を挿入し、ポイントをその間に置きます。

プレフィクス引数 *n*を指定すると、このコマンドはバッファーのポイントの後ろにある *n*個の単語の周囲にタグを配します。リージョンがアクティブなときは、タグをリージョンの周辺に配します (Transient Mark モードがオフのときは、引数に −1 を与えることにより、これを行います)。

`C-c C-a` 対話的にカレントタグの属性値を挿入します (`sgml-attributes`)。

`C-c C-f` 対応の取れたタグのグループ (開始タグと対応する終了タグまでの範囲) をスキップします (`sgml-skip-tag-forward`)。数引数は繰り返し回数として振る舞います。

`C-c C-b` 対応の取れたタグのグループ (開始タグと対応する終了タグまでの範囲) を後方にスキップします (`sgml-skip-tag-backward`)。数引数は繰り返し回数として振る舞います。

`C-c C-d` ポイント位置またはポイントの後ろのタグを削除し、それに対応するタグも削除します (`sgml-delete-tag`)。ポイントの後のタグが開始タグの場合は、終了タグも削除します。終了タグの場合は、開始タグも削除します。

`C-c ?` *tag* `RET`
タグ *tag*の意味の説明を表示します (`sgml-tag-help`)。引数 *tag*が空のときは、ポイント位置のタグを説明します。

`C-c /` 一番内側の終了されていないタグの終了タグを挿入します (`sgml-close-tag`)。タグまたはコメントの中で呼び出されたときは、終了タグを挿入するかわりにそれを終了させます。

C-c 8 挿入した Latin-1 文字を、その文字自身ではなく、その文字を表す SGML コマンドを挿入するマイナーモードを切り替えます (sgml-name-8bit-mode)。

C-c C-v カレントバッファーを SGML として評価するシェルコマンド (要指定) を実行します (sgml-validate)。(HTML モードでは、このキーシーケンスは違うコマンドを実行する。)

C-c TAB バッファーの既存のタグの可視・不可視を切り替えます。これは簡単なプレビューとして使用できます (sgml-tags-invisible)。

XML ドキュメントを編集するためのメジャーモードは nXML モードと呼ばれます。このモードは多くの既存の XML スキーマを認識することができ、M-TABを通じて XML エレメントの補完、同様にエラーをハイライトするオンザフライ (on-the-fly) な XML の妥当性検証を提供します。既存のバッファーで nXML モードを有効にするには、M-x nxml-modeとタイプするか、M-x xml-modeとタイプしても同じです。Emacs は.xmlという拡張子をもつファイルにたいして nXML モードを使用します。.xhtmlという拡張子をもつ XHTML ファイルにたいしては、デフォルトで Emacs は HTML モードを使用します。変数 auto-mode-alistをカスタマイズすることにより nXML モードを使用させることができます (Section 20.3 [Choosing Modes], page 244 を参照してください)。nXML モードについては、Emacs とともに配布されている Info マニュアルで説明されています。

XML は SGML の厳格なサブセットなので、XML を編集するのに、それほど強力ではない SGML モードの使用を選択することもできます。既存のバッファーで SGML モードを有効にするには、M-x sgml-modeとタイプします。SGML を有効にするとき、Emacs はバッファーが XML かどうかを検証します。もし XML の場合、変数 sgml-xml-modeに非 nil値をセットします。これにより上述した SGML モードのタグ挿入コマンドは、常に明示的に終了タグを挿入するようになります。

22.13 Nroff モード

Nroff モードは Text モードから派生した、nroff ファイル (たとえば Unix の man) の編集に特化したメジャーモードです。M-x nroff-modeとタイプすることによりこのモードにはいります。Nroff モードに入ることにより、フック text-mode-hook、その後で nroff-mode-hookが実行されます (Section 33.2.2 [Hooks], page 509 を参照してください)。

Nroff モードでは、nroff コマンド行はパラグラフの区切りとして扱われ、ページは '.bp'で区切られ、コメントはバックスラッシュとダブルクォートで始まります。これは以下のコマンドも定義します:

M-n nroff コマンドではない次の行の先頭に移動します (nroff-forward-text-line)。引数は繰り返し回数です。

M-p M-nと同様ですが上に移動します (nroff-backward-text-line)。

M-? エコーエリアにリージョンの、(nroff コマンドではない) テキストの行数を表示します (nroff-count-text-lines)。

Electric Nroff モードはバッファーローカルなマイナーモードで、Nroff モードとともに使うことができます。このマイナーモードを切り替えるには、M-x nroff-electric-modeとタイプします (Section 20.2 [Minor Modes], page 241 を参照してください)。このモードがオンのとき、グループ化を行う種類の nroff コマンドを含む行を RETをタイプして終端すると、グループ化を閉じる nroff コマンドが自動的に後続の行に挿入されます。

Nroff モードとともに Outline minor モード (Section 22.9 [Outline Mode], page 261 を参照してください) を使用する場合、ヘッダー行は '.H'の後に数字 (ヘッダーレベル) が続く形式の行です。

22.14 Enriched テキスト

Enriched(多くの刺激を含む、豊かにした) は、フォーマットされたテキストファイルを WYSIWYG(What You See Is What You Get) な方法で編集するためのマイナーモードです。Enriched モードが有効な場合、フォントやカラーなどのさまざまなフォーマットプロパティを、バッファーのテキストに適用できます。バッファーを保存するときは、それらのプロパティはテキストとともに、MIME 形式 '`text/enriched`'のファイルフォーマットを使用して保存されます。

Enriched モードは通常、Text モードとともに使用されます (Section 22.8 [Text Mode], page 261 を参照してください)。これは多くのメジャーモード (構文のハイライトのために Font Lock モードを使用するほとんどのプログラミング言語関連のモードを含む) で使用される、Font Lock モードとの互換性はありません。Enriched モードとは異なり、Font Lock モードはカレントバッファーの内容にもとづいて、テキストのプロパティを自動的に割り当てます。これらのプロパティはディスクには保存されません。

Emacs の `data-directory`のファイル `enriched.txt`は、Enriched モードの機能例として役に立ちます。

22.14.1 Enriched モード

Enriched モードは、バッファーローカルなマイナーモードです (Section 20.2 [Minor Modes], page 241 を参照してください)。'`text/enriched`'フォーマットで保存されたファイルを visit すると、Emacs は自動的に Enriched モードを有効にし、ファイル内のフォーマット情報をバッファーのテキストに適用します。Enriched モードが有効なときにバッファーを保存すると、ファイルはフォーマット情報を含む '`text/enriched`'フォーマットで保存されます。

フォーマットされたテキストで新しいファイルを作成するには、存在しないファイルを visit して `M-x enriched-mode`とタイプします。このコマンドは実際に Enriched モードを切り替えます。プレフィクス引数を指定した場合、引数が正のときは Enriched モードを有効にし、それ以外は Enriched モードを無効にします。Enriched モードを無効にすると、Emacs は '`text/enriched`'フォーマットでバッファーを保存しなくなります。バッファーに追加されたフォーマットプロパティはバッファーに残りますが、ディスクには保存されません。

Enriched モードは、すべての Emacs のテキストプロパティを保存しません。変数 `enriched-translations`に指定されたものだけを保存します。これにはフォント、カラー、インデント、行端揃えのプロパティが含まれます。

ファイルを visit したとき、ファイルが '`text/enriched`'フォーマットだと Emacs が認識できなかった場合は、`M-x format-decode-buffer`とタイプします。このコマンドはファイルフォーマットの入力を求め、そのフォーマットでファイルを再読み込みします。'`text/enriched`'フォーマットを指定することにより、自動的に Enriched モードが有効になります。

'`text/enriched`'ファイルを raw 形式 (フォーマットされたテキストではなく、マークアップタグを伴う通常テキスト) として閲覧するには、`M-x find-file-literally`を使用します (Section 15.2 [Visiting], page 146 を参照してください)。

Emacs が '`text/enriched`'のようなファイルフォーマットを認識・変換する方法については、Section "Format Conversion" in *the Emacs Lisp Reference Manual* を参照してください。テキストプロパティについての詳細は、Section "Text Properties" in *the Emacs Lisp Reference Manual* を参照してください。

22.14.2 ハード改行とソフト改行

Enriched モードでは、Emacs は 2 つの異なる改行、ハード改行とソフト改行を区別します。`M-x use-hard-newlines`とタイプすることにより、他のバッファーにたいしてこの機能を有効または無効にできます。

ハード改行は、パラグラフを分割したりテキストがフィルされる方法に関らず行区切りが必要な場所で使用され、ソフト改行はフィルで使用されます。`RET` (`newline`) および `C-o` (`open-line`) コマンドはハード改行を挿入します。Auto Fill(Section 22.6.1 [Auto Fill], page 256 を参照してください) を含むフィルコマンドは、ソフト改行だけを挿入し、削除もソフト改行だけでハード改行は削除せずに残します。

したがって Enriched モードで編集するときは、フィルされたパラグラフの途中で行を分けるのに`RET`や`C-o`を使うべきではありません。かわりに Auto Fill モード、または明示的なフィルコマンド (Section 22.6.2 [Fill Commands], page 256 を参照してください) を使用します。テーブルやリストのように、行区切りを常に残したい場所で`RET`や`C-o`を使用します。そのような行では、行端揃えスタイル (justification style) も`unfilled`にしたいと思うかもしれません (Section 22.14.6 [Enriched Justification], page 278 を参照してください)。

22.14.3 フォーマット情報の編集

プロパティを変更する一番簡単な方法は、'`Text Properties`'メニューです。このメニューは、メニューバー (Section 1.4 [Menu Bar], page 10 を参照してください) の '`Edit`'メニュー、または`C-mouse-2`(Section 18.4 [Menu Mouse Clicks], page 198 を参照してください) で取得することができます。'`Text Properties`'メニューのコマンドのいくつかを、以下にリストします (`M-x`) で呼び出すこともできます):

`Remove Face Properties`
: リージョンからフェイスプロパティを削除します (`facemenu-remove-face-props`)。

`Remove Text Properties`
: リージョンからフェイスプロパティを含む、すべてのテキストプロパティを削除します (`facemenu-remove-all`)。

`Describe Properties`
: ポイントの後ろにある文字の、すべてのテキストプロパティと他の情報をリストします (`describe-text-properties`)。

`Display Faces`
: 定義されたフェイスのリストを表示します (`list-faces-display`)。Section 11.8 [Faces], page 82 を参照してください。

`Display Colors`
: 定義されたカラーのリストを表示します (`list-colors-display`)。Section 11.9 [Colors], page 83 を参照してください。

他のメニューエントリーについては、以下のセクションで説明します。

22.14.4 Enriched テキストのフェイス

以下のコマンドは、フェイスを追加または削除するのに使用することができます (Section 11.8 [Faces], page 82 を参照してください)。マークがアクティブのときは、リージョンのテキストに適用され、マークが非アクティブのときは、次の自己挿入文字に適用されます。プレフィクス引数を指定すると、これらのコマンドはリージョンがアクティブな場合でも、次の自己挿入文字に適用されます。

`M-o d`
: すべての`face`プロパティを削除します (`facemenu-set-default`)。

`M-o b` `bold`(太字) フェイスを適用します (`facemenu-set-bold`)。

`M-o i` `italic`(斜体) フェイスを適用します (`facemenu-set-italic`)。

`M-o l` `bold-italic`(太字斜体) フェイスを適用します (`facemenu-set-bold-italic`)。

`M-o u` `underline`(下線) フェイスを適用します (`facemenu-set-underline`)。

`M-o o` *face* `RET`
フェイス *face*を適用します (`facemenu-set-face`)。

`M-x facemenu-set-foreground`
カラーの入力を求め (Section 11.9 [Colors], page 83 を参照)、それをフォアグラウドカラーに適用します。

`M-x facemenu-set-background`
カラーの入力を求め、それをバックグラウンドカラーに適用します。

これらのコマンドは、Text Properties メニューからも利用可能です。

自己挿入文字は通常、フェイスプロパティ(および他のほとんどのテキストプロパティ) を、そのバッファーの前の文字から継承します。次の自己挿入文字にたいして上記のコマンドを指定した場合、その文字は前の文字からフェイスプロパティは継承しませんが、他のテキストプロパティは継承します。

Enriched モードは、他の追加のフェイス `excerpt`および `fixed`を定義します。これらは text/enriched ファイルフォーマットで使われるコードに対応します。`excerpt`フェイスは引用のために使用されることを意図されており、デフォルトでは `italic`と同じように表示されます。`fixed` フェイスは固定幅テキストを指定し、デフォルトでは `bold`と同じように表示されます。

22.14.5 Enriched テキストのインデント

Enriched モードでは、パラグラフまたはパラグラフの一部の、右余白または左余白に異なる量のインデントを指定できます。これらの余白は `M-q` (Section 22.6 [Filling], page 256 を参照してください) のようなフィルコマンドにも影響します。

Indentation サブメニューの Text プロパティは、インデントを指定するコマンドを提供します:

`Indent More`
リージョンを 4 列ずつインデントします (`increase-left-margin`)。Enriched モードでは、このコマンドは `C-x TAB`でも利用可能です。数引数を指定した場合、何列を余白に追加するかを指定します (負の引数は余白を何列縮めるかを指定します)。

`Indent Less`
リージョンから 4 列のインデントを削除します。

`Indent Right More`
右端から 4 列インデントすることにより、テキストを狭くします。

`Indent Right Less`
右端からのインデントを 4 列削除します。

変数 `standard-indent`は、これらのコマンドがインデントを追加または減少させる列数を指定します。デフォルトは 4 です。Enriched にたいするデフォルトの右余白は、通常のように `fill-column` により制御されます。

`C-c [` (`set-left-margin`) または `C-c ]` (`set-right-margin`) とタイプしても、左余白または右余白をセットできます。数引数で余白の幅を指定できます。指定しない場合、これらのコマンドはミニバッファーを通じて値の入力を求めます。

それに加えてフィルプレフィクスがある場合は、指定されたパラグラフのインデントに使用されます。C-x .は、フィルプレフィクスにたいして指定された、新しい値の中の空白文字は含めません。フィルコマンドは各行のインデントの後ろにフィルプレフィクスを探します。Section 22.6.3 [Fill Prefix], page 258 を参照してください。

22.14.6 Enriched テキストの位置調整

Enriched モードでは、以下のコマンドを使って、フィルのためにさまざまな行端揃えスタイル *(justification styles)* を指定できます。これらのコマンドは。ポイントを含むパラグラフ、またはリージョンがアクティブの場合は、リージョンと重なるすべてのパラグラフに適用されます。

`M-j l` 行を左余白に揃えます (`set-justification-left`)。

`M-j r` 行を右余白に揃えます (`set-justification-right`)。

`M-j b` 行の途中にスペースを挿入することにより、行を両端の余白に揃えます (`set-justification-full`)。

`M-j c`
`M-S` 両端の余白の間で行を中央に揃えます (`set-justification-center`)。

`M-j u` フィルを完全にオフに切り替えます (`set-justification-none`)。このセッティングにより、フィルコマンドはテキストに何も行わなくなります。それでも左余白はインデントすることができます。

Text Properties メニューの Justification サブメニューを使用して、行端揃えスタイルを指定することもできます。デフォルトの行端揃えスタイルはバッファーごとの変数 `default-justification` により指定されます。この変数の値はシンボル `left`、`right`、`full`、`center`、または `none`のうちの 1 つです。シンボルの意味は上述したコマンドに対応します。

22.14.7 その他のテキストプロパティのセッティング

Text Properties メニューの Special Properties サブメニューには、4 つのテキストプロパティ — `read-only` (テキストの変更を不可にします)、`invisible` (テキストを非表示にします)、`intangible` (テキスト内でのポイントの移動を不可にします)、`charset` (これは文字の表示に正しいフォントを選択するために重要です) — を追加または削除するエントリーがあります。'`Remove Special`'メニューアイテムは、リージョン内のテキストにたいする、これらの特別なプロパティを削除します。

プロパティ`invisible`および `intangible`は保存されません。

Enriched モードは`display`プロパティ(Section "Display Property" in *the Emacs Lisp Reference Manual* を参照) の保存とリストアもサポートします。このプロパティはスクリーン上でテキストがどのように表示されるかに影響を与え、バッファーのテキスト以外のソース由来のイメージや文字列も表示することができます。`display`プロパティはディスプレイにたいするプロパティ処理の一部として、任意の Lisp フォームの実行もサポートします。したがって、表示時しか知ることができないコンディションにディスプレイを動的に合わせる能力を提供します。任意の Lisp 実行は、そのenriched テキストのソースが Emacs、あるいはシステム外からのもの (たとえば受信した電子メールの添付ファイル) である場合は、Emacs に攻撃の可能性を開くので、Enriched モードではそのような実行はデフォルトで無効になっています。変数 `enriched-allow-eval-in-display-props`を非`nil`値にカスタマイズすることにより、これを有効にすることができます。

22.15 テキストベーステーブルの編集

tableパッケージは、テキストベースのテーブルを簡単に編集するコマンドを提供します。以下は、そのようなテーブルがどのように見えるかの例です:

```
+-----------------+--------------------------------+-----------------+
|     Command     |          Description           |   Key Binding   |
+-----------------+--------------------------------+-----------------+
|  forward-char   |Move point right N characters   |       C-f       |
|                 |(left if N is negative).        |                 |
|                 |                                |                 |
+-----------------+--------------------------------+-----------------+
|  backward-char  |Move point left N characters    |       C-b       |
|                 |(right if N is negative).       |                 |
|                 |                                |                 |
+-----------------+--------------------------------+-----------------+
```

このようなテキストがテーブルだと Emacs が認識した場合 (Section 22.15.3 [Table Recognition], page 280 を参照してください)、テーブルのセル内容を編集すると、内容が大きくなってセルに収まらなくなったときは、テーブルのセルを自動的にサイズ変更します。以下のセクションで説明されている、テーブルのレイアウトを操作したり編集するコマンドを使用することができます。

M-x table-fixed-width-modeとタイプすることにより、自動的なテーブルのサイズ変更を切り替えることができます。

22.15.1 テキストベーステーブルとは?

テーブル *(table)* は矩形のテキスト領域からなり、それらはセル *(cells)* に分割されます。セルは少なくとも 1 文字分の幅と高さをもち、それにボーダーラインは含まれません。セルは複数のセルに分割できますが、それらは重なることはできません。

セルのボーダーラインは、以下の変数で指定された 3 つの特別な文字で描画されます:

`table-cell-vertical-char`
: 垂直ラインに使用される文字です。デフォルトは '|'です。

`table-cell-horizontal-chars`
: 水平ラインに使用される文字です。デフォルトは '"-="'です。

`table-cell-intersection-char`
: 水平ラインと垂直ラインの交点に使用される文字です。デフォルトは '+'です。

以下は無効なテーブルの例です:

```
+-----+       +--+    +-++--+
|     |       |  |    | ||  |
|     |       |  |    | ||  |
+--+  |    +--+--+    +-++--+
|  |  |    |  |  |    +-++--+
|  |  |    |  |  |    | ||  |
+--+--+    +--+--+    +-++--+
   a          b          c
```

左から順に説明します:

a. 重なったセル、または非矩形のセルは許されません。

b. ボーダーは矩形でなければなりません。

c. セルは最小でも 1 文字分の幅と高さがなければなりません。

22.15.2 テーブルの作成

テキストベースのテーブルをスクラッチから作成するには、M-x table-insertとタイプします。このコマンドはテーブルの列数、テーブルの行数、セルの幅と高さの入力を求めます。セルの幅とセルの高さに、セルのボーダーは含まれません。これらは 1 つの整数で指定するか (これは各セルの幅と高さが同じになることを意味します)、スペースまたはカンマで区切られた整数のシーケンス (これはテーブルの左から右の列、上から下の行の個別のセルにたいして幅と高さを指定します) です。その後、指定されたテーブルはポイント位置に挿入されます。

M-x table-insertで挿入されたテーブルは、Emacs にそれをテキストベースのテーブルとして特別に扱うように指示する、特別なテキストプロパティを含みます。そのバッファーをファイルに保存して、後で再び visit すると、それらのプロパティは失われ、Emacs にとってテーブルは普通のテキストとして判断されます。これをテーブルに変換する方法については、次のセクションを参照してください。

22.15.3 テーブルの認識

バッファーに既存のテーブルがあり、そのテーブルから M-x table-insertで適用された特別なテキストプロパティが失われているとき、それはテーブルとして特別に扱われなくなります。これにテキストプロパティを適用するには、M-x table-recognizeとタイプします。このコマンドはカレントバッファーをスキャンして、有効なテーブルセルを認識して、適切なテキストプロパティを適用します。反対に、M-x table-unrecognizeとタイプすることにより、特別なテキストプロパティを削除して、テーブルをプレーンテキストに変換して、カレントバッファーのすべてのテーブルを認識しなくなります。

以下のコマンドを使って、選択的にテーブルを認識したり認識させなくすることもできます:

M-x table-recognize-region
: カレントリージョンのテーブルを認識します。

M-x table-unrecognize-region
: カレントリージョンのテーブルを認識しなくします。

M-x table-recognize-table
: ポイント位置のテーブルを認識してアクティブにします。

M-x table-unrecognize-table
: ポイント位置のテーブルを非アクティブにします。

M-x table-recognize-cell
: ポイント位置のセルを認識してアクティブにします。

M-x table-unrecognize-cell
: ポイント位置のセルを非アクティブにします。

テーブルを認識する他の方法については、Section 22.15.7 [Table Conversion], page 282 を参照してください。

22.15.4 テーブルセルにたいするコマンド

コマンド M-x table-forward-cellおよび M-x table-backward-cellは、ポイントをカレントセルから隣接するセルに移動します。その順番は巡回的です。ポイントがテーブルの最後のセルにあるとき、M-x table-forward-cellはポイントを最初のセルに移動します。同様に、ポイントが最初のセルにあるとき、M-x table-backward-cellはポイントを最後のセルに移動します。

M-x table-span-cellは方向 — 右、左、上、下 — の入力を求め、カレントセルをその方向の隣接するセルにマージします。マージした結果が不正なセルのレイアウトになる場合、このコマンドはエラーをシグナルします。

M-x table-split-cellは、ミニバッファーを使って分割する方向の入力を求め、カレントセルを垂直 (vertically) または水平 (horizontally) に分割します。特定の方向に分割するには、M-x table-split-cell-verticallyと M-x table-split-cell-horizontallyを使います。垂直に分割する場合、古いセルの内容は自動的に 2 つの新しいセルに分割されます。水平に分割する場合、セルが空でない場合は、セル内容をどのように分割するか入力を求めます。オプションは‘split’(内容をポイント位置で分割)、‘left’(すべての内容を左のセルへ)、‘right’(すべての内容を右のセルへ)です。

以下のコマンドは、セルを拡大または縮小します。デフォルトでは、1 行または 1 列ずつサイズ変更します。数引数が与えられた場合、それはサイズ変更を何行または何列単位で行うかを指定します。

M-x table-heighten-cell
: カレントセルを垂直方向に拡大します。

M-x table-shorten-cell
: カレントセルを垂直方向に縮小します。

M-x table-widen-cell
: カレントセルを水平方向に拡大します。

M-x table-narrow-cell
: カレントセルを水平方向に縮小します。

22.15.5 セルの位置調整

コマンド M-x table-justifyは、テキストベーステーブルの 1 つ以上のセルにたいして、位置調整 *(justification)* を行います。位置調整は、セルのテキストがセルの外枠にたいして、どのように位置合わせされるかを決定します。テーブルの各セルは個別に位置調整できます。

M-x table-justifyは、最初に何の位置調整をするかの入力を求めます。オプションは‘cell’(カレントセルのみ)、‘column’(テーブルのカレント列のすべてのセル)、‘row’ (テーブルのカレント行のすべてのセル) です。その後、コマンドは位置調整のスタイルの入力を求めます。オプションは left、center、right、top、middle、bottom、または none(垂直方向の位置調整を行わないことを意味します) です。

水平および垂直方向の位置調整スタイルは独立して指定され、2 つのタイプは同時に適用できます。たとえば M-x table-justifyを 2 回呼び出して、1 回目は位置調整に rightを指定して、2 回目は位置調整に bottomを指定することにより、セルの内容を右下に位置調整することができます。

位置調整スタイルは、テキストプロパティとしてバッファーに格納され、バッファーを kill するか Emacs を終了すると失われます。M-x table-recognize(Section 22.15.3 [Table Recognition], page 280 を参照してください) のようなテーブルの認識コマンドは、セルの内容を確認して各セルの位置調整スタイルの決定と再適用を試みます。この機能を無効にするには、変数 table-detect-cell-alignmentを nilに変更してください。

22.15.6 テーブルの行と列

M-x table-insert-rowは、テーブルのカレント行の前にセル行を挿入します。カレント行とポイントは、新しい行の下になります。テーブル最下にある最後の行の後ろに行を挿入するには、ポイントをテーブルの最下線の直下にポイントを移動して、このコマンドを呼び出します。数引数を指定することにより 2 つ以上の行を挿入できます。

同様に、M-x table-insert-columnは、テーブルのカレント列の左にセル列を挿入します。右端の最右列の右に列を挿入するには、ポイントを最右線の右 (テーブルの外) に移動して、このコマンドを呼び出します。数引数は挿入する列の数を指定します。

M-x table-delete-columnは、ポイントがあるセルの列を削除します。同様に、M-x table-delete-rowは、ポイントがあるセルの行を削除します。どちらのコマンドも数引数により削除する列または行の数を指定します。

22.15.7 プレーンテキストとテーブルの変換

コマンドM-x table-captureは、プレーンテキストをリージョンにキャプチャーして、それをテーブルに変換します。M-x table-recognize(Section 22.15.3 [Table Recognition], page 280 を参照してください) とは異なり、元のテキストはテーブルの体裁をとる必要はありません。テーブル的な論理構造もつことだけが必要です。

たとえば以下の数字があったとして、これらは 3 つの行と、カンマで水平方向に分割されます:

```
1, 2, 3, 4
5, 6, 7, 8
, 9, 10
```

このテキストにたいしてM-x table-captureを呼び出すと、以下のテーブルが生成されます:

```
+-----+-----+-----+-----+
|1    |2    |3    |4    |
+-----+-----+-----+-----+
|5    |6    |7    |8    |
+-----+-----+-----+-----+
|     |9    |10   |     |
+-----+-----+-----+-----+
```

M-x table-releaseは逆のことを行います。これはテーブルを元のプレーンテキストに戻し、セルのボーダーを削除します。

この 2 つのコマンドの 1 つの用途としては、テキストをレイアウトで編集することです。以下の 3 つのパラグラフを見てください:

```
table-capture is a powerful command.
Here are some things it can do:

Parse Cell Items   Using row and column delimiter regexps,
                   it parses the specified text area and
                   extracts cell items into a table.
```

上記のテキストを含むリージョンにtable-captureを適用して、列と行を区切る regexps に空文字列を指定すると、以下のような 1 つのセルからなるテーブルが作成されます。

```
+-------------------------------------------------------+
|table-capture is a powerful command.                   |
|Here are some things it can do:                        |
|                                                       |
|Parse Cell Items   Using row and column delimiter regexps,|
|                   it parses the specified text area and  |
|                   extracts cell items into a table.   |
+-------------------------------------------------------+
```

その後でセル分割コマンド (Section 22.15.4 [Cell Commands], page 280 を参照してください) を使って、各パラグラフが 1 つのセルを占めるようにテーブルを分割できます。

```
+-------------------------------------------------------+
```

```
|table-capture is a powerful command.                    |
|Here are some things it can do:                         |
+-----------------+--------------------------------------+
|Parse Cell Items | Using row and column delimiter regexps,|
|                 | it parses the specified text area and  |
|                 | extracts cell items into a table.      |
+-----------------+--------------------------------------+
```

これで各セルは他のセルのレイアウトに影響を与えることなく、独立して編集することができるようになりました。終了したら `M-x table-release`を呼び出して、テーブルをプレーンテキストに戻します。

22.15.8 テーブル、その他

コマンド `table-query-dimension`は、テーブルのレイアウトとポイント位置のテーブルセルをレポートします。以下は出力の例です:

```
Cell: (21w, 6h), Table: (67w, 16h), Dim: (2c, 3r), Total Cells: 5
```

これはカレントセルの幅が 21 文字、高さが 6 行で、テーブルの幅が 67 文字、高さが 16 行で、2 列 3 行で 5 つのセルがあることを示します。

`M-x table-insert-sequence`は、テキスト文字列のシーケンスを、テーブルのセルを横断して、適切に各セルに挿入します。これは、そのシーケンスの基本文字列 (base string) を尋ねて、その基本文字列を数値的 (基本文字列が数字で終わる場合)、または ASCII順で “増加” させてシーケンスを生成します。基本文字列に加え、このコマンドはシーケンス内の要素数、増分、セル間隔、各セル内のテキストの行端揃えの入力を求めます。

`M-x table-generate-source`は特定のマークアップ言語にフォーマットされたテーブルを生成します。このコマンドは言語 (これは `html`、`latex`、`cals`、`wiki`、`mediawiki`のいずれかでなければなりません) と、結果を出力するバッファー、テーブルの表題を尋ね、生成されたテーブルを指定したバッファーに出力します。デフォルトの出力バッファーは `table.`*`lang`*で、*lang*は指定された言語です。

22.16 2 列編集

2C-two-column(2 列) モードは、横に並んだテキストの列を、簡単に編集できるようにします。これは横に並んだ 2 つのウィンドウを使って、それぞれのウィンドウは自身のバッファーを表示します。2C モードに入るには 3 つの方法があります。

`F2 2`または `C-x 6 2`
: カレントバッファーを左に、カレントバッファーの名前にもとづいた名前のバッファーを右にして、2C モードに入ります。右側のバッファーがまだ存在しない場合、そのバッファーは空で開始されます。カレントバッファーの内容は変更されません。

  このコマンドは、カレントバッファーが空か 1 列だけしか含まなくて、それに別の列を追加したいときに適しています。

`F2 s`または `C-x 6 s`
: 2 列のテキストを含むカレントバッファーを 2 つのバッファーに分割して、それを横に並べて表示します (`2C-split`)。カレントバッファーは左側のバッファーになりますが、右側の列のテキストは、右側のバッファーに移動します。カレント列は分割ポイントを指定します。分割はカレント行からバッファーの最後まで続きます。

  このコマンドはすでに 2 列になったテキストを含むバッファーがあり、一時的にそれを列に分割したいときに適しています。

F2 b *buffer* RET
C-x 6 b *buffer* RET
: カレントバッファーを左側のバッファー、バッファー *buffer*を右側のバッファーにして 2C モードに入ります (2C-associate-buffer)。

F2 sまたは C-x 6 sは、列の区切りを探します。区切りは各行に出現する 2 列の間にある文字列です。区切りの幅は、F2 sへの数引数で指定することができます。これはポイントの前の指定した文字数文を区切り文字とします。デフォルトでは幅が 1 なので、ポイントの前の文字が列区切りになります。

行が適切な位置で分割された場合、F2 sは区切りの後ろのテキストを右側のバッファーに移動して、区切りを削除します。適切な位置に列区切りをもたない行は、分割されずに残ります。分割されない行は左側のバッファーに残り、右側のバッファーの対応する行は空になります (これは 2C モードで 2 列にまたがる行を書くときの方法で、そのような行は左側のバッファーに記述して、右側のバッファーは空の行にします)。

コマンド F2 RETまたは C-x 6 RET (2C-newline) は、2 つのバッファーの対応する位置にそれぞれ改行を挿入します。これは 2 列のテキストを分割されたバッファーで編集するときに、新しい行を追加する一番簡単な方法です。

望みどおりに両方のバッファーを編集し終えたら、F2 1または C-x 6 1 (2C-merge) で、それらをマージします。これは右側のバッファーからテキストを、もう一方のバッファーの 2 列目にコピーします。2 列編集に戻るには、F2 sを使用します。

F2 dまたは C-x 6 dは、バッファーをそのままにして、2 つのバッファーを分割します。F2 dとタイプしたとき、カレントではないバッファーが空の場合、F2 dによりそのバッファーは kill されます。

23 プログラムの編集

このチャプターでは、プログラムの編集を容易にする Emacs の機能を説明します。これらの機能で行えるいくつかは、以下のようなものです:

- トップレベルの定義の検索や移動 (Section 23.2 [Defuns], page 286 を参照してください)。
- 言語の通常のインデント規則の適用 (Section 23.3 [Program Indent], page 288 を参照してください)。
- カッコの対応をとります (Section 23.4 [Parentheses], page 291 を参照してください)。
- コメントの挿入、kill、位置揃え (Section 23.5 [Comments], page 295 を参照してください)。
- プログラム構文のハイライト (Section 11.13 [Font Lock], page 89 を参照してください)。

23.1 プログラミング言語のためのメジャーモード

Emacs には、プログラミング言語のために特化した、多くのメジャーモード (Section 20.1 [Major Modes], page 240 を参照してください) があります。プログラミング言語に関連したモードは通常、式の構文、インデントの慣習的ルール、言語の構文をハイライトする方法、関数定義の開始と終了を検索する方法を指定します。プログラムをコンパイルしたり、デバッグするための機能をもつ場合もあります。各言語にたいするメジャーモード名は、言語名により名づけられます。たとえば C プログラミング言語にたいするメジャーモードは、`c-mode`です。

Emacs には Lisp、Scheme、Scheme ベースの DSSSL expression 言語、Ada、ASM、AWK、C、C++、C#、Fortran、Icon、IDL(CORBA)、IDLWAVE、Java、Javascript、M4、Makefile、Metafont(フォント作成のための TEX の仲間)、Modula2、Object Pascal、Objective-C、Octave、Pascal、Perl、Pike、PostScript、Prolog、Python、Ruby、Simula、SQL、Tcl、TypeScript、Verilog、VHDL などのプログラミング言語のためのモードがあります。Perl のための代替モードは CPerl モードと呼ばれます。一般的な GNU および Unix シェルのスクリプティング言語、MS-DOS/MS-Windows の '`BAT`'ファイル、JSON、DNS master、CSS(Cascading Style Sheets)、Docker、CMake などのファイル、それに一連のさまざまな設定ファイルのためのモードも利用可能です。

理想的には、Emacs は編集したいと望むすべてのプログラミング言語のためのメジャーモードをもつべきです。しかし、もしあなたのお気に入りの言語のためのモードがない場合、それは Emacs とともに配布されないパッケージで実装されているかもしれません (Chapter 32 [Packages], page 490 を参照してください)。または、あたたが貢献することもできます。

Emacs が '`tree-sitter`'ライブラリーとともにコンパイルされていれば、このライブラリーにもとづく複数の編集モードが提供されます。これらは '`tree-sitter`'がもたらすインクリメンタルな解析機能を利用するモードです。これらのモードには `c-ts-mode`、`python-ts-mode`のように名前に '`-ts-`'が含まれています。

ほとんどのプログラミング言語では、インデントはプログラム構造を示すために行ごとに異なります。したがって、ほとんどのプログラミング言語のモードでは、`TAB`とタイプすることにより、カレント行のインデントが更新されます (Section 23.3 [Program Indent], page 288 を参照してください)。さらに `DEL`は通常、タブをあたかも等価な数のスペースであるかのように扱って、後方に削除する `backward-delete-char-untabify`にバインドされているので、空白文字がスペースなのかタブなのか気にせずに、インデントを 1 列ずつ削除できます。

プログラミング言語のモードに入ることにより、フック変数 `prog-mode-hook`に指定されたカスタム Lisp 関数と、その後でモード自身のモードフックが実行されます (Section 20.1 [Major Modes],

page 240 を参照してください)。たとえばC モードに入ることにより、prog-mode-hookと c-mode-hookが実行されます。フックについての情報は、Section 33.2.2 [Hooks], page 509 を参照してください。

Emacs ディストリビューションには Ada、C/C++/Objective C/Java/Corba IDL/Pike/AWK、Octave、VHDL、IDLWAVE のためのメジャーモードの Info マニュアルが含まれています。Fortran モードに関しては、Section “Fortran” in *Specialized Emacs Features* を参照してください。

23.2 トップレベルの定義、または defun

Emacs では関数などの、バッファーのトップレベルの主要な定義は、*defun* と呼ばれます。この名前は Lisp が由来ですが、Emacs ではすべての言語に使用します。

23.2.1 左端の慣習

プログラミング言語のモードの多くは、伝統的に左マージンにある開カッコや開大カッコ (opening parenthesis or brace) をトップレベルの定義や defun の開始とみなします。そのために defun 開始を探すコマンドは、デフォルトではそのような区切りを defun 開始位置として受け取ります。

この慣習をオーバーライドしたければ、ユーザーオプション open-paren-in-column-0-is-defun-startを nilにセットすることによって行うことができます。このオプションが t (デフォルト) にセットされている場合には、この defun 開始の探すコマンドは、コメントや文字列中以外の列 0 の開カッコか開大カッコ (braces) で停止します。nilなら defun は最外レベルのカッコまたは大カッコを検索することにより発見されます。Emacs の低レベルルーチンはもはやこの慣習に依存しないので、通常は open-paren-in-column-0-is-defun-startをデフォルトから変更する必要はないでしょう。

23.2.2 defun の移動

これらのコマンドはトップレベルの主要な定義、または *defuns* にもとづいてポイントを移動したり、リージョンをセットアップします。

C-M-a カレントまたは直前の defun の先頭に移動します (beginning-of-defun)。

C-M-e カレントまたは直後の defun の最後に移動します (end-of-defun)。

C-M-h カレントまたは後続の defun 全体の周囲にリージョンを設定します (mark-defun)。

カレントの defun の先頭または最後に移動するコマンドは、C-M-a (beginning-of-defun) と C-M-e (end-of-defun) です。これらのコマンドの 1 つを繰り返すか、正の数引数を使用すると、繰り返しごとに動作方向の次の defun に移動します。

C-M-aで負の引数 $-n$を指定すると、次の defun の開始へと n回移動します。これは C-M-eに引数 nを与えたときに移動する位置と、正確に同じではありません。defun の終わりは通常、後続の defun の開始と同じ位置ではないからです (空白文字、コメント、もしかしたら宣言がこれらの defun を分割するからです)。同様に C-M-eに負の引数を与えると、defun の最後に後方に移動しますが、これは C-M-aに正の引数を与えた場合とは完全に異なる位置になります。

カレントの defun を操作するには、C-M-h (mark-defun) を使用します。これはカレントの defun の最後にマークをセットし、先頭にポイントを配します。Section 8.2 [Marking Objects], page 54 を参照してください。これは、その defun を kill してファイルの他の場所に移動するための、一番簡単な準備方法です。defun の直前 (間に空行を挟まない) にコメントがある場合は、そのコメントもマークされます。ポイントが defun の間にある場合は、後続の defun を使用します。マークがすでにアク

ティブのときにこのコマンドを使用すると、リージョンの最後が、複数の defun を含むように拡張されます。プレフィクス引数を指定した場合は、その数の defun をマークするか、適切な数の defun になるようリージョンを拡張します。負のプレフィクス引数の場合は、反対方向の defun をマークするとともに、以降の `mark-defun`使用での選択方向も変更します。

C モードでは、`mark-defun`とほとんど同じな `c-mark-function`を実行します。違いは、それが引数定義、関数名、リターンデータ型を含むことで、これにより C 関数全体にリージョンが設定されます。これは標準のキーバインディングをメジャーモードが調整する方法の例です。これにより特定の言語によりふさわしい方法で標準的な作業を行うのです。この目的のために、他のメジャーモードは、これらのキーバインディングすべてを置き換えているかもしれません。

一部のプログラミング言語ではネストされた *defun(nested defuns:* 入れ子関数*)* がサポートされています。これは他の defun の内部 (body、すなわち本体の一部として) で定義された defun のことです (defun とは関数、メソッド、クラスのような類い)。上述した関数はデフォルトではポイント近傍でもっとも内側の defun の先頭か終端を探して見つけられます。tree-sitter ライブラリーにもとづくメジャーモードでは、この挙動にたいする制御が提供されます。変数 `treesit-defun-tactic`の値に `top-level`がセットされていれば、defun コマンドは内側ではなくもっとも外側の defun を探すようになります。

23.2.3 Imenu とは

Imenu 機能は、ファイル内の主要な定義を、名前で検索する方法を提供します。これはチャプター、セクションなどを扱う、テキストフォーマッターのためのメジャーモードでも有用です (複数ファイルを扱う、より強力な機能については、Section 25.4 [Xref], page 357 を参照してください)。

`M-g i` (`imenu`) とタイプすると、これはミニバッファーを使用して定義の名前を読み取りポイントをその定義に移動します。名前の指定に補完を使用できます。このコマンドは常に有効な名前の全体のリストを表示します。

かわりに、コマンド `imenu`をマウスクリックにバインドできます。そうすると定義名を選択するために、マウスメニューが表示されます。`imenu-add-menubar-index`を呼び出して、バッファーのインデックスをメニューバーに追加することもできます。特定のメジャーモードのすべてのバッファーにたいして、このメニューバーアイテムを有効にしたい場合、モードフックに `imenu-add-menubar-index`を追加して、これを行うことができます。しかし、それを行うことにより、そのモードでファイルを visit するたびに、Emacs がそのバッファーのすべての定義を検索する間、待つ必要があります。

バッファーの内容を変更する際、定義の追加や削除をした場合は、メニューの '`*Rescan*`'アイテムを呼び出すことにより、新しい内容にもとづいてバッファーのインデックスを更新することができます。`imenu-auto-rescan`を非 `nil`値にセットした場合、再スキャンは自動的に発生します。テキストを少量変更しただけの場合、再スキャンは必要ありません。

`imenu-auto-rescan`はバイト数が `imenu-auto-rescan-maxout`より大きいバッファーでは無効になり、スキャンが `imenu-max-index-time`秒を超えるようなら停止されるでしょう。

変数 `imenu-sort-function`をセットして、メニューがソートされる方法をカスタマイズすることができます。デフォルトでは、名前はバッファーで出現する順にソートされています。アルファベット順にソートしたいときは、値にシンボル `imenu--sort-by-name`を使用します。Lisp コードを記述することにより、独自の比較関数を定義することもできます。

カレントバッファーのプロジェクト (Section 25.2 [Projects], page 352 を参照) とメジャーモードにたいして Eglot がアクティブであれば、Eglot はバッファーのインデックスを生成するための独自機能を提供します。これはカレントバッファーを管理している言語サーバー (language-server) によるプログラムのソース解析にもとづいた機能です。Section “Eglot Features” in *Eglot: The Emacs LSP Client* を参照してください。

Imenu は Which Function モードに情報を提供します (以下参照)。Speedbar もこれを使用します (Section 18.9 [Speedbar], page 204 を参照してください)。

23.2.4 Which Function モード

Which Function(どの関数) モードは、グローバルなマイナーモード (Section 20.2 [Minor Modes], page 241 を参照してください) で、これはカレントの関数名をモードラインに表示して、バッファーの移動にしたがってそれを更新します。

Which Function モードを有効または無効にするには、コマンド `M-x which-function-mode`を使用します。Which Function モードは、グローバルなマイナーモードです。デフォルトでは、それをサポートする方法を知っている、すべてのメジャーモード (たとえば Imenu をサポートするすべてのメジャーモード) に影響を与えます。変数 `which-func-modes`の値を `t`(これは利用可能なすべてのメジャーモードをサポートすることを意味します) から、特定のメジャーモードのリストに変更することにより、これを制限することができます。

23.3 プログラムのインデント

プログラムを正しくインデントされた状態に保つには、何か変更したら Emacs を使って再インデントするのが一番よい方法です。Emacs には 1 行、指定した数の行、1 つのカッコでグループ化されたすべての行をインデントするコマンドがあります。

インデントに間する全般的な情報は、Chapter 21 [Indentation], page 247 を参照してください。このセクションでは、プログラミング言語のモードに特有のインデント機能について説明します。

Emacs は `pp`パッケージで、Lisp のプリティープリンター (pretty-printer) も提供します。これは Lisp オブジェクトを、見栄えのよいインデントで再フォーマットします。Section “Output Functions” in *The Emacs Lisp Reference Manual* を参照してください。

23.3.1 プログラムの基本的なインデントコマンド

`TAB` カレント行のインデントを調整します (`indent-for-tab-command`)。

`RET` 改行を挿入して、次の行のインデントを調整します (`newline`)。

基本的なインデントコマンドは `TAB` (`indent-for-tab-command`) で、これは Chapter 21 [Indentation], page 247 で説明されています。プログラミング言語のモードでは、`TAB`は前の行のインデントと構文的な内容にもとづき、カレント行をインデントします。リージョンがアクティブのとき、`TAB`はカレント行だけでなく、リージョン内の各行をインデントします。

Section 4.1 [Inserting Text], page 17 で説明されているコマンド `RET` (`newline`) は、`C-j`の後に `TAB`を続けるのと同じです。これは改行を挿入してから、その行のインデントを調整します。

カッコによるグループ化により開始される行でインデントする場合、通常 Emacs はグループ内の前の行の開始、または、カッコの後ろのテキストの下に、行の開始を合わせます。(たとえば美的観点により) これらの行に手動で非標準的なインデントを与えた場合は、その下の行もそれにしたがいます。

プログラミング言語のモードのほとんどは、左端の開きカッコ、開き大カッコ (open-brace)、その他の開始区切りを関数の開始とみなします。編集しているコードがこの前提に違反する場合は— それらの区切りが文字列やコメントの中にある場合でも — インデントが正しく機能するために、`open-paren-in-column-0-is-defun-start`に `nil`をセットしなければなりません。Section 23.2.1 [Left Margin Paren], page 286 を参照してください

23.3.2 複数行のインデント

複数行のコードを、1 度に再インデントしたいときがあるかもしれません。これを行う 1 つの方法は、マークを使う方法です。マークがアクティブでリージョンが空でないとき、TABはリージョン内の各行をインデントします。一方、コマンド C-M-\ (indent-region) は、マークがアクティブか否かにかかわらず、リージョン内の各行をインデントします (Section 21.1 [Indentation Commands], page 247 を参照してください)。

それらに加えて、Emacs はコードの大きな断片をインデントするために、以下のコマンドを提供します:

C-M-q　カッコでグループ化された、すべての行を再インデントします。

C-u TAB　カッコでグループ化されたすべてを横にシフトして、最初の行が正しくインデントされるようにします。

M-x indent-code-rigidly
　リージョン内のすべての行を横にシフトしますが、コメントや文字列内で開始される行は変更しません。

1 つのカッコでグループ化されたものを再インデントするには、ポイントをグループ化の前に配して、C-M-qとタイプします。これはグループ化されたものの全体的なインデント (たとえばグループ化が開始される行のインデント) は変更せずに、相対的なインデントを変更します。C-M-qで実行される関数は、Lisp モードでは indent-pp-sexp、C モードでは c-indent-exp、のようにメジャーモードに依存します。全体的なインデントも同様に訂正したい場合は、最初に TABをタイプします。

グループ内の相対的なインデントは好ましいが、それの最初の行のインデントが好ましくない場合は、その最初の行にポイントを移動して C-u TABとタイプします。Lisp、C、他のいくつかのメジャーモードでは、数引数を指定した TABは通常どおりカレント行をインデントしてから、カレント行で始まる、カッコでグループ化されたすべての行を同じ量で再インデントします。このコマンドは巧妙なので、文字列内で開始される行は変更しません。C モードのときは C プリプロセッサー行は変更しませんが、それらにアタッチされた継続行は再インデントします。

コマンド M-x indent-code-rigidlyは、indent-rigidlyが行うように (Section 21.1 [Indentation Commands], page 247 を参照してください)、リージョン内のすべての行を厳格 (rigidly) にシフトします。このコマンドはリージョンが文字列内で開始される場合を除き、文字列内で開始される行のインデントは変更しません。プレフィクス引数は、インデントする列数です。

23.3.3 Lisp のインデントのカスタマイズ

Lisp 式のインデントパターンは、式により呼ばれる関数によって決定することができます。それぞれの Lisp 関数のために、事前定義された複数のインデントパターンの中から選択するか、Lisp プログラムで独自のものを定義できます。

標準的なインデントのパターンは以下のとおりです。最初の引数が式の開始行にある場合、式の 2 行目は最初の引数の下にインデントされます。そうでない場合、2 行目は関数名の下にインデントされ、後続の行はネストの深さが同じなら前の行の下にインデントされます。

変数 lisp-indent-offsetが非 nilの場合、これは式の 2 行目の通常のインデントパターンをオーバーライドするので、常にこれを含むリストより lisp-indent-offset列余計にインデントされます。

特定の関数は、標準のパターンをオーバーライドします。名前が defで始まる関数は、式の開始の開きカッコより lisp-body-indent列多く 2 行目をインデントすることにより、2 行目を *body* の開始として扱います。

関数名の `lisp-indent-function`プロパティにより、個々の関数の標準パターンを、さまざまな方法でオーバーライドできます。これは通常 `declare`構成を使用することにより、マクロ定義のために行われます。Section “Defining Macros” in *The Emacs Lisp Reference Manual* を参照してください。

Emacs Lisp でのリストは、通常はあたかも関数であるかのようにインデントされます:

```
(setq foo '(bar zot
                gazonk))
```

ただし開カッコの後にスペースを加えることにより、Emacs にそれがコード片ではなくデータリストであることが伝えられて、以下のようにインデントされます:

```
(setq foo '( bar zot
             gazonk))
```

23.3.4 C のインデントのためのコマンド

C および関連するモードのインデントのために、特別な機能があります。

`C-c C-q` カレントのトップレベル関数を再インデント、または型定義を統合します (CC モードでは `c-indent-defun`、tree-sitter ベースの `c-ts-mode`では `c-ts-mode-indent-defun`)。

`C-M-q` ポイントの後の、釣り合いの取れた式 (Section 23.4.1 [Expressions], page 292 を参照) の中の各行を再インデントします (`c-indent-exp`)。これは CC モードでは `c-indent-exp`、tree-sitter ベースの `c-ts-mode`ではより汎用的な `prog-indent-sexp`を呼び出します。プレフィックス引数を指定すると、無効な構文 (invalid syntax) に関する警告メッセージを抑止します。

`TAB` カレント行、アクティブなリージョン、その行は始点となるブロックを再インデントします (`c-indent-command`)。プレフィックス引数を指定すると、もしカレント行の再インデントが必要なら、カレント行を起点に釣り合いの取れた式を厳格に再インデントします

`c-tab-always-indent`が `t`の場合、このコマンドは常にカレント行をインデントし、他には何もしません。これがデフォルトです。

この変数が `nil`の場合、このコマンドはポイントが左端か、その行のインデント位置にある場合だけ、カレント行を再インデントします。そうでない場合、このコマンドはタブ (`indent-tabs-mode`が `nil`の場合は等価な数のスペース) を挿入します。

(`nil`でも `t`でもない) 他の値の場合、常にその行をインデントし、コメントか文字列の中の場合は、タブも挿入します。

カレントバッファー全体を再インデントするには、`C-x h C-M-\`とタイプします。これは最初にバッファー全体をリージョンとして選択し、それからリージョンを再インデントします。

カレントブロックを再インデントするには、`C-M-u C-M-q`を使用します。これはブロックの前に移動してから、ブロックのすべてを再インデントします。

23.3.5 C のインデントのカスタマイズ

C モードおよび関連するモードは、インデントをカスタマイズするために柔軟なメカニズムを使用します。C モードはソース行を 2 ステップでインデントします。最初のステップは、行の内容とコンテキストに応じて行の構文を分類します。次のステップで、構文コンストラクトで選択されたスタイ

ルにより、関連付けられたインデントのオフセットを決定して、これをアンカーとなる命令文 *(anchor statement)* のインデントに加えます。

`C-c . style RET`
: 事前定義されたスタイル *style* (CC モードでは `c-set-style`、tree-sitter ベースの `c-ts-mode`では `c-ts-mode-set-style`) を選択します。

スタイルとは、C モードおよび関連するモードで使用できるカスタマイズ用の名前つきコレクションです。完全な説明は、Section “Styles” in *The CC Mode Manual* を参照してください。Emacs には、`gnu`、`k&r`、`bsd`、`stroustrup`、`linux`、`python`、`java`、`whitesmith`、`ellemtel`、`awk`を含む、事前に定義されたスタイルがいくつか付属します。これらのスタイルのうちいくつかは、主に 1 つの言語を意図したものですが、他のスタイルはこれらのモードをサポートする任意の言語で使用できます。スタイルがどのように見えるかは、何らかのコードにたいしてスタイルを選択して、(たとえば関数定義の最初で `C-M-q`とタイプして) 再インデントしてみることです。

カレントバッファーのスタイルを選択するには、コマンド `C-c .` を使用します。引数として、スタイル名を指定します (大文字小文字に意味はありません)。このコマンドはカレントバッファーだけに影響し、将来のインデントコマンドの呼び出しだけに影響します。そのバッファーにすでにあるコードの再インデントはしません。バッファー全体を新しいスタイルで再インデントするには、`C-x h C-M-\` とタイプしてください。

CC モード使用時には変数 `c-default-style`をセットして、さまざまなメジャーモードにたいしてデフォルトのスタイルを指定できます。この値は、スタイル名 (文字列)、または各要素が 1 つのメジャーモードと、それに使用するインデントスタイルを指定する alist です。たとえば、

```
(setq c-default-style
      '((java-mode . "java")
        (awk-mode . "awk")
        (other . "gnu")))
```

これは Java モード、AWK モードにたいして明示的に選択して、その他の C-like なモードにたいしては、デフォルトの ‘`gnu`’スタイルを指定します。この変数は C-like なメジャーモードを選択したときに効果を表します。したがって Java モードにたいして新しいデフォルトスタイルを指定するには、既存の Java モードのバッファーで、`M-x java-mode`とタイプすることにより効果が表れます。

tree-sitter ベースの `c-ts-mode`を使用時には、変数 `c-ts-mode-indent-style`をカスタマイズすることによってデフォルトのインデントスタイルをセットできます。

`gnu`スタイルは、C にたいする GNU プロジェクトで推奨されるフォーマットを指定します。これがデフォルトで、私たちが使用を推奨するスタイルです。

既存のスタイルのオーバーライドや、独自のスタイルを定義する方法など、C および関連するモードにたいするインデントのカスタマイズに間する情報は、Section “Indentation Engine Basics” in *the CC Mode Manual*、および Section “Customizing Indentation” in *the CC Mode Manual* を参照してください。

スタイルを指定するかわりに、サンプルコードのバッファーで `M-x c-guess`とタイプすることにより、Emacs にスタイルを推測させることができます。その後、`M-x c-guess-install`として、推測されたスタイルを他のバッファーに適用できます。詳細については、Section “Guessing the Style” in *the CC Mode Manual* を参照してください。

23.4 カッコに付随する編集のためのコマンド

このセクションでは、プログラム内のカッコ構造を活用したり、それらを対応が取れた状態に保つためのコマンドと機能について説明します。

これらの機能を語るとき、用語 “カッコ (parenthesis)” には、大カッコ (braces)、角カッコ (brackets)、またはマッチするペアとして定義される区切りも含まれます。メジャーモードはどの区切りに意味があるかを、構文テーブル (syntax table) を通じて制御します (Section “Syntax Tables” in *The Emacs Lisp Reference Manual* を参照してください)。Lisp では丸カッコ (parentheses) だけが考慮され、C ではこれらのコマンドは大カッコや角カッコにも適用されます。

`M-x check-parens`を使用して、バッファー内の対応の取れていないカッコと、対応が取れていない文字列のクォートを検索することができます。

23.4.1 対応が取れたカッコの式

プログラミング言語のモードはそれぞれ、対応が取れた式 *(balanced expression)* にたいする、独自の定義をもちます。対応が取れた式は通常、個別のシンボル、数字、文字列定数、同様にマッチする区切りで囲まれたコードの断片を含みます。以下のコマンドは対応が取れた式にたいするものです (Emacs では、内部的にそのような式は *sexp* として参照されます [1])。

`C-M-f` 対応が取れた式を、前方に飛び越して移動します (`forward-sexp`)。

`C-M-b` 対応が取れた式を、後方に飛び越して移動します (`backward-sexp`)。

`C-M-k` 対応が取れた式を、前方に kill します (`kill-sexp`)。

`C-M-t` 式を入れ替えます (`transpose-sexps`)。

`C-M-@`
`C-M-SPC` 後続の式の後ろに、マークを配します (`mark-sexp`)。

対応が取れた式を前方に飛び越して移動するには、`C-M-f` (`forward-sexp`) を使用します。ポイントの後ろの最初の文字が開始区切り (たとえば C では ‘(’、‘[’、‘{’) の場合、このコマンドはそれにマッチする終了区切りまで移動します。文字がシンボル、文字列、数字で始まる場合、このコマンドはそれらを飛び越して移動します。

コマンド `C-M-b` (`backward-sexp`) は対応が取れた式を後方 — `C-M-f`と同様ですが逆向き— に飛び越して移動します。式の前にプレフィクス文字がある場合 (Lisp ではシングルクォート、バッククォート、カンマ)、このコマンドはそれらも同様に飛び越して後方に移動します。

`C-M-f`および `C-M-b`に数引数を与えると、指定した回数繰り返し操作をします。負の引数を与えると、反対の方向に移動します。ほとんどのモードでは、これらの 2 つのコマンドはコメントをあたかも空白文字のように通過します。これら `C-M-f`および `C-M-b`のキーは、文字単位で移動する `C-f`および `C-b`(Section 4.2 [Moving Point], page 18 を参照してください)、単語単位に移動する `M-f`および `M-b`(Section 22.1 [Words], page 251 を参照してください) に類似していることに注意してください。

対応が取れた式全体を kill するには、`C-M-k` (`kill-sexp`) とタイプします。これは `C-M-f`が飛び越すテキストを kill します。

`C-M-t` (`transpose-sexps`) は、前にある対応の取れた式と、次の対応の取れた式の位置をスイッチします。このコマンドは、文字を入れ替える `C-t`コマンド (Section 13.2 [Transpose], page 132 を参照してください) と類似しています。`C-M-t`への引数は繰り返し回数を意味し、前の式を、何個か先の式に移動します。負の引数は前にある対応が取れた式を後方に、それらの式の前に移動します。引数 0 は何もしないのではなく、ポイントの位置で終わる対応が取れた式、またはポイントの後ろの式と、マークの後ろの式を入れ替えます。

[1] 単語 “sexp” は Lisp で式を参照するのに使用されます。

リージョンにたいして動作するコマンドで対応が取れた式を操作するには、C-M-SPC (mark-sexp) とタイプします。これはC-M-fが移動する位置にマークをセットします。マークがアクティブの間、このコマンドを連続で呼び出すとマークが 1 つの式ごとにシフトしてリージョンが拡張します。正または負の引数はマークを前方または後方に指定した数の式分のマークを移動します。エイリアス C-M-@ は、C-M-SPCと等価です。これに関連したコマンドは、Section 8.2 [Marking Objects], page 54 を参照してください。

C のように挿入オペレーター (infix operators) を使用する言語では、与えられた位置で複数の可能な解釈があるので、すべての対応が取れた式を認識するのは不可能です。たとえば ‘foo + bar’は 1 つの C の式ですが、C モードはこれを 1 つの式とは扱いません。かわりに ‘foo’を 1 つの式、‘bar’をもう 1 つの式、そしてそれらの間にある ‘+’を句読点として認識します。しかしカッコがあれば C モードは ‘(foo + bar)’を 1 つの式として認識します。

23.4.2 カッコ構造の移動

以下のコマンドはカッコ (または、あなたが使用している言語で、そのような区切りとして使用される文字) で区切られてグループ化されたものを移動するコマンドです。これらのコマンドは、カッコを含んでいたとしても文字列とコメントを無視し、エスケープ文字でクォートされたカッコも無視します。これらのコマンドは主にプログラムの編集を意図していますが、カッコを含む任意のテキストの編集にも有用です。これらは内部的には “リストコマンド” として参照されます。なぜなら Lisp ではこれらのグループはリストだからです。

これらのコマンドは、開始ポイントが文字列やコメントの中でないと仮定します。これらのコマンドを文字列やコメントの中から呼び出したとき、結果は信頼できません。

C-M-n カッコでグループ化されたグループを飛び越えて、前方に移動します (forward-list)。

C-M-p カッコでグループ化されたグループを飛び越えて、後方に移動します (backward-list)。

C-M-u カッコによる構造を上に移動します (backward-up-list)。

C-M-d カッコによる構造を下に移動します (down-list)。

リスト (list) コマンドの C-M-n (forward-list) と、C-M-p (backward-list) は、グループ化されたカッコを前方または後方に、1 つ (または *n*個) 飛び越えて移動します。

C-M-nと C-M-pは、カッコによる構造において同じレベルに留まろうと試みます。1 レベル (または *n*レベル) 上に移動するには、C-M-u (backward-up-list) を使用します。C-M-uは対応が取れていない、前にある開始区切りへ、後方に移動します。正の引数は繰り返し回数を意味します。負の引数は移動の方向を逆転するので、このコマンドは 1 レベル以上を上に、前方に移動します。

カッコによる構造を下に移動するには、C-M-d (down-list) を使用します。Lisp モードでは ‘(’は開始区切りなので、これは ‘(’を検索するのとほとんど同じです。引数は何レベルしたに移動するかを指定します。

23.4.3 マッチするカッコ

Emacs にはカッコのマッチング *(parenthesis matching)* 機能がいくつかあります。これにより、どのカッコ (または他の区切り) がどのようにマッチするか簡単に見ることができます。

終了文字となる自己挿入文字をタイプすると、Emacs はそれがスクリーン上にあれば、マッチする開始区切りの位置を簡単に示します。スクリーン上にない場合、Emacs は開始区切りの近くにあるテキストをエコーエリアに表示します。どちらの方法でも、どのグループを終了したのか見分けることができます。開始区切りと終了区切りがマッチしない— ‘[x)’のような場合、エコーエリアに警告メッセージが表示されます。

3 つの変数が、マッチするカッコの表示を制御します:

- `blink-matching-paren`は、この機能をオンまたはオフに切り替えます。`nil`は無効にしますが、デフォルトの `t`は有効にします。`jump`にセットすると、マッチする開始区切りに数瞬カーソルを移動して指示します。`jump-offscreen`にセットすると、開始区切りが画面上にない場合でもカーソルをジャンプさせます。
- `blink-matching-delay`は、マッチする開始区切りを何秒表示するかを指定します。これには整数または浮動小数点数を指定します。デフォルトは 1 です。
- `blink-matching-paren-distance`には、マッチする開始区切りを後方に何文字検索するかを指定します。この文字数内にマッチが見つからない場合、Emacs は検索を中止して、何も表示しません。デフォルトは 102400 です。

Show Paren モードは、より強力な自動的なマッチングの類を提供するマイナーモードです。開始区切りの前、または終了区切りの後ろにポイントを移動すると、(相手としてマッチする) 区切り、およびオプションで区切り同士の間にあるテキストをハイライトします。Show Paren モードをグローバルに切り替えるには `M-x show-paren-mode`、カレントバッファーだけで切り替えるには `M-x show-paren-local-mode`とタイプします。

このモードはデフォルトでは編集を意図したすべてのバッファーでオンになっています。ただしデータ表示用のバッファーでは有効になっていません。これを制御するのがユーザーオプション `show-paren-predicate`です。

モードをカスタマイズするには `M-x customize-group RET paren-showing` とタイプしてください。このモードでの操作を制御するカスタマイズ可能なオプションには以下が含まれます:

- `show-paren-highlight-openparen`は、ポイントが開きカッコの直前にあるとき (つまりいずれにせよそこはカーソルによってマークされる)、そのカッコをハイライトするかどうかを制御します。デフォルトは非 `nil` (ハイライトする) です。
- `show-paren-style`は 2 つのカッコだけをハイライトするか、それともカッコの間のテキストもハイライトされるかどうかを制御します。有効なオプションは `parenthesis` (マッチするカッコをハイライト)、`expression` (カッコで括られた式全体をハイライト)、`mixed` (ウィンドウ内でカッコが表示されているときはマッチするカッコ、カッコが表示されていないときは式をハイライト) です。
- `show-paren-when-point-inside-paren`が非 `nil`の場合には、ポイントがカッコの内側にあるときもハイライトされます。デフォルトは `nil`です。
- `show-paren-when-point-in-periphery`が非 `nil`なら、行頭の空白文字にポイントがあり行の最初か最後の非空白文字の位置にカッコがある、または行末にポイントがあり行の最後の非空白文字の位置にカッコがある場合にもハイライトを行います。
- `show-paren-context-when-offscreen`が非 `nil`の場合にはポイントが終了デリミタ (delimiter: 区切り文字) にあり、かつ開始デリミタが画面上にないときはエコーエリアにコンテキストの一部を表示します。このコンテキストは通常は開始デリミタを含む行ですが、開始デリミタそれ自体が行である場合にはその前の非ブランク行がコンテキストに含められます。

グローバルなマイナーモードの Electric Pair モードは、マッチするカッコ (parentheses)、大カッコ (braces)、角カッコ (brackets)、... などの区切りを簡単に挿入する方法を提供します。開始区切りを挿入すると、マッチする終了区切りが自動的に挿入され、2 つの区切りの間にポイントが置かれます。反対に終了区切りを既存の区切りの先に挿入した場合、何の挿入もされず、その位置は単にスキップされます。リージョンがアクティブ (Chapter 8 [Mark], page 52 を参照) の場合、区切りの挿入はそのリージョンを操作します。リージョン内の文字はマッチする区切りのペアー内に括られ、ポイントはタイプした区切りの後に置かれます。

以下の変数は、Electric Pair モードの追加機能を制御するのに使用できます:

- `electric-pair-preserve-balance` — 非`nil`の場合、デフォルトのペアリングロジックは、開始区切りと終了区切りの数の釣り合いになります。
- `electric-pair-delete-adjacent-pairs` — 非`nil`の場合、隣接する区切りの間でのバックスペースにより、終了区切りも自動的に削除します。
- `electric-pair-open-newline-between-pairs` — 非`nil`の場合、2 つの隣接するペアの間での改行の挿入は、ポイントの後ろに自動的に追加の改行をオープンします。
- `electric-pair-skip-whitespace` — 非`nil`の場合、終了区切りのスキップを決定する前に、マイナーモードが空白文字を前方にスキップするようにします。

Electric Pair モードに切り替えるには、`M-x electric-pair-mode`とタイプします。1 つのバッファーにたいしてこのモードを切り替えるには、`M-x electric-pair-local-mode`を使用してください。

23.5 コメントの操作

コメントは、プログラミングにおいて重要なパートなので、Emacs はコメントの編集や挿入を行う特別なコマンドを提供します。Flyspell Prog モードによる、コメントのスペルチェックも行うことができます (Section 13.4 [Spelling], page 134 を参照してください)。

異なる種類のコメントのインデントにたいして、特別なルールをもつメジャーモードがいくつかあります。たとえば Lisp コードでは、2 つのセミコロンで始まるコメントは、それらの行がコードであるかのようにインデントされ、3 つのセミコロンで始まるコメントは左端に揃えてインデントされ、しばしば区分けの目的で使用されます。Emacs はこれらの慣習を理解します。たとえば、コメント行で `TAB`をタイプすると、そのコメントを適切な位置にインデントします。

```
;; 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.
```

23.5.1 コメントコマンド

以下のコマンドは、コメントにたいして処理を行います:

`M-;`
: カレント行にコメントを挿入、または位置揃えします。リージョンがアクティブのときは、かわりにリージョンをコメント化、または非コメント化します (`comment-dwim`)。

`C-x C-;`
: カレント行をコメント、または非コメントにします (`comment-line`)。リージョンがアクティブの場合は、かわりにリージョンをコメント、または非コメントにします。

`C-u M-;`
: カレント行のコメントを kill します (`comment-kill`)。

`C-x ;`
: コメント列をセットします (`comment-set-column`)。

`C-M-j`
`M-j`
: `RET`の後にコメントを挿入して位置揃えします (`default-indent-new-line`)。Section 23.5.2 [Multi-Line Comments], page 297 を参照してください。

`M-x comment-region`
`C-c C-c` (C-like なモードの場合)
: リージョンのすべての行にたいして、コメント区切りを追加します。

コメントを挿入または位置揃えするコマンドは、M-; (comment-dwim) です。単語 “dwim” は、“Do What I Mean(私が言ったとおりにしてください)” の頭文字をとったものです。このコマンドは、コメントに関係する多くの異なる作業に使用できます。それは、このコマンドをどこで使うかという、状況に依存することを示します。

リージョンがアクティブ (Chapter 8 [Mark], page 52 を参照してください) の場合、M-;はリージョンのコメント区切りの追加と削除のどちらかを行います。リージョンの各行がすでにコメントの場合、それらのコメント区切りを削除することにより、それらの行を非コメント化します。そうでない場合は、リージョンのテキストにコメント区切りを追加します。

リージョンがアクティブのときに M-;に数引数を与えると、それは追加または削除するコメント区切りの数を指定します。正の引数 nは n個の区切りを追加し、負の引数 $-n$は n個の区切りを削除します。

リージョンが非アクティブで、カレント行にコメントがない場合、M-;はカレント行に新しいコメントを追加します。ブランク行 (たとえば空または空白文字しか含まない) の場合、TABをタイプしたときにインデントされるのと同じ位置に、コメントがインデントされます (Section 23.3.1 [Basic Indent], page 288 を参照)。非ブランク行の場合、コメントはその行の最後の非空白文字の後ろに配されます。Emacs は、可能なら変数 comment-columnと comment-fill-column (Section 23.5.3 [Options for Comments], page 297 を参照) で指定された列の間にコメントを配そうと試みます。それ以外では、Emacs は別の適切な位置、通常、非コメントのテキストと少なくとも 1 つのスペースを空けてコメントを配します。どちらの場合も、Emacs はコメントの開始区切りの後にポイントを配すので、すぐにコメントのタイプを開始できます。

既存のコメントの位置揃えにも M-;を使用できます。行がすでにコメント開始文字列を含む場合、M-;は慣習的な位置にそれを位置揃えして、ポイントをコメント開始区切りの後ろに移動します。例外として、列 0 で始まるコメントは移動されません。既存のコメントがすでに正しく位置揃えされているときでも、M-;はコメントテキストの開始に直接移動するので有用です。

C-x C-; (comment-line) は、行そのものをコメント化または非コメント化します。リージョンがアクティブ (Chapter 8 [Mark], page 52 を参照してください) の場合は、リージョン内の行をコメント化または非コメント化します。リージョンが非アクティブの場合、このコマンドはポイントがある行をコメント化または非コメント化します。正のプレフィクス引数 nを与えた場合、カレント行から数えて n行を処理します。負の引数 $-n$の場合は、先行する n行に作用します。負の引数を与えてこのコマンドを呼び出した後は、それに続けて正の引数を指定して連続して呼び出した場合、あたかも負の引数が与えられたかのように、先行する行を処理します。

C-u M-; (comment-dwimにプレフィクス引数を指定) は、リージョンが非アクティブなら、カレント行のすべてのコメントとその前の空白文字とを緒に kill します。コメントは kill リングに保存されるので、他の行の行末に移動して C-yで挿入して、M-;でコメントの位置揃えをすることができます。M-x comment-killとタイプしても C-u M-;と同じ効果を得ることができます (comment-dwimはプレフィクス引数を与えられたとき、実際にサブルーチンとして comment-killを呼び出す)。C-u n M-; のようにプレフィックス数引数とともに comment-dwimを呼び出すと、アクティブなリージョンがなければ n行のコメントの kill を指示したことになります。

コマンド M-x comment-regionは、アクティブなリージョンにたいして M-;を呼び出すのと等価ですが、このコマンドはマークが非アクティブのときでも、常にリージョンにたいして動作します。C モードおよび関連するモードでは、このコマンドは C-c C-cにバインドされています。コマンド M-x uncomment-regionは、リージョンの各行を非コメント化します。数引数は、削除するコメント区切りの数を指定します (負の引数は、追加するコメント区切りの数を指定します)。

C-like なモードにたいしては、変数 c-indent-comment-alistおよび c-indent-comments-syntactically-pをセットすることにより、M-;の正確な効果を設定できます。たとえば閉じ大カッ

コで終わる行では、M-;は comment-columnではなく、大カッコの後ろにスペースを 1 つ空けてコメントを配します。完全な詳細は、Section “Comment Commands” in *The CC Mode Manual* を参照してください。

23.5.2 複数行のコメント

コメントをタイプするとき、それを次の行に継続したくなったときは、M-jまたは C-M-j (default-indent-new-line) とタイプします。これはカレント行を行ブレークして、コメントを継続するために必要なコメント区切りとインデントを挿入します。

コメントの終了区切りをもつ言語 (たとえば C の ‘*/’) では、M-jの正確な振る舞いは、変数 comment-multi-lineの値に依存します。変数の値が nilの場合、このコマンドは古い行のコメントを終了させて、新しい行で新しいコメントを開始します。そうでない場合、カレントのコメント区切りの中で新しい行を開きます。

Auto Fill モードがオンの場合、コメントをタイプしているときにフィル列に達すると、明示的に M-jを呼び出したときと同じ方法で、コメントが継続されます。

既存の行をコメントにするには、リージョンがアクティブのときは M-;、または M-x comment-regionを使用します。as described in the preceding section.

複数行ブロックのコメントの行の開始で ‘/’をタイプすると、コメントを終了するように C モードを設定できます。これを行なうには、クリーンアップ (clean-up) で、comment-close-slashを有効にします。Section “Clean-ups” in *The CC Mode Manual* を参照してください。

23.5.3 コメントを制御するオプション

Section 23.5.1 [Comment Commands], page 295 で示したように、M-jコマンドが行にコメントを追加するとき、バッファーローカルな変数 comment-columnと comment-fill-column(nilの場合は fill-columnの値。Section 22.6.2 [Fill Commands], page 256 を参照されたい) の間にコメントを配そうと試みます。このバッファーローカルな変数のデフォルト値、またはローカルな値は、通常の方法でセットできます (Section 33.2.3 [Locals], page 511 を参照)。かわりに C-x ; (comment-set-column) とタイプすることにより、カレントバッファーの comment-columnの値を、現在コメントがある列にセットすることができます。C-u C-x ;は、コメント列をそのバッファーのポイントの前にある最後のコメントにセットします。その後で M-;を行うことにより、カレント行のコメントを前のコメントに揃えることができます。

コメントコマンドは、変数 comment-start-skipの値である正規表現にもとづいて、コメントを認識します。この正規表現がヌル文字列にマッチしないように気をつけてください。単語という言葉の厳格な意味から考えると、これはコメント開始区切りより長い文字列にマッチするかもしれません。たとえば C モードでの変数の値としては\"/*+[\\t]*\\\\|//+[\\t]*\"が考えられますが、これは ‘/*’自身とその後ろの余分なアスタリスクやスペースにマッチして、C++スタイルのコメント (‘//’) も許容します (文字列に ‘\’を含む場合には Lisp 構文では ‘\\’と記述する必要があることに注意。これは最初のアスタリスクにたいして正規表現においてアスタリスクがもつ特別な意味を打ち消すために必要)。

コメントコマンドが新しいコメントを作るとき、これは comment-startの値をコメント開始区切りとして挿入します。これはポイントの後ろに、コメント終了区切りとして comment-endの値も挿入します。たとえば Lisp モードでは comment-startが ‘";"’で、comment-endが""(空文字列) です。C モードでは comment-startが"/* "で、comment-endが" */"です。

変数 comment-paddingは、コメントコマンドが、コメント区切りとコメントテキストの間を区切る文字列を指定します。デフォルトでは ‘" "’、つまり 1 つのスペースが指定されます。かわりに数字を指定すると、これは指定した数のスペースになり、nilの場合、スペースは挿入されません。

変数 comment-multi-lineは、M-jと Auto Fill モードがコメントをどのように複数行に継続するかを制御します。Section 23.5.2 [Multi-Line Comments], page 297 を参照してください。

変数 comment-indent-functionは、新しく挿入されたコメント、または既存のコメントの位置揃え位置を計算するために呼び出される関数を指定します。これはメジャーモードごとに異なります。関数は引数なしで呼び出されますが、新しいコメントが挿入されるときはコメント開始位置のポイント、または行末のポイントで呼び出されます。この関数はコメントが開始されるべき列を返す必要があります。たとえば Lisp モードでは、デフォルト関数の決定は、既存のコメントがいくつのコメント文字で始まるかにもとづきます。

さらに Emacs は、コメントを付近の行と揃えるようとも試みます。これをオーバーライドするには、インデントの許容できる範囲を示す、(もしかしたら等しい)2 つの整数から成るコンスを関数がしても構いません。

23.6 ドキュメントの照会

Emacs は、関数やプログラムで使おうと計画している変数およびコマンドを、ドキュメントから探すのに使用できる機能をいくつか提供します。

23.6.1 Info ドキュメントの照会

Info ドキュメントをもつ言語に適用されるメジャーモードでは、プログラムで使用されるシンボルにたいして C-h S (info-lookup-symbol) を使用することにより、Info ドキュメントを閲覧できます。シンボルはミニバッファーで指定します。デフォルトはバッファーのポイント位置にあるシンボルです。たとえば C モードでは、シンボルを C Library Manual から探します。このコマンドは適切な Info ファイルのマニュアルがインストールされているときだけ機能します。

Emacs は、どのドキュメントのどこからシンボルを探すか — つまり、どの Info ファイルを探すのか、そしてどのインデックスを検索するか — をメジャーモードにもとづき決定します。M-x info-lookup-fileを使用して、ドキュメントのファイル名を指定することもできます。

C-h Sをサポートしないメジャーモードでこれを使用すると、これは symbol help mode を指定するよう求めます。ここでは c-modeコマンドのように、C-h Sをサポートするメジャーモードを選択します。

23.6.2 man-page の照会

Unix では、オンラインドキュメントのメインフォームは *manual page* または *man page* です。GNU オペレーティングシステムでは man を、Info でブラウズできる、より組織化されたマニュアルで置き換えることを目指しています。このプロセスは終了していないので、man を読むことはまだ有用です。

オペレーティングシステムのコマンド、ライブラリー関数、システムコールにたいする man page を、M-x manコマンドで読むことができます。このコマンドは補完つき (Section 5.4 [Completion], page 31 を参照してください) でトピック (topic) の入力を求め、対応する man page をフォーマットするために、manコマンドを実行します。そのシステムが許すなら、このコマンドは manを非同期で実行するので、ページがフォーマットされる間、編集を続けることができます。フォーマットされた結果は*Man *topic**という名前のバッファーに表示されます。このバッファーは、Man モードという特別なメジャーモードを使用します。これはスクロールや他の man page にジャンプする機能をもちます。詳細については Man モードのバッファーで、C-h mをタイプしてください。

それぞれの man page は、1 つ以上のセクション *(sections)* に属します。セクションの名前は数字または数字と文字です。同じ名前の man page が、複数のセクションに存在することもあります。特定のセクションの man page を読むには、M-x manがトピックの入力を求める際、'*topic*(*section*)'または '*section topic*'とタイプします。たとえば C のライブラリー関数

chmodはセクション 2 ですが、同じ名前のシェルコマンドの man page はセクション 1 です。前者を閲覧するには、M-x man RET chmod(2) RET とタイプしてください。

セクションを指定しないと、M-x manは通常、最初に見つかった man page だけを表示します。manに、コマンドラインオプション '-a'を指定できるシステムもいくつかあります。これは指定したトピックにたいするすべての man page を表示するよう指定します。これを使用するには、変数Man-switchesの値を '"-a"'に変更します。そうすると Man モードのバッファーで、M-nと M-pで異なるセクションの man pagfe を切り替えることができます。モードラインには、利用可能な man page の数が表示されます。

man page を読む他の方法として、M-x womanコマンドがあります。M-x manとは異なり、これは man page をフォーマットするために外部のプログラムを実行しないので、MS-Windows のような、manプログラムが利用できないかもしれないシステムでも機能します。このコマンドは、表示する man page の入力を求め、それを*WoMan *section topic*という名前のバッファーに表示します。

M-x womanは、コマンドを最初に呼び出したとき、man page の補完リストを計算します。数引数を指定すると、このリストを再計算します。これは man page を追加・削除したとき有用です。

man page の名前を入力して、M-x womanが複数のセクションで同じ名前の man page を見つけた場合、これはウィンドウをポップアップして利用可能な候補を示し、それらから 1 つを選択するよう求めます。

M-x womanはモダンな man-pages の最新機能をまだサポートしていないので、システムで利用可能なら M-x manの使用をわたしたちが推奨していることに注意してください。

M-x womanのセットアップと使用についての情報は、Emacs とともに配布されている WoMan Info マニュアルを参照してください。

23.6.3 プログラミング言語のドキュメントの照会

Emacs Lisp コードを編集するとき、コマンド C-h f (describe-function) および C-h v (describe-variable) で、使用したい関数または変数のビルトインドキュメントを閲覧できます。Section 7.2 [Name Help], page 45 を参照してください。

ElDoc[2] はプログラム内のシンボル (関数、メソッド、クラス、変数等) のドキュメントを照会を手助けするための、バッファーローカルなマイナーモードです。このモードが有効だと、ポイント位置にあるシンボルにドキュメントがある際には常にエコーエリアに有益な情報が表示されます。たとえば Emacs Lisp モードのバッファーではポイント位置にあるのが関数なら引数リスト、Lisp 変数ならドキュメント文字列の 1 行目が表示されます。

ElDoc モードのオンとオフを切り替えるには M-x eldoc-modeとタイプします。Global ElDoc モードも存在します。このモードはデフォルトでオンになっており、以降で説明する変数をセットするようなメジャーモードをもつバッファーでは、ElDoc モードがオンに切り替わります。モードをグローバルにオフに切り替えるには M-x global-eldoc-mode を使用してください。

Global ElDoc モードはさまざまなメジャーモードのドキュメント関数を使用するために、それらのモードによって構成されるモードです。含まれるモードとしてはたとえば Emacs Lisp モード、Python モード、Cfengine モードが挙げられます。更に ElDoc を構成する複数のメジャーモードにたいするサポートを提供する Emacs 機能が挙げられます。この Emacs 機能によって、ElDoc がドキュメントを取得する際にこれらのモードの機能を使うことができるのです。これらの例には言語サーバーからの情報にもとづきドキュメントを提供する Eglot (Section "Eglot Features" in *Eglot: The Emacs LSP Client* を参照)、Semantic の Idle Summary モード (Section "Idle Summary

[2] Emacs Lisp のバッファーのサポートを起源とすることによる歴史的なアクシデントから、このモードは "ElDoc" と命名されてしまいました。

Mode" in *Semantic Manual* を参照)、ポイント位置の診断情報を表示するために ElDoc を使用する Flymake が含まれます (Section "Finding diagnostics" in *GNU Flymake manual* を参照)。

ElDoc モードは Emacs がしばらくの短い間アイドルになった後に、ポイント位置のシンボルにたいして利用可能なドキュメントの表示をスケジュールすることで機能します。これによってあなたが迅速かつ遅延なしでタイプした際に、エコーエリアやモードラインでドキュメント文字列が煩わしくちらついて表示されるのを防がれるのです。

コマンド `M-x eldoc-print-current-symbol-info`を使うことによって、ポイント位置にあるシンボルのドキュメント表示をトリガーすることもできます。

ElDoc モードを構成するために、以下の変数を使用することができます:

`eldoc-idle-delay`
: このユーザーオプションの値は、ポイント位置のドキュメントが表示されるまでのアイドル時間の量を制御します。この値には待機する秒数をセットする必要があります。値 0 は遅延なしで表示を行うことを意味します。デフォルトは 0.5 秒です。

`eldoc-print-after-edit`
: このユーザーオプションが非 `nil`の場合には、何らかのテキストの挿入や削除といった一部の編集コマンドの後だけ ElDoc はドキュメントを表示します。これはすでにバッファーにあるシンボルではなく、あなたがタイプしたシンボルに関するドキュメントだけを Emacs に表示させたい場合に便利です。デフォルト値は `nil`です。この値を変更した場合には、もう一度 `eldoc-mode`のオンとオフを切り替える必要があります。

`eldoc-echo-area-use-multiline-p`
: これはエコーエリアで単一のスクリーン行として表示できない長さのドキュメントテキストを切り詰めるかどうか、切り詰めるとしたらどのように切り詰めるかを制御するユーザーオプションです。値が正の数値なら、それは ElDoc がドキュメントを切り詰めずにエコーエリアに表示できるスクリーン行の行数を指定します。正の整数なら表示に用いるスクリーン行の最大行数の絶対値、正の浮動小数点数ならスクリーン行の行数をフレームの高さにたいする割り合いを指定します。値が `t`ならドキュメントの切り詰めを行わず (エコーエリアは `max-mini-window-height`が許容する高さまでリサイズされる。Section 5.3 [Minibuffer Edit], page 30 を参照)、値が `nil`ならドキュメントがスクリーン行の 1 行より長いと切り詰められることを意味します。最後に特別な値 `truncate-sym-name-if-fit`(デフォルト) は、シンボルの名前を表すドキュメント部分を切り詰めればドキュメントがスクリーン行の 1 行に収まる場合には切り詰めを行うことを意味します。

`eldoc-echo-area-display-truncation-message`
: 非 `nil`(デフォルト) の場合には、エコーエリアに表示されているドキュメントが長すぎるために切り詰められていれば、完全なドキュメントを閲覧する方法についての手順をドキュメントの後に示します。`nil`の場合には、単に '`...`'でドキュメントが切り詰められていることを示します。

`eldoc-echo-area-prefer-doc-buffer`
: このユーザーオプションの値が `t`なら、ElDoc バッファーがどこかのウィンドウでドキュメントを既に表示している場合には、ElDoc はドキュメントをエコーエリアに表示しません (コマンド `M-x eldoc-doc-buffer`を使えば、いつでも ElDoc バッファーを表示できる)。このオプションの値がシンボル `maybe`なら、どこかのウィンドウに ElDoc バッファーが表示されていて、かつドキュメントをエコーエリアに表示するためには切り詰めが必要な場合には、エコーエリアにドキュメントが表示されることはありません。最

後に値が nil(デフォルト) なら、それはドキュメントを常にエコーエリアに表示することを意味します。

eldoc-documentation-strategy
: このカスタマイズ可能な変数には、ポイント位置にあるシンボルのドキュメント取得に用いる関数が保持されています。ドキュメントはフック eldoc-documentation-functions内の関数によって生成されます。eldoc-documentation-strategyのデフォルト値では、eldoc-documentation-functionsフックの関数が最初に生成したドキュメントテキストを表示するよう ElDoc に指示することになっていますが、たとえばすべてドキュメントテキストを結合して表示させる等、別の方法によって動作するよう eldoc-documentation-strategyをカスタマイズできます。

eldoc-documentation-functions
: このアブノーマルフックの値はポイント位置にあるシンボルにたいして、カレントバッファーのメジャーモードに即したドキュメントを生成できる関数のリストです。これらの関数は ElDoc にたいするバックエンドのコレクションとして動作します。メジャーモードはこの変数のバッファーローカルな値にモードのドキュメント照会用の関数を追加することで、それらの関数を Eldoc に登録するのです。

23.7 Hideshow マイナーモード

Hideshow モードは、バッファーローカルなマイナーモードで、ブロックと呼ばれるプログラムの一部を、選択的に表示させることができます。このマイナーモードに切り替えるには、M-x hs-minor-mode とタイプします (Section 20.2 [Minor Modes], page 241 を参照してください)。

ブロックを非表示にするために Hideshow モードを使用したとき、そのブロックはスクリーンに表示されなくなり、かわりに省略記号 (3 つのピリオド) に置き換えられます。何をブロックと定めるかは、メジャーモードに依存します。C モード、および関連するモードでは、ブロックは大カッコ (braces) で区切られ、Lisp モードでは丸カッコ (parentheses) で区切られます。複数行のコメントもブロックとみなされます。

Hideshow モードは以下のコマンドを提供します:

C-c @ C-h
C-c @ C-d カレントブロックを隠します (hs-hide-block)。

C-c @ C-s カレントブロックを表示します (hs-show-block)。

C-c @ C-c
C-c @ C-e
S-mouse-2
: カレントブロックを表示、または非表示にします (hs-toggle-hiding)。

C-c @ C-M-h
C-c @ C-t トップレベルのすべてのブロックを隠します (hs-hide-all)。

C-c @ C-M-s
C-c @ C-a バッファーのすべてのブロックを表示します (hs-show-all)。

C-u *n* C-c @ C-l
: 現在のブロックの、*n*レベル下のすべてのブロックを隠します (hs-hide-level)。

以下の変数は Hideshow モードをカスタマイズするのに使用されます:

hs-hide-comments-when-hiding-all
: 非 nilの場合、C-c @ C-M-h (hs-hide-all) はコメントも隠します。

`hs-isearch-open`
: この変数は、インクリメンタル検索でマッチするテキストが隠されたブロックにあるとき、それを表示すべき条件を指定します。変数の値は、`code`(コードブロックだけを表示)、`comment`(コメントだけを表示)、`t`(コードブロックとコメントの両方を表示)、`nil`(どちらも表示しない) のいずれかです。デフォルト値は `code`です。

23.8 シンボル名の補完

補完は通常ミニバッファーで行われますが (Section 5.4 [Completion], page 31 を参照してください)、シンボル名の補完を、普通の Emacs バッファーで行うこともできます。

プログラミング言語用のほとんどのモードでは、`C-M-i`または `M-TAB`キー [3] はポイント位置にあるシンボルにたいして利用可能な補完リストを生成するコマンド `completion-at-point`を呼び出します。このコマンドは補完候補を導き出すために、利用できるサポート機能を使用します:

- カレントバッファーのプロジェクト (Section 25.2 [Projects], page 352 を参照) とメジャーモードにたいして Eglot がアクティブであれば、補完候補のリストを生成するために対応する言語サーバーの使用を試みます。Section “Eglot Features” in *Eglot: The Emacs LSP Client* を参照してください。
- このコマンドは Semantic モード (Section 23.10 [Semantic], page 303 を参照) が有効なら、補完において Semantic のパーサーデータの使用を試みます。
- Semantic モードが有効でなかったり補完の処理に失敗した場合には、このコマンドは選択済みの tags テーブル (Section 25.4.2 [Tags Tables], page 363 を参照) を用いて補完を試みます。これが機能するためには `M-x visit-tags-table` で tags テーブルをうﾞぃしする必要があります。
- Emacs Lisp モードでは、このコマンドは Emacs のカレントセッションにおいて定義されている関数、変数、あるいはプロパティの名前を用いて補完を実行します。

その他のすべての面において、バッファーでのシンボル補完はミニバッファーでの補完と同様に振る舞います。たとえば Emacs が一意なシンボルを補完できない場合には、他のウィンドウに補完候補のリストを表示します。その後は `M-DOWN`と `M-UP`のキーを使って元のバッファーを離れることなく補完バッファーに表示されている補完候補間を移動、`M-RET`でカレントでハイライトされている補完候補をバッファーに挿入することができます。Section 5.4 [Completion], page 31 を参照してください。

Text モード、および関連するモードでは、`M-TAB`はスペルチェッカーの辞書にもとづいて単語を補完します。Section 13.4 [Spelling], page 134 を参照してください。

23.9 大文字小文字の混ざった単語

いくつかのプログラミングスタイルでは、‘`unReadableSymbol`’のような大文字小文字が混ざった (“CamelCase” の) シンボルを使います (GNU プロジェクトでは、識別子の単語の区切りに、大文字小文字の違いではなく、アンダースコアを使用することを推奨しています)。Emacs には、そのようなシンボルに簡単に対処するための、さまざまな機能があります。

バッファーローカルなマイナーモードの Glasses モードは、そのようなシンボルが表示される方法を変更することにより、それらを読みやすくします。デフォルトでは、小文字とそれに続く大文字の

[3] グラフィカルなディスプレイでは、`M-TAB`キーは通常だとグラフィカルなウィンドウの切り替え用にウィンドウマネージャーに予約されているので、かわりに `C-M-i`または `ESC TAB`とタイプする必要があります。

間に余分なアンダースコアを表示します。これはバッファーのテキストを変更するわけではなく、表示の仕方だけを変更します。

Glasses モードに切り替えるには、`M-x glasses-mode`とタイプします (Section 20.2 [Minor Modes], page 241 を参照してください)。Glasses モードが有効な場合、モードラインのマイナーモードインジケーターには '`o^o`'が表示されます。Glasses モードに間する情報を得るには、`C-h P glasses RET`とタイプしてください。

Subword モードは、バッファーローカルなマイナーモードです。Subword モードでは、Emacs の単語コマンドは、'`StudlyCapsIdentifiers`'のような、単語の中の大文字を単語境界と認識します。Subword モードが有効なときは、モードラインのマイナーモードインジケーターに '`,`'が表示されます。同様なモード `superword-mode`も参照してください (Section 23.11 [Misc for Programs], page 304 を参照してください)。

23.10 Semantic とは

Semantic は、ソースコードパーサー `(source code parsers)`にもとづく、言語認識 (language-aware) による編集のためのコマンドを提供します。このセクションは、Semantic についての簡単な説明を提供します。完全な詳細については、Emacs とともに配布されている Semantic の Info マニュアルを参照してください。

Font Lock モード (Section 11.13 [Font Lock], page 89 を参照してください) のような、Emacs の言語認識機能 (language aware features) のほとんどは、rules of thumb[4] にもとづいています。これは大抵の場合においてよい結果を得られますが、完全に正しい結果は決して得られない、ということを意味します。対照的に、Semantic で使用されるパーサーは、プログラミング言語の構文を正確に理解します。これにより Semantic は、より強力で正確な検索、操作、補完コマンドが提供できるのです。

Semantic の使用を開始するには、`M-x semantic-mode`をタイプするか、'`Tools`'メニューの、'`Source Code Parsers (Semantic)`'という名前のメニューをクリックします。これはグローバルなマイナーモードの Semantic モードを有効にします。

Semantic モードが有効な場合、Emacs はファイルを visit するたびに自動的にパースを試みます。現在のところ、Semantic は C、C++、HTML、Scheme、Java、Javascript、Make、Python、Scheme、SRecord、Texinfo を理解します。パースされたバッファーでは以下のコマンドが利用可能です:

`C-c , j` カレントファイルで定義された関数名の入力を求め、ポイントをそこに移動します (`semantic-complete-jump-local`)。

`C-c , J` Emacs がパースした任意のファイルで定義された関数名の入力を求め、ポイントをそこに移動します (`semantic-complete-jump`)。

`C-c , SPC` ポイント位置のシンボルにたいして可能な補完候補のリストを表示します (`semantic-complete-analyze-inline`)。これは補完候補を選択するための特別なキーバインドのセットをアクティブにします。`RET`はカレントの補完候補を選択し、`M-n`と `M-p`は可能な補完候補を巡回、`TAB`は可能なところまで補完を行ってから巡回、そして、`C-g`または他のキーは補完を中止します。

`C-c , l` ポイント位置のシンボルにたいして可能な補完候補のリストを、他のウィンドウに表示します (`semantic-analyze-possible-completions`)。

[4] 正規表現と構文テーブル。

上記のコマンドに加えて、Semantic パッケージは、パーサー情報を使用する他のさまざまな方法を提供します。たとえば、Emacs がアイドルのとき、補完候補のリストを表示するために、それを使用することができます。

23.11 プログラムを編集するための他の便利な機能

プログラムを編集するためにデザインされているわけではありませんが、有用な Emacs コマンドもいくつかあります

単語、センテンス、パラグラフを操作する Emacs コマンドは、コードを編集するのに有用です。ほとんどのシンボル名は単語 (Section 22.1 [Words], page 251 を参照してください) を含んでおり、文字列やコメントの中でセンテンス (Section 22.2 [Sentences], page 252 を参照してください) を見つけることができます。パラグラフについては、ほとんどのプログラミング言語のモードは空行をパラグラフの開始および終了に定義しています。したがって空行を注意深く使用してプログラムをクリアーにすることにより、パラグラフコマンドが機能できる、有意なテキストの集合を提供することができます。プログラミング言語のモードで Auto Fill モードが有効な場合、新しい行の作成でインデントされるようになります。

Superword はバッファーローカルなマイナーモードで、編集および移動コマンドがシンボル (たとえば ‘`this_is_a_symbol`’) を単語として扱うようになります。Superword モードが有効な場合、モードラインのマイナーモードインジケーターに、‘²’ が表示されます。同様なモード `subword-mode` も参照してください (Section 23.9 [MixedCase Words], page 302 を参照してください)。

Electric Layout モード (`M-x electric-layout-mode`) はグローバルなマイナーモードで、特定の文字をタイプしたときに自動的に改行を挿入します。たとえば Javascript モードでは ‘`{`’、‘`}`’、‘`;`’などです。

Hideshow モード (Section 23.7 [Hideshow], page 301 を参照してください) は別として、プログラムの一部を選択的に表示するには、選択的な表示機能 (Section 11.18 [Selective Display], page 96 を参照してください) を使う方法があります。プログラミングのモードには、Foldout パッケージ (Section 22.9.6 [Foldout], page 265 を参照してください) とともに使用できる、Outline minor マイナーモード (Section 22.9 [Outline Mode], page 261 を参照してください) をサポートするものもあります。

Prettify Symbols モードはバッファーローカルなマイナーモードで、特定の文字の表示をよりアトラクティブ (attractive: 見栄えのする) なバージョンに置き換えます。たとえば Emacs Lisp モードでは、文字列 ‘`lambda`’を、ギリシャ文字のラムダ ‘λ’に置き換えます。TEX バッファーでは ‘`\alpha`’ ... ‘`\omega`’、および他の数学マクロ (math macros) を、対応する Unicode 文字に置き換えます。このモードを、プログラミングとは関係のないモードで使いたいと思うかもしれません。`prettify-symbols-alist`にエントリーを追加することにより、このモードをカスタマイズできます。デフォルト値である `prettify-symbols-default-compose-p`の値が適切でない場合、`prettify-symbols-compose-predicate`をカスタマイズすることで、より入念なカスタマイズが可能です。グローバルなバージョン `global-prettify-symbols-mode`は、サポートするすべてのモードでこれを有効にします。

ポイント位置のシンボルを、元の形式で表示することもできます。これは変数`prettify-symbols-unprettify-at-point`により制御されます。非 `nil`の、ポイントがシンボル位置にある限り、そのシンボルの元の形式が復元されます。

23.12 C および関連するモード

このセクションでは C、C++、Objective-C、Java、CORBA IDL、Pike、AWK のためのモード (これらは “C モードおよび関連するモード” と呼ばれます) で利用できる、特別な機能を簡単に説明します。詳細は、Emacs とともに配布される CC mode の Info マニュアルを参照してください。

23.12.1 C モードの移動コマンド

このセクションでは、C モードおよび関連するモードで、ポイントを移動するコマンドを説明します。

`C-M-a`
`C-M-e` カレント関数またはトップレベルの定義の、先頭または最後にポイントを移動します。スコープによる囲い (C++の class など) をもつ言語では、カレント関数 *(current function)* はスコープ内の隣接する関数です。そうでない場合、それは大カッコ (braces) で囲まれることにより定義されます。Section 23.2.2 [Moving by Defuns], page 286 を参照してください。

`C-c C-u` マークを置いたまま、ポイントを含むプリプロセッサー条件を後方に移動します。プレフィクス引数は、繰り返し回数として振る舞います。負の引数を指定すると、ポイントを含むプリプロセッサー条件の最後に、前方へポイントを移動します。

‘`#elif`’は、‘`#else`’の後に ‘`#if`’を続けたのと同じなので、この関数は後方に移動するときは ‘`#elif`’で止まりますが、前方に移動するときは止まりません。

`C-c C-p` マークを置いたまま、プリプロセッサー条件を越えてポイントを後方に移動します。プレフィクス引数は、繰り返し回数として振る舞います。負の引数の場合は前方に移動します。

`C-c C-n` マークを置いたまま、プリプロセッサー条件を越えて、ポイントを前方に移動します。プレフィクス引数は、繰り返し回数として振る舞います。負の引数の場合は後方に移動します。

`M-a` 一番内側の C ステートメントの先頭に、ポイントを移動します (`c-beginning-of-statement`)。すでにポイントがステートメントの先頭にある場合は、その前のステートメントの先頭に移動します。プレフィクス引数 nを指定した場合、$n-1$ 個前のステートメントに、後方へ移動します。

2 行以上のコメントまたは文字列の場合、このコマンドはステートメントではなくセンテンス単位で移動します。

`M-e` C ステートメントまたはセンテンスの最後に、ポイントを移動します。`M-a`と同様ですが、これは逆の方向に移動します (`c-end-of-statement`)。

23.12.2 エレクトリック C 文字

C モードおよび関連するモードでは、特定の文字はエレクトリック *(electric)* — つまり自分自身を挿入するのに加えて、カレント行を再インデントしたり、オプションで改行を挿入します。エレクトリックな文字は、`{`、`}`、`:`、`#`、`;`、`,`、`<`、`>`、`/`、`*`、`(`、`)`です。

混乱したインデントのコードを編集している場合には、エレクトリックなインデントを不便だと感じるかもしれません。CC モードになれていない場合、それはあなたを当惑させるかもしれません。エレクトリックな動作は、コマンド `C-c C-l`で切り替えることができます。有効な場合には、モードラインのモード名の後ろに ‘`/cl`’が表示されます (*c*が表示される場合、それはコメントスタイルがブロックスタイルかラインスタイルかに応じて ‘`*`’か ‘`/`’が表示される)。

C-c C-l エレクトリックな動作を切り替えます (c-toggle-electric-state)。正のプレフィクス引数を指定した場合、、このコマンドはエレクトリックな動作を有効にし、負の引数の場合は無効にします。

エレクトリックな文字は、エレクトリックな状態に加えて、自動改行 *(auto-newline)* の機能が有効な場合 (モードラインのモード名の後ろに '/cla'が表示されている状態のとき) だけ、改行を挿入します。この機能は C-c C-aで、オンまたはオフに切り替えることができます。

C-c C-a 自動改行機能を切り替えます (c-toggle-auto-newline)。このコマンドにプレフィクス引数を指定した場合、引数が正のときは自動改行機能をオン、負のときはオフに切り替えます。

CC モードのスタイルは通常、Emacs が自動改行する正確な状況を設定します。これを直接設定することもできます。Section "Custom Auto-newlines" in *The CC Mode Manual* を参照してください。

23.12.3 C の欲張りな削除機能

ポイント位置の空白文字のブロック全体を削除したい場合は、*hungry deletion(*欲張りな削除*)* を使うことができます。これはポイントの前後の、一連の空白文字を 1 回の操作で削除します。空白文字 *(whitespace)* にはタブと改行が含まれますが、コメントとプリプロセッサーコマンドは含まれません。

C-c C-DEL
C-c DEL ポイントの前の空白文字のブロック全体を削除します (c-hungry-delete-backwards)。

C-c C-d
C-c C-Delete
C-c Delete
ポイントの後の空白文字のブロック全体を削除します (c-hungry-delete-forward)。

上記のコマンドのかわりに、*hungry delete* モードを有効にすることができます。この機能が有効な場合 (モードラインのモード名の後に '/'と 'h'が示されます)、1 回の DELで 1 つのスペースではなくポイントの前に続くすべての空白文字を削除し、C-d(Deleteではありません) でポイントの後に続くすべての空白文字を削除します。

M-x c-toggle-hungry-state
hungry-delete 機能を切り替えます (c-toggle-hungry-state)。このコマンドにプレフィクス引数を指定した場合、正の場合は hungry-delete 機能をオン、負の場合はオフにします。

変数 c-hungry-delete-keyは、hungry-delete 機能を有効にするかを制御します。

23.12.4 C モードのその他のコマンド

M-x c-context-line-break
このコマンドは、コンテキストに応じたマナーで、行ブレークの挿入と新しい行のインデントを行います。通常のコードの中では、RET (newline) の働きをしますが、C プリプロセッサー行の中では、行ブレークに追加で '\'を挿入し、コメントの中では M-j (c-indent-new-comment-line) のように動作します。

デフォルトでは、c-context-line-breakはキーにバインドされていませんが、便利に使うためには、キーにバインドする必要があります。以下のコードは、このコマンド

を RETにバインドします。この例ではキーマップを変更する前に、それがロードされていることを確実にするために c-initialization-hookを使用しています。

```
(defun my-bind-clb ()
  (keymap-set c-mode-base-map "RET"
              'c-context-line-break))
(add-hook 'c-initialization-hook 'my-bind-clb)
```

C-M-h マークを関数定義の最後に配し、ポイントを先頭に配します (c-mark-function)。

M-q C および C++のコメントに対応した、パラグラフのフィルを行います (c-fill-paragraph)。カレント行がコメントを含む、またはカレント行がコメントの場合、このコマンドはコメントのインデントとコメント区切りを保ちながら、コメントまたはポイントがあるパラグラフをフィルします。

C-c C-e リージョンのテキストにたいして、C プリプロセッサーを実行して、すべてのマクロ呼び出しの展開を含めて、結果を表示します。リージョンの前にあるバッファーのテキストも、マクロ定義がそこにあるためにプリプロセスされますが、この部分は出力には表示されません。

マクロを使用する C コードをデバッグするとき、マクロがどのように展開されるか正確に解明するのが難しいときがあります。このコマンドにより、展開結果を見ることができるので、わざわざ解明する必要がなくなります。

C-c C-\ リージョン内の各行末に、'\'文字を挿入、または位置揃えします (c-backslash-region)。これは C マクロ定義を、記述または編集した後に便利です。

行がすでに (c-backslash-region) で終了されている場合、このコマンドはそれの前の空白文字の数を調整します。そうでない場合は、新規に '\'を挿入します。しかしリージョンの最後の行は特別に扱われます。この行には '\'は挿入されず、もし '\'がある場合は削除します。

M-x cpp-highlight-buffer

プリプロセッサー条件に対応するテキスト部分を、ハイライトします。このコマンドは *CPP Edit*という名前の別のバッファーを表示します。これは特定の条件とその内容をどのように表示するかを選択するための、グラフィックメニューを供します。さまざまなセッティングを変更した後に、'[A]pply these settings'をクリック (またはそのバッファーで aをタイプ) すると、それに応じて C モードのバッファーが再ハイライトされます。

C-c C-s カレントソース行の構文的な情報を表示します (c-show-syntactic-information)。この情報は、その行がどのようにインデントされるべきか決定するために使用されます。

M-x cwarn-mode

M-x global-cwarn-mode

CWarn マイナーモードは、ある種の疑わしい C および C++の構文をハイライトします。

- 式の中での値の割り当て。
- 'if'、'for'、'while'('do ... while'命令は除く) の直後のセミコロン。
- 参照パラメーターをともなう C++関数。

このモードを 1 つのバッファーにたいして有効にするにはコマンド M-x cwarn-mode、すべての適合するバッファーにたいして有効にするにはコマンド M-x global-cwarn-mode、または変数 global-cwarn-modeをカスタマイズします。これが機能するためには、Font Lock モードも有効にしなければなりません。

`M-x hide-ifdef-mode`
: Hide-ifdef マイナーモードは、プリプロセッサーブロック '`#if`' および '`#ifdef`' の中の選択されたコードを隠します。変数 `hide-ifdef-shadow` を `t` に変更した場合、Hide-ifdef マイナーモードははプリプロセッサーブロックを隠すかわりに、より目立たないフェイスでそれらのブロックを shadow します。詳細は、`hide-ifdef-mode` のドキュメント文字列を参照してください。

`M-x ff-find-related-file`
: カレントバッファーで visit されたファイルに関連するファイルを、特別な方法で検索します。通常これは C/C++ ソースファイルにたいするヘッダーファイル、またはその逆です。変数 `ff-related-file-alist` は、関連するファイル名をどのように計算するかを指定します。

23.13 Asm モード

Asm モードは、アセンブラーコードのファイルを編集するためのメジャーモードです。このモードは、以下の 3 つのコマンドを定義します:

`TAB`
: `tab-to-tab-stop`.

`C-j`
: 改行を挿入してから、`tab-to-tab-stop` を使ってインデントします。

`:`
: コロンを挿入してから、コロンの前のラベルからインデントを削除します。その後、`tab-to-tab-stop` を実行します。

`;`
: コメントの挿入または位置揃えをします。

変数 `asm-comment-char` はアセンブラー構文でコメントを開始する文字を指定します。

24 プログラムのコンパイルとテスト

前のチャプターでは、プログラムを変更するのに便利な Emacs コマンドについて議論しました。このチャプターでは、プログラムのコンパイルとテストに役立つコマンドを扱います。

24.1 Emacs 下でのコンパイルの実行

Emacs は、C や Fortran のような言語のためのコンパイラーを実行でき、コンパイルログを Emacs のバッファーに取り込むことができます。エラーメッセージを解析して、エラーが発生した場所を示すこともできます。

`M-x compile`
: Emacs 下で非同期にコンパイラーを実行し、エラーメッセージは`*compilation*`バッファーに送られます。

`M-x recompile`
`g` (Compilation mode)
: 最後に呼び出した `M-x compile`と同じコマンドで、コンパイラーを呼び出します。

`M-x kill-compilation`
: サブプロセスで実行されているコンパイルを kill します。

`make`、または他のコンパイルコマンドを実行するには、`M-x compile`とタイプします。これはミニバッファーを使用してシェルのコマンドラインを読み取り、シェルを Emacs のサブプロセス (または下位プロセス *(inferior process)*) として、そのコマンドを実行します。出力は`*compilation*`という名前のバッファーに挿入されます。カレントバッファーのデフォルトディレクトリーが、コマンドを実行する作業ディレクトリーとして使用されます。したがって、通常はそのディレクトリーでコンパイルが行われます。

デフォルトのコンパイルコマンドは‘`make -k`’で、これは`make`ユーティリティーを使ってコンパイルするプログラムにたいして通常正しいコマンドです (‘`-k`’フラグは`make`に、エラー後も可能な限りコンパイルを継続するよう指示します)。Section “Make” in *GNU Make Manual* を参照してください。前に `M-x compile`を実行している場合、それに指定したコマンドは自動的に変数 `compile-command` に格納されます。これは、次に `M-x compile`とタイプしたときのデフォルトとなります。ファイルのファイルローカルな値で `compile-command`を指定することもできます (Section 33.2.4 [File Variables], page 512 を参照してください)。

コンパイルを開始すると、他のウィンドウで`*compilation*`バッファーが表示されますが、そのウィンドウは選択されません。コンパイルが実行中は、`*compilation*`バッファーのメジャーモードインジケーターに‘`run`’という単語が表示され、単語‘`Compiling`’がすべてのモードラインに表示されます。コンパイル実行中、常に`*compilation*`バッファーを表示している必要はありません。表示されていなくてもコンパイルは継続します。何らかの理由によりコンパイルが終了したときは、`*compilation*`バッファーのモードラインが‘`exit`’(その後に終了コード。‘`[0]`’の場合は通常終了)、または‘`signal`’(何らかのシグナルがプロセスを終了させた場合) に変化します。

コンパイルの経過を見たいときは、バッファー`*compilation*`に切り替えて、ポイントをバッファーの最後に移動します。ポイントが最後にある場合、コンパイル出力はポイント位置に挿入されるので、ポイントは最後に留まります。そうでない場合は、バッファーの最後にコンパイル出力が追加される間も、ポイント位置は固定されたままです。

コンパイル処理中、モードラインにはその時点までのエラー、警告の数と、コンパイラーからの情報が表示されます。

変数 compilation-scroll-outputを非 nil値に変更した場合、*compilation*バッファーは出力に追随して自動的にスクロールします。値が first-errorの場合は、最初のエラーが出現した箇所でスクロールがストップし、ポイントはエラー箇所に留まります。その他の任意の非 nil値の場合は、出力がなくなるまでスクロールが継続されます。

最後にコンパイルしたのと同じコマンドで再実行するには、M-x recompileとタイプします。これは最後に呼び出した M-x compileからコンパイルコマンドを再利用します。これは*compilation*バッファーも再利用し、コンパイルもそのバッファーのデフォルトディレクトリー、つまり前にコンパイルが開始されたのと同じディレクトリーで行われます。*compilation*バッファーでは、このコマンドは gにバインドされています。

新しいコンパイルの開始は、すでに*compilation*で実行中のコンパイルを kill します。これは、そのバッファーが 1 度に 1 つのコンパイルしか処理できないからです。しかし実行中のコマンドを実際に kill する前に、M-x compile、および M-x recompileは確認を求めます。常に確認なしで自動的にコンパイルを kill するには、変数 compilation-always-killを tに変更します。コマンド M-x kill-compilationで、コンパイルプロセスを kill することもできます。

1 度に 2 つのコンパイルを実行するには、最初に 1 つを開始してから (多分 rename-uniquely を使用して。Section 16.3 [Misc Buffer], page 179 を参照してください)*compilation*、バッファーをリネームして、それからバッファーを切り替えて他のコンパイルを開始します。これにより新しい*compilation*バッファーが作成されます。

コンパイルコマンドに渡される環境は、変数 compilation-environmentで制御できます。この変数の値は環境変数のセッティングのリストで、各要素は文字列"*envvarname*=*value*"の形式です。これらの環境変数のセッティングは、通常の値をオーバーライドします。

コンパイル出力中で非常に長い行を表示することによって、Emacs の速度が低下する恐れがあります。compilation-max-output-line-lengthよりも長い行には、その制限を超過した部分にボタンが表示されて、そのボタンをクリックすれば隠された部分を表示することができます。この変数を nilにセットすれば、何も非表示にはされなくなります。

24.2 Compilation モード

*compilation*バッファーは、Compilation モードと呼ばれるメジャーモードを使用します。Compilation モードは、バッファーのエラーメッセージをハイパーリンクに変換します。ポイントをそこに移動して RETをタイプするか、マウスでクリック (Section 18.3 [Mouse References], page 197 を参照してください) すると、別のウィンドウでエラーメッセージの *locus* を visit します。locus とは、エラーが発生したファイルの特定の位置を意味します。

エラーメッセージや警告メッセージにたいする compilation-errorと compilation-warning のように、*compilation*バッファーの外観は、*compilation*バッファーのハイライト箇所に使用されるフェイスをカスタマイズすることにより制御されます。これらのフェイスは errorフェイスと warningフェイスから継承されるので、親フェイスを直接カスタマイズすることも可能なことに注意してください。

カスタマイズに関する変数とフェイスの完全なリストを確認するには、M-x customize-group RET compilation を使用してください。

変数 compilation-auto-jump-to-first-errorを非 nil値に変更した場合、Emacs は、*compilation*バッファーに表れる最初のエラーメッセージの locus を自動的に visit します (自動的にエラーの locus を visit する条件を変更するために、この変数に if-location-knownや first-knownを指定することもできる)。

Compilation モードは、以下の追加のコマンドを提供します。これらのコマンドは*grep*バッファーでも使用できます。このバッファーではエラーメッセージのかわりに、検索にたいするマッチにハイパーリンクが設定されます (Section 24.4 [Grep Searching], page 313 を参照してください)。

M-g M-n
M-g n
C-x `　次のエラーメッセージ (またはマッチ) の locus を visit します (next-error)。

M-g M-p
M-g p　前のエラーメッセージ (またはマッチ) の locus を visit します (previous-error)。

M-n　locus を visit せずに、ポイントを次のエラーメッセージ (またはマッチ) に移動します (compilation-next-error)。

M-p　locus を visit せずに、ポイントを前のエラーメッセージ (またはマッチ) に移動します (compilation-previous-error)。

M-}　他のファイルで発生した次のエラーメッセージ (またはマッチ) にポイントを移動します (compilation-next-file)。

M-{　他のファイルで発生した前のエラーメッセージ (またはマッチ) にポイントを移動します (compilation-previous-file)。

C-c C-f　Next Error Follow マイナーモードに切り替えます。これは compilation バッファーでのカーソル移動にしたがって、ソースを自動的に表示するモードです。

g　出力が*compilation*バッファー内に表示されている、最後のコマンドを再実行します。

M-x next-error-select-buffer
　next-errorと previous-errorの次回の呼び出しで使用されるバッファーを選択します。

順番にエラーを visit するには、C-x ` (next-error) とタイプするか、これと等価な M-g M-nまたは M-g nとタイプします。このコマンドは Compilation モードのバッファーだけでなく、任意のバッファーから呼び出すことができます。コンパイル後に最初に呼び出すときは、最初のエラーメッセージの locus を visit します。連続した M-g M-n は、同じ方法で次のエラーを visit します。*compilation* バッファーから RETまたはマウスクリックで特定のエラーを visit した場合、M-g M-n はそのエラーの次のエラーから visit していきます。これ以上 visit するエラーメッセージがない場合、M-g M-n はエラーをシグナルします。C-u M-g M-n は compilation バッファーの先頭から再開して、最初の locus を visit します。

M-g M-pまたは M-g p (previous-error) は、反対方向にエラーを巡回します。

コマンド next-errorおよび previous-errorは、バッファー*compilation*または*grep*にリストされたエラー (またはマッチ) だけに作用されるわけではありません。これらのコマンドは M-x occur (Section 12.11 [Other Repeating Search], page 126 を参照のようなコマンドで生成されたエラー (またはマッチ) を巡回する方法も知っています。カレントバッファーがエラーメッセージ、またはマッチを含む場合には、これらのコマンドはそれらを巡回するでしょう。そうでなければ Emacs は (変数 next-error-find-buffer-functionの値を next-error-buffer-on-selected-frameにカスタマイズしていれば) 選択されたフレームのウィンドウの中からエラーメッセージ (またはマッチ) を含むバッファー、次に next-errorまたは previous-errorが最後に使用したバッファー、最後にその他のすべてのバッファーを探します。これらのコマンドが巡回するために選択されたバッファーがカレントでウィンドウに表示されていなければ、そのバッファーが表示さ

れるでしょう。その後の next-error呼び出しで使用される別バッファーへの切り替えには、コマンド next-error-select-bufferを使用できます。

デフォルトでは、コマンド next-errorおよび previous-errorは、重要でないメッセージはスキップします。変数 compilation-skip-thresholdが、これを制御します。デフォルト値は 1 で、これは警告 (warning) より重要でないメッセージをスキップします。2 の場合、エラー (error) より重要でないものをスキップし、0 はメッセージをスキップしません。

Emacs がエラーメッセージの locus を visit しているとき、関連するソース行が一時的にハイライトされます。選択されたバッファーの locus のハイライトの持続時間は変数 next-error-highlight、それ以外のバッファーでは next-error-highlight-no-selectにより決定されます。更にメッセージを含むバッファーでカレントエラーメッセージをハイライトする方法を定義する、変数 next-error-message-highlightをカスタマイズすることもできます。

*compilation*バッファーが左フリンジ (Section 11.15 [Fringes], page 93 を参照してください) のあるウィンドウで表示されている場合、locus を visit するコマンドはカレントエラーメッセージを指す矢印をフリンジに配します。テキスト端末のように、左フリンジがないウィンドウの場合、これらのコマンドは、カレントメッセージがウィンドウの一番上にくるようにウィンドウをスクロールします。変数 compilation-context-linesを tに変更すると、かわりに列 0 の前に可視の矢印を挿入します。整数値 *n*に変更した場合には、これらのコマンドはフリンジの有無に関わらずメッセージがウィンドウの上から *n*行目にくるようにウィンドウをスクロールします。デフォルト値の nilでは上述のように振る舞います。

コンパイル出力が非常に冗長になることもありますが、多くの場合にはユーザーにとって特に重要ではありません。ユーザーオプション compilation-hidden-outputに regexp か regexp のリストをセットすることによって、それらにマッチする出力を非表示にできます。たとえば再帰的な Makefile からの冗長な出力を非表示にするために、以下のような指定ができます:

```
(setq compilation-hidden-output
      '("^make[^\n]+\n"))
```

コンパイラーからのメッセージを解析するために、Compilation モードは変数 compilation-error-regexp-alistを使用します。これはさまざまなエラーメッセージのフォーマットをリストし、それらから locus を抽出する方法を Emacs に指示します。同じような変数 grep-regexp-alistは、grepコマンド (Section 24.4 [Grep Searching], page 313 を参照してください) の出力を解析する方法を指示します。

Compilation モードは、スクリーン単位でスクロールを行うために、キー SPCおよび DELも定義します。M-n (compilation-next-error) および M-p (compilation-previous-error) は、次または前のエラーメッセージに移動します。M-{ (compilation-next-file) および M-} (compilation-previous-file) は、違うソースファイルの、次または前のエラーメッセージに移動します。

C-c C-fとタイプして、Next Error Follow モードに切り替えることができます。このマイナーモードでは、compilation バッファーでの通常のカーソル移動により、自動的にソースを表示するバッファーが更新されます。たとえばカーソルをエラーメッセージに移動すると、そのエラーにたいする locus が表示されます。

Compilation モードの機能は、Compilation Minor モード呼ばれるマイナーモードでも利用可能です。これは通常のコンパイル出力のバッファーだけでなく、任意のバッファーのエラーメッセージを解析します。M-x compilation-minor-modeとタイプすることにより、このマイナーモードが有効になります。たとえば Rlogin バッファー (Section 31.5.10 [Remote Host], page 468 を参照してください) では、Compilation minor モードはリモートのソースファイルに、FTP を通じて自動的にアクセスします (Section 15.1 [File Names], page 145 を参照してください)。

24.3 コンパイルのためのサブシェル

このセクションには、compilation バッファー内で、シェルやその機能を使用するための、さまざまなテクニックとアドバイスが含まれています。ローカルでのコンパイルに特有なトピックを扱うので、デフォルトディレクトリーがリモートホスト上であるような compilation バッファーでは、おそらくほとんどは機能しない (または無関係) でしょう。

`M-x compile`コマンドは、コンパイルコマンドを実行するためにシェルを使いますが、オプションで非対話的なシェルを指定します。これは、シェルがプロンプトなしで開始されることを意味します。`*compilation*`バッファーで、通常のシェルプロンプトの見映えがよくない場合、それはシェルの初期化ファイルで、無条件にプロンプトをセットするという間違いを犯していることを意味します (この初期化ファイルは使用しているシェルに応じて`.bashrc`、`.profile`、`.cshrc`、`.shrc`などの名前がついています)。シェルの初期化ファイルでは、プロンプトがすでにあるときだけプロンプトをセットするべきです。これを bash で行うには、以下のようにします:

```
if [ "${PS1+set}" = set ]
then PS1=...
fi
```

csh で行うには以下のようにします:

```
if ($?prompt) set prompt = ...
```

compilation のサブシェルに渡す環境変数 `TERM`の値をカスタマイズしたい場合は、変数`comint-terminfo-terminal`の値をカスタマイズしてください (Section 31.5.7 [Shell Options], page 466 を参照)。

Emacs は、コンパイラープロセスが非同期なサブプロセスで実行されることを要求しません。もしこれを行う場合、メインのコンパイラープロセスが終了した後で、サブプロセスがまだ実行中のときは、Emacs はこれらを kill するか、それらの出力は Emacs には到達しません。この問題を避けるには、メインのコンパイルプロセスが、それのサブプロセスの終了まで wait するようにします。シェルスクリプトでは、以下のように‘`$!`’と‘`wait`’を使用して、これを行うことができます:

```
(sleep 10; echo 2nd)& pid=$!  # サブプロセスの pid を記録
echo first message
wait $pid                     # サブプロセスの wait
```

バックグラウンドのプロセスが compilation バッファーに何も出力せず、メインのコンパイルプロセスが終了したときに、これらが kill されるのを防ぐことだけが必要な場合は、以下で充分です:

```
nohup command; sleep 1
```

24.4 Emacs 下での Grep による検索

Emacs からコンパイラーを実行して、コンパイルエラーの行を visit できるように、`grep`を実行して見つかったマッチの行を visit することもできます。これは`grep`が報告するマッチを、エラーのように扱うことで機能します。出力バッファーは Grep モードを使用します。これは Compilation モードの変種です (Section 24.2 [Compilation Mode], page 310 を参照してください)。

`M-x grep`
`M-x lgrep` Emacs 下で`grep`を非同期で実行し、`*grep*`という名前のバッファーにマッチした行をリストします。

`M-x grep-find`
`M-x find-grep`
`M-x rgrep` `find`を通じて`grep`を実行し、出力を`*grep*`バッファーに収集します。

`M-x zrgrep`
: `zgrep`を実行して、出力を`*grep*`バッファーに収集します。

`M-x kill-grep`
: 実行中の `grep`サブプロセスを kill します。

`grep`を実行するには、`M-x grep`とタイプしてから、どのように `grep`を実行するかを指定するコマンドラインを入力します。これは通常、`grep`を実行するとき与える引数と同じです。`grep`スタイルの regexp(通常、シェルのスペシャル文字をクォートするためシングルクォートで囲む) の後に、ファイル名 (ワイルドカードも使用できる) を続けます。`M-x grep`にプレフィクス引数を指定した場合、バッファーのポイント位置周辺の識別子 (Section 25.4 [Xref], page 357 を参照してください) を探して、それを `grep`コマンドのデフォルトにします。

指定するコマンドは、単純に `grep`を実行するものである必要はありません。同じフォーマットで出力を生成するシェルコマンドを使用することができます。たとえば、以下のように、`grep`コマンドを連結することができます:

```
grep -nH -e foo *.el | grep bar | grep toto
```

`grep`コマンドの出力は、`*grep*`バッファーに送られます。オリジナルのファイルの対応する行は、コンパイルエラーと同様、`M-g M-n`、`RET`などで見つけることができます。コマンドのより詳細な説明と、`*grep*`バッファー内で利用可能なキーバインディングについては、Section 24.2 [Compilation Mode], page 310 を参照してください。

マッチをハイライトするために、その周囲に特別なマーカーを出力する '`--color`'オプションを指定できる grep プログラムもあります。この機能を使うには、`grep-highlight-matches`を `t`にセットします。これによりソースバッファーのマッチを表示するとき、ソース行全体ではなく、正確なマッチだけがハイライトされます。ハイライトは `grep`が発行する ANSIエスケープシーケンスにたいするマッチを通じて提供されます。このシーケンスにたいするマッチングは `grep-match-regexp`により制御されており、これは別の `grep`プログラムに対応するようにカスタマイズできます。

コンパイルコマンド (Section 24.1 [Compilation], page 309 を参照) のときと同様、grep コマンド実行中には、モードラインにはそれまでに見つかったマッチ数が表示されて、ハイライトされます。

`grep`コマンドは、実行前にバッファーの保存を提案するでしょう。これは、変数 `grep-save-buffers`により制御されます。利用できる値は `nil`(保存しない)、`ask`(保存前に尋ねる)、または述語として使用される関数 (ファイル名をパラメーターとして呼び出され、バッファーを保存する場合は非 `nil`をリターンすべきである) のいずれかである。その他の非 `nil`値は、すべてのバッファーが確認なしで保存されるべきであることを意味します。デフォルト値は `ask`です。

コマンド `M-x grep-find`(`M-x find-grep`でも利用可能) は、`M-x grep`と似ていますが、コマンドにたいして提供される初期のデフォルトが異なります — このデフォルトは `find`と `grep`の両方を実行するもので、これによりディレクトリーツリーの各ファイルを検索できます。Section 27.16 [Dired and Find], page 397 の `find-grep-dired`コマンドも参照してください。

コマンド `M-x lgrep` (local grep) および `M-x rgrep` (recursive grep) は、`grep`および `grep-find`のユーザーフレンドリーなバージョンで、これらはマッチにたいする正規表現、検索するファイル、検索の基準となるディレクトリーを個別に尋ねます。検索での大文字小文字の区別は、`case-fold-search`の値で制御されます。コマンド `M-x zrgrep`は `M-x rgrep`と似ていますが、これは `grep`のかわりに `zgrep`を呼び出し、gzip されたファイルの内容を検索します。

これらのコマンドは、変数 `grep-template`(`lgrep`用)、および `grep-find-template`(`rgrep`用) にもとづいてシェルコマンドを構築します。検索するファイルには、変数 `grep-files-aliases`で定義されたエイリアスを使用できます。

変数 grep-find-ignored-directoriesにリストされたディレクトリーは、M-x rgrepの検索で自動的にスキップされます。デフォルト値には、さまざまなバージョンコントロールシステムで使用されるデータディレクトリーが含まれます。

デフォルトでは lgrep、rgrep、zgrepにたいして構築されるシェルコマンドは、ファイルとディレクトリーの長いリストを含む部分を隠蔽することにより省略して表示されます。隠蔽された部分を表す省略記号のボタンをクリックすれば隠蔽部分を表示できます。M-x grep-find-toggle-abbreviation とタイプしてインタラクティブに隠蔽部分の表示を切り替えることもできます。このシェルコマンドの隠蔽を無効にするにはオプション grep-find-abbreviateを nil値にカスタマイズしてください。

24.5 オンザフライで構文エラーを見つける

Flymake モードは C、C++、Perl、HTML、TEX/LATEX を含む、多くのプログラミング言語およびマークアップ言語の構文チェックを、オンザフライ (on-the-fly) で処理するマイナーモードです。これは通常の人間の言語にたいしてスペルチェックを処理する、Flyspell モード (Section 13.4 [Spelling], page 134 を参照してください) と、その方法において類似しています。Flymake モードはファイルの編集にしたがい、そのバッファーの一時的なコピーを使用して、適切な構文チェックツールをバックグラウンドで実行します。それからエラーメッセージと警告メッセージを解析して、そのバッファーの間違った行をハイライトします。使用される構文チェックツールは、言語に依存します。たとえば通常、C/C++ファイルの場合は、C コンパイラーです。Flymake は、複雑なプロジェクトにたいしてのチェックでは、makeのようなビルドツールを使うこともできます。

Flymake モードを有効にするには、M-x flymake-modeとタイプします。M-x flymake-goto-next-error および M-x flymake-goto-prev-error を使用して、これが見つけたエラーにジャンプすることができます。カレントバッファーにたいする診断の概観について詳細を表示するにはコマンド M-x flymake-show-buffer-diagnostics、同じようにプロジェクト全体 (Section 25.2 [Projects], page 352 を参照) にたいする診断の概観を表示!にはコマンド M-x flymake-show-project-diagnostics を使用してください。

Flymake の使用についての詳細は、Emacs とともに配布されている Flymake Info manual を参照してください。

24.6 Emacs 下でのデバッガーの実行

GUD(Grand Unified Debugger) ライブラリーは、広範なシンボリックデバッガーにたいする Emacs のインターフェースを提供します。これは GNU デバッガー (GDB)、同様に DBX、SDB、XDB、Guile の REPL のデバッグコマンド、Perl のデバッグモード、Python デバッガーの PDB、Java デバッガーの JDB を実行することができます。

Emacs は GDB にたいする特別なインターフェースを提供します。これはデバッグされているプログラムの状態を表示する追加の Emacs ウィンドウを使用します。Section 24.6.5 [GDB Graphical Interface], page 320 を参照してください。

Emacs は、Emacs Lisp プログラムにたいするビルトインのデバッガーももっています。Section “The Lisp Debugger” in *the Emacs Lisp Reference Manual* を参照してください。

24.6.1 GUD の開始

デバッガーサブプロセスを開始する複数のコマンドがあり、それらは特定のデバッガープログラムに対応しています。

`M-x gdb` GDB をサブプロセスとして実行し、IDE-like な Emacs インターフェースを通じてやりとりをします。このコマンドに間する詳細は、Section 24.6.5 [GDB Graphical Interface], page 320 を参照してください。

`M-x gud-gdb`
GDB サブプロセスとの入出力に、GUD interaction バッファーを使用して GDB を実行します ((Section 24.6.2 [Debugger Operation], page 316 を参照してください))。そのようなバッファーがすでに存在している場合はそのバッファーに切り替え、存在しない場合はバッファーを作成して切り替えます。

ここにリストされている他のコマンドは、他のデバッガープログラムにたいして同じことを行います。

`M-x perldb`
Perl インタープリターをデバッグモードで実行します。

`M-x jdb` Java デバッガーを実行します。

`M-x pdb` Python デバッガーを実行します。

`M-x guiler`
Guile Scheme プログラムをデバッグするために、Guile REPL を実行します。

`M-x dbx` DBX デバッガーを実行します。

`M-x xdb` XDB デバッガーを実行します。

`M-x sdb` SDB デバッガーを実行します。

これらの各コマンドは、ミニバッファーを使ってデバッガーを呼び出すコマンドラインを読み取ります。ミニバッファーの初期内容は、デバッガーの標準的な実行ファイル名とオプションで、デバッグしたいと推測される実行ファイル名の場合もあります。シェルのワイルドカードと変数は、このコマンドラインでは使用できません。Emacs は '-'で始まらない最初のコマンド引数を、実行ファイル名とみなします。

Tramp は、同じリモートホスト上のデバッガーとプログラムによる、リモートデバッグ機能を提供します。詳細については、Section "Running a debugger on a remote host" in *The Tramp Manual* を参照してください。これは GDB のリモートデバッグ機能とは別の物です、なぜなら、プログラムとデバッガーは違うマシンで実行されるからです (Section "Debugging Remote Programs" in *The GNU debugger* を参照してください)。

24.6.2 デバッガーの操作

GUD interaction バッファーは、デバッガーサブプロセスにテキストコマンドを送ったり、それの出力を記録するのに使用される Emacs バッファーです。これは `M-x gud-gdb`や、前のセクションにリストされた他のコマンドで使用される、デバッガーとやりとりするための基本的なインターフェースです。`M-x gdb`コマンドは、ブレークポイント、スタックフレーム、その他のデバッガーの状態の様相を制御する、追加の特別なバッファーにより、この機能を拡張します (Section 24.6.5 [GDB Graphical Interface], page 320 を参照してください)。

GUD interaction は Shell モードの変種を使用するので、Shell モードで定義された Emacs コマンドが利用可能です (Section 31.5.3 [Shell Mode], page 460 を参照してください)。ほとんどのデバッガーコマンドにたいして補完 (Section 5.4 [Completion], page 31 を参照してください) が利用可能で、それらを繰り返すのに、通常の Shell モードのヒストリーコマンドを使うことができます。GUD interaction バッファーで使用できる特別なコマンドについては、次のセクションを参照してください。

プログラムをデバッグすると、Emacs は関連するソースファイルを Emacs バッファーに visit して、カレント実行行には左フリンジに矢印が表示されます (テキスト端末では最初の 2 列に '=>'の矢印が表示されます)。そのようなバッファーでのポイントの移動は、矢印を移動しません。これらのソースファイルの編集はできますが、行の挿入や削除により矢印の位置は失われることに注意してください。なぜなら Emacs には編集されたソース行が、デバッガーサブプロセスから報告されるどの行に対応するか、知る手立てがないからです。この情報を更新するには通常、プログラムのリコンパイルと再実行が必要です。

GUD Tooltip モードは、GUD にツールチップサポートを追加するグローバルなマイナーモードです。このモードに切り替えるには、`M-x gud-tooltip-mode`とタイプします。このモードはデフォルトで無効になっています。有効にした場合、変数、関数、マクロ (識別子として総称される) にマウスポインターを移動すると、それらの値がツールチップで表示されます (Section 18.19 [Tooltips], page 213 を参照)。値を表示したい式の上にマウスポインターを置くだけでは値が表示されない場合は、マウスでその式をドラッグしてマークし、マウスポインターをそのマークされた領域内に置いたままにすることにより、より明示的に Emacs に指示することができます。かわりにマウスをドラッグして識別子または式をマークしてから、マウスをマークした領域から離すと、式の値がツールチップに表示されます。GUD Tooltip モードは、GUD interaction バッファー、および `gud-tooltip-modes` にリストされたメジャーモードの、すべてのソースバッファーで効果があります。変数 `gud-tooltip-echo-area`が非 `nil`の場合、またはツールチップモードがオフの場合は、ツールチップではなくエコーエリアに値が表示されます。

`M-x gud-gdb`で GUD Tooltip モードを使用する場合、GDB により表示される式の値は、マクロを展開する場合があり、これはデバッグされているプログラムに副作用をもたらすかもしれません。この理由により、`gud-gdb`ではツールチップの使用は無効になっています。`M-x gdb`インターフェースを使用する場合、この問題は発生しません。なぜなら副作用を避ける特別なコードがあるからです。さらにプログラムが実行されていないときに、識別子に関連付けられたマクロの定義を表示することもできます。

24.6.3 GUD のコマンド

GUD はブレークポイントのセットとクリアー、スタックフレームの選択、プログラムのステップ実行のためのコマンドを提供します。

`C-x C-a C-b`
: ポイントのあるソース行にブレークポイントをセットします。

ソースバッファーから `C-x C-a C-b` (`gud-break`) が呼び出された場合、カレントソース行にデバッガーのブレークポイントをセットします。このコマンドは GUD を開始した後だけ利用可能です。デバッガーサブプロセスに関連付けられていないバッファーで呼び出すと、エラーをシグナルします。

以下のコマンドは、GUD interaction バッファーとグローバルの両方で利用可能ですが、キーバインドが異なります。キーが `C-c`で始まるものは GUD interaction バッファーだけで利用可能で、`C-x C-a`で始まるものはグローバルに利用可能です。コマンドのいくつかはツールバーを通じても利用可能です。また、特定のデバッガーではサポートされないものもあります。

`C-c C-l`
`C-x C-a C-l`
: GUD interaction バッファーで参照される最後のソース行を、別のウィンドウに表示します (`gud-refresh`)。

C-c C-s
C-x C-a C-s
: 次の 1 行を実行します (gud-step)。その行が関数呼び出しを含む場合、関数呼び出しに入った後に実行をストップします。

C-c C-n
C-x C-a C-n
: 次の 1 行を実行します (gud-next)。その行が関数呼び出しを含む場合、関数の中でストップせずに関数をステップオーバーします。

C-c C-i
C-x C-a C-i
: 機械語の 1 命令を実行します (gud-stepi)。

C-c C-p
C-x C-a C-p
: ポイント位置の式を評価します (gud-print)。表示したい正確な式を Emacs が表示しない場合、最初に式をリージョンとしてマークします。

C-c C-r
C-x C-a C-r
: 停止位置を指定せずに実行を継続します。プログラムは、ブレークポイントに達する、プログラム終了、またはデバッガーがチェックしているシグナルを受けとるまで実行を続けます。

C-c C-d
C-x C-a C-d
: カレントソース行にブレークポイントがある場合、ブレークポイントを削除します。GUD interaction バッファーでこのコマンドを使用する場合、プログラムが最後に停止した位置に適用されます。

C-c C-t
C-x C-a C-t
: カレントソース行に、一時的なブレークポイントをセットします (gud-tbreak)。GUD interaction バッファーでこのコマンドを使用した場合、プログラムが最後に停止した位置に適用されます。

C-c <
C-x C-a <
: 次の外側のスタックフレームを選択します (gud-up)。これは GDB コマンドの ‘up’と等価です。

C-c >
C-x C-a >
: 次の内側のスタックフレームを選択します (gud-down)。これは GDB コマンドの ‘down’と等価です。

C-c C-u
C-x C-a C-u
: カレント行まで実行を継続します (gud-until)。プログラムは、ブレークポイントに達する、プログラム終了、またはデバッガーがチェックしているシグナルを受けとる、またはカーソルがある行に到達するまで実行を続けます。

`C-c C-f`
`C-x C-a C-f`
: 選択されたフレームがリターンするか、他の理由により停止するまでプログラムを実行します (`gud-finish`)。

GDB を使用している場合、追加のキーバインディングが利用可能です:

`C-x C-a C-j`
: ソースバッファーだけで有用です。`gud-jump`はプログラムの実行箇所をカレント行に転送します。別の言い方をすると、プログラムが次に実行するのは、このコマンドを与えた位置になります。新しく実行される行が前の関数とは異なる場合、多分奇妙な結果になるので、GDB は確認を求めます。詳細は、GDB マニュアルのエントリー `jump`を参照してください。

`TAB`
: GDB の場合、シンボル名を補完します (`gud-gdb-complete-command`)。このキーは GUD interaction バッファーだけで利用可能です。

これらのコマンドは、それが意味がある場合には、数引数を繰り返し回数と解釈します。

`TAB`は補完コマンドに割り当てられているので、GDB でデバッグしているプログラムへのタブの入力には使えません。タブの入力には `C-q TAB`とタイプしてください。

24.6.4 GUD のカスタマイズ

起動時に GUD は以下のフックの 1 つを実行します:
GDB を使用している場合は `gdb-mode-hook`、
DBX を使用している場合は `dbx-mode-hook`、
SDB を使用している場合は `sdb-mode-hook`、
XDB を使用している場合は `xdb-mode-hook`、Guile REPL のデバッグには `guiler-mode-hook`、
Perl のデバッグモードを使用している場合は `perldb-mode-hook`、
PDB を使用している場合は `pdb-mode-hook`、
JDB を使用している場合は `jdb-mode-hook`を実行します。
Section 33.2.2 [Hooks], page 509 を参照してください。

Lisp マクロ `gud-def`(Section “Defining Macros” in *the Emacs Lisp Reference Manual* を参照してください) は、デバッガーに特定のコマンド文字列を送る Emacs コマンドを定義して、GUD interaction バッファーで、それにたいするキーバインドをセットアップする便利な方法を提供します:

```
(gud-def function cmdstring binding docstring)
```

これはデバッガープロセスに *cmdstring*を送る、ドキュメント文字列が *docstring*の、*function*という名前のコマンドを定義します。コマンド *function*を、任意のバッファーで使用できます。*binding*が非 `nil`の場合、`gud-def`はそのコマンドを、GUD バッファーのモードでは `C-c` *`binding`*、グローバルには `C-x C-a` *`binding`*にバインドします。

コマンド文字列 *cmdstring*には、*function*が呼び出されたときに書き込まれるデータのための、特定の ‘`%`’シーケンスを含めることができます:

‘`%f`’
: カレントソースファイルの名前です。カレントバッファーが GUD バッファーの場合、カレントソースファイルはプログラムがストップしているファイルです。

‘`%l`’
: カレントソース行の番号です。カレントバッファーが GUD バッファーの場合、カレントソース行はプログラムがストップしている行です。

‘`%e`’
: transient-mark-mode では、リージョンがアクティブの場合はリージョンのテキストです。そうでない場合、ポイント位置またはそれに隣接する位置にある C の lvalue(左辺値)、または関数呼び出し式です。

‘%a’ ポイント位置またはそれに隣接する位置にある、16 進アドレスのテキストです。

‘%p’ 呼び出された関数の数引数の 10 進数です。コマンドに数引数が指定されなかった場合、‘%p’は空文字列になります。

コマンド文字列に ‘%p’を使用しない場合、定義したコマンドは数引数を無視します。

‘%d’ カレントソースファイルのディレクトリー名です。

‘%c’ ポイントを取り囲む式から派生された、完全に記述されたされた class 名 (fully qualified class name) です (jdb のみ)。

24.6.5 GDB のグラフィカルインターフェース

コマンド M-x gdbはブレークポイント、スタックフレーム、その他のデバッグ状態の様相を制御するために特化したバッファーで、IDE-like なインターフェースで GDB を開始します。これは、たとえばマウスソースバッファーのフリンジをクリックすることにより、そこにブレークポイントをセットするなどの、マウスによりデバッグセッションを制御する追加の方法も提供します。

これらの追加機能を使わずに GUD interaction バッファーのインターフェースだけを使って GDB を実行するには、M-x gud-gdb (Section 24.6.1 [Starting GUD], page 315 を参照) を使用します。

内部的には、M-x gdbは GDB にたいしてスクリーンサイズに制限がないと告げます。正しい操作のために、デバッグセッションの間は GDB のスクリーンの高さと幅の値を変更してはいけません。

24.6.5.1 GDB のユーザーインターフェースのレイアウト

変数 gdb-many-windowsが nil(デフォルト) の場合、M-x gdbは通常 GUD interaction バッファーだけを表示します。しかし gdb-show-mainが非 nilの場合、2 つのウィンドウで開始します。その場合、1 つは GUD interaction バッファーを表示して、もう一方はデバッグするプログラムの main 関数のソースを表示します。

gdb-many-windowsが非 nilの場合、M-x gdbは以下のフレームレイアウトを表示します。

```
+--------------------------------+--------------------------------+
|   GUD interaction buffer       |   Locals/Registers buffer      |
|--------------------------------+--------------------------------+
|   Primary Source buffer        |   I/O buffer for debugged pgm  |
|--------------------------------+--------------------------------+
|   Stack buffer                 |   Breakpoints/Threads buffer   |
+--------------------------------+--------------------------------+
```

上のいずれかにもとづきウィンドウレイアウトをカスタマイズして、gdb-save-window-configurationを使用してそのレイアウトをファイルに保存できます。そのレイアウトは後から gdb-load-window-configurationを使用してロードすることで戻すことができます (内部的には Emacs はウィンドウレイアウトのかわりにウィンドウ構成という用語を使用する)。gdb-default-window-configuration-fileをカスタマイズすることで、gdb-many-windowsが使用するデフォルトレイアウトにカスタムレイアウトをセットできます。これが絶対ファイル名でなければ、GDB は gdb-window-configuration-directoryの配下でファイルを探します。gdb-window-configuration-directoryのデフォルトは user-emacs-directoryです (Section 33.4.4 [Find Init], page 533 を参照)。

ウィンドウのレイアウトを変更した場合には、M-x gdb-restore-windowsとタイプして、デフォルトのレイアウトをリストアできます。複数ウィンドウレイアウトと、GUD interaction バッファーとソースファイルだけの単純なレイアウトを切り替えるには、M-x gdb-many-windowsとタイプしてください。

ウィンドウを複雑にセットアップをしていて、gdb-many-windowsがそれを混乱させるのを望まない場合は、別のフレーム内でM-x gdbを呼び出すほうがよいでしょう。その場合は、元のフレームのウィンドウのアレンジに影響はありません。テキスト端末で作業する場合は、GDBセッションに別のフレームを使用すれば、各ウィンドウにたいするスクリーン資源がもっとも活用される可能性があるので、特に便利になり得ます。同一フレーム上でのGDB開始を選択する場合には、gdb-restore-window-configuration-after-quitを非nilにセットすることを考慮してください。その場合には、GDBのquit後に元のレイアウトがリストアされるでしょう。tなら常にリストア、if-gdb-many-windowsならgdb-many-windowsが非nilのときだけリストア、if-gdb-show-mainならgdb-show-mainが非nilのときだけリストアされます。

同じフレームまたは異なるフレームに、GDBに関連した追加のバッファーを表示するように指定できます。M-x gdb-display-*buffertype*-bufferまたはM-x gdb-frame-*buffertype*-bufferとタイプして、望むバッファーを選択します。ここで*buffertype*は‘breakpoints’や‘io’のような、該当するバッファータイプです。‘GUD’メニューの、サブメニュー‘GDB-Windows’または‘GDB-Frames’により、メニューバーから同じことができます。

デフォルトではGDBはソースファイルを表示するために最大で1つのウィンドウを使用します。gdb-max-source-window-countをカスタマイズして、より多くのウィンドウを使用することができます。gdb-display-source-buffer-actionをカスタマイズして、GDBがソースファイルを表示する方法を制御することもできます。

デバッグを終えたらC-x kでGUD interactionバッファーをkillすれば、このセッションでの関連するすべてのバッファーをkillできます。しかしEmacsでソースコードの編集とリコンパイル終えて、さらにデバッグを続けたいときは、これを行う必要はありません。実行を再開すると、GDBは自動的に新しい実行ファイルを見つけます。GUD interactionバッファーを残しておけば、シェルヒストリー、同様にGDBブレークポイントを残すことができる利点があります。最近編集したソースファイルのブレークポイントが、正しい場所にあるかチェックする必要があります。

24.6.5.2 Sourceバッファー

mouse-1 (in fringe)
: その行のカレントブレークポイントをセット、またはクリアーします(gdb-mouse-set-clear-breakpoint)。

C-mouse-1 (in fringe)
: その行のブレークポイントを有効または無効にします(gdb-mouse-toggle-breakpoint-margin)。

mouse-3 (in fringe)
: その行まで実行を継続します(gdb-mouse-until)。

C-mouse-3 (in fringe)
: その行にジャンプします(gdb-mouse-jump)。

グラフィカルなディスプレイでは、sourceバッファーのフリンジをmouse-1でクリックして、その行にブレークポイントをセットできます(Section 11.15 [Fringes], page 93を参照してください)。クリックした場所に赤いドットが表示されます。すでにそこにブレークポイントが存在する場合、クリックでそれを削除します。既存のブレークポイントをC-mouse-1でクリックすることにより、有効または無効にします。クリアーされておらず無効になったブレークポイントは、グレイのドットで示されます。

テキスト端末またはフリンジが無効な場合、有効なブレークポイントはウィンドウの左端に、‘B’という文字で示されます。無効なブレークポイントは‘b’で示されます(余白はブレークポイントがあるときだけ表示されます)。

source バッファーの左フリンジの塗りつぶされた矢印は、デバッグされているプログラムがストップした最内フレームの行を示します。中抜きの矢印はより高いレベルのフレームの現在実行されている行を示します。フリンジの矢印を mouse-1でドラッグすると、ボタンを離した行まで実行が進みます。かわりにフリンジを mouse-3でクリックすることにより、その行まで実行を進めることができます。フリンジを C-mouse-3でクリックすることにより、間にある行を実行せずに、その行にジャンプできます。このコマンドは後方へもジャンプできるので、すでに実行中のコードの実行の詳細を調べるのに便利です。

デフォルトではソースファイル名とデバッグされるプログラム内の非 ASCII 文字列はデフォルトコーディングシステムを使用してデコードされます。デバッグされるプログラムが別の文字エンコーディングを使用する等の理由により違うデコーディングを望むなら、変数 gdb-mi-decode-strings に適切なコーディングシステムをセットするか、あるいは非 ASCII 文字を未デコードの 8 進エスケープのままにするよう nilをセットしてください。

24.6.5.3 Breakpoints バッファー

GDB Breakpoints バッファーは、デバッガーセッションのブレークポイント (breakpoint)、ウォッチポイント (watchpoint)、キャッチポイント (catchpoint) を表示します。Section “Breakpoints” in *The GNU debugger* を参照してください。これは以下のコマンドを提供します。これらのコマンドのほとんどはカレントブレークポイント (ポイントのあるブレークポイント) に適用されます。

SPC カレントブレークポイントを有効または無効にします (gdb-toggle-breakpoint)。グラフィカルなディスプレイでは、これは source バッファーのフリンジのドットのカラーを変更します。ドットのカラーは、ブレークポイントが有効なときは赤、無効なときはグレーです。

D カレントブレークポイントを削除します (gdb-delete-breakpoint)。

RET カレントブレークポイントのソース行を visit します (gdb-goto-breakpoint)。

mouse-2 クリックしたブレークポイントのソース行を visit します (gdb-goto-breakpoint)。

gdb-many-windowsが非 nilの場合、GDB Breakpoints バッファーは、GDB Threads バッファーとウィンドウを共有します。一方から他方へ切り替えるには、ヘッダー行の関連するボタンを mouse-1でクリックします。gdb-show-threads-by-defaultが非 nilの場合、GDB Threads バッファーがデフォルトとして表示されます。

24.6.5.4 Threads バッファー

GDB Threads バッファーは、デバッグされているプログラムのスレッドのサマリーを表示します。Section “Debugging programs with multiple threads” in *The GNU debugger* を参照してください。スレッドを選択するには、ポイントをそこに移動して RET (gdb-select-thread) を押すか、それを mouse-2でクリックします。これにより、それに関連する source バッファーが表示され、他の GDB バッファーの内容も更新されます。

GDB Threads バッファー内に含まれる項目を選択するために、gdb-buffersグループ配下の変数をカスタマイズできます。

gdb-thread-buffer-verbose-names
‘Thread 0x4e2ab70 (LWP 1983)’のような長いスレッド名を表示します。

gdb-thread-buffer-arguments
スレッドのトップフレームの引数を表示します。

`gdb-thread-buffer-locations`
: ファイル情報またはライブラリー名を表示します。

`gdb-thread-buffer-addresses`
: thread バッファーのスレッドフレームのアドレスを表示します。

複数のスレッドの情報を同時に閲覧するには、GDB Threads バッファーの以下のコマンドを使用します。

`d` : カレント行のスレッドの disassembly バッファーを表示します (`gdb-display-disassembly-for-thread`)。

`f` : カレント行のスレッドの GDB Stack バッファーを表示します (`gdb-display-stack-for-thread`)。

`l` : カレント行のスレッドの GDB Locals バッファーを表示します (`gdb-display-locals-for-thread`)。

`r` : カレント行のスレッドの GDB Registers バッファーを表示します (`gdb-display-registers-for-thread`)。

これらのコマンドの大文字 `D`、`F`、`L`、`R`は、対応するバッファーを新しいフレームに表示します。

特定のスレッドについての情報を表示するバッファーを作成した場合、それはそのスレッドにバインドされて、プログラムをデバッグする間、情報を表示し続けます。各 GDB バッファーのモードインジケーターには、バッファーに情報が表示されているスレッドの番号が表示されます。スレッドの番号はバインドされたバッファーのバッファー名にも含まれます。

GDB Threads バッファーでは、さらに他のコマンドも利用可能で、それはプログラムの実行を制御するのに使われる GDB のモードに依存します。Section 24.6.5.8 [Multithreaded Debugging], page 325 を参照してください。

24.6.5.5 Stack バッファー

GDB Stack バッファーは、コールスタック *(call stack)* を表示します。これは、1 行がデバッガーセッションでのネストされたサブルーチン呼び出し (*stack frames:* スタックフレーム) にそれぞれ対応します。Section “Backtraces” in *The GNU debugger* を参照してください。

グラフィカルなディスプレイでは、選択されたスタックフレームは、フリンジの矢印で示されます。テキスト端末、またはフリンジが無効な場合、選択されたスタックフレームは反転して表示されます。スタックフレームを選択するには、ポイントをその行に移動して `RET` (`gdb-frames-select`) とタイプするか、それを `mouse-2`でクリックします。これを行うことにより、Locals バッファーも更新されます (次のセクションで説明します)。

各スタックフレームのフレームアドレスを表示したい場合は、変数 `gdb-stack-buffer-addresses`を非 `nil`値にカスタマイズしてください。

24.6.5.6 その他の GDB バッファー

`M-x gdb`が提供するバッファーのうち、オプションとして表示を要求できるバッファーとして、他にも以下のバッファーがあります:

Locals バッファー
: このバッファーは、カレントのスタックフレームのローカル変数の値を、簡単なデータ型で表示します (Section “Information on a frame” in *The GNU debugger* を参照)。値を編集したいときは、そこで `RET`を押すか、`mouse-2`でクリックしてください。

配列と構造体については、その型だけが表示されます。GDB 6.4 以降では、ポイント位置で RETをタイプ、または mouse-2でクリックすることにより、ローカル変数の値を調べることができます。それより前のバージョンの GDB では、型の説明 (‘[struct/union]’ または ‘[array]’) にたいして、RETまたは mouse-2を使用します。Section 24.6.5.7 [Watch Expressions], page 324 を参照してください。

Locals バッファーを表示するには M-x gdb-display-locals-bufferとタイプしてください。

I/O Buffer

デバッグ中のプログラムが標準入力や標準出力のストリームを用いてユーザーと対話していたり、あるいは無視できない量の出力を標準出力に出力する場合には、GDB との対話とプログラムの I/O を分離したいと思うかもしれません。コマンド M-x gdb-display-io-buffer を使えば、デバッグ中のプログラムからの入出力を Emacs がリダイレクトするためのバッファー用にウィンドウを表示させることができます。

Registers バッファー

このバッファーは、レジスターに保持されている値を表示します (Section “Registers” in *The GNU debugger* を参照)。このバッファーの表示を要求するコマンドは M-x gdb-display-registers-bufferです。値を編集したいときは、そのレジスターで RETを押すか、mouse-2をクリックします。GDB6.4 以降では、最近変化したレジスター値は、font-lock-warning-faceで表示されます。

Assembler バッファー

assembler バッファーは、カレントフレームをマシン語コードで表示します。矢印はカレント命令を指し、source バッファーのようにブレークポイントのセットと削除ができます。ブレークポイントのアイコンも、フリンジまたは余白に表示されます。このバッファーの表示を要求するコマンドは M-x gdb-display-disassembly-bufferです。

Memory バッファー

memory バッファーは、プログラムのメモリーセクションを調べるためのバッファーです (Section “Examining memory” in *The GNU debugger* を参照してください)。ヘッダー行の適切な箇所を mouse-1でクリックすることにより、そのバッファーが表示するメモリーの開始アドレス、またはデータアイテムの数が変化します (または Sおよび Nを使用)。ヘッダー行を mouse-3でクリックすることにより、データアイテムのフォーマット、またはユニットサイズのどちらを表示するか選択します。このバッファーの表示を要求するコマンドは M-x gdb-display-memory-buffer です。

gdb-many-windowsが非 nilの場合、breakpoints バッファーと threads バッファーがウィンドウを共有するように、locals バッファーと registers バッファーもウィンドウを共有します。一方から他方へ切り替えるには、ヘッダー行の関連するボタンを mouse-1でクリックしてください。

24.6.5.7 ウォッチ式

プログラムを停止するたびに、変数がどのように変化するか見たいときは、ポイントを変数名に移動して、ツールバーのウォッチアイコンをクリック (gud-watch) するか、C-x C-a C-wとタイプします。プレフィクス引数を指定した場合、変数名をミニバッファーで入力することができます。

各ウォッチ式は、speedbar に表示されます (Section 18.9 [Speedbar], page 204 を参照してください)。配列や、構造体、共有体のような複雑なデータ型はツリー形式で表示されます。ツリーの子ノード、および単純なデータ型では、式の名前とその値が表示され、speedbar フレームが選択された

ときは型がツールチップで表示されます。それより高いレベルでは名前、型、ポインターのアドレス値、そうでない場合は名前と型だけが表示されます。ルート式では、それらがどこで定義されているかを識別するために、ツールチップでフレームアドレスも表示されます

複雑なデータ型を展開または折り畳むには、式の左のタグを mouse-2をクリックするか SPCを押します。式の子にあたるデータの数が、変数 gdb-max-childrenの値を超える場合、Emacs は式を展開する前に確認を求めます。

複雑なウォッチ式を削除するには、speedbar のルート式にポイントを移動して、D (gdb-var-delete) とタイプしてください。

単純なデータ型の変数、または複雑なデータ型の単純な要素を編集するには、speedbar のその箇所にポイントを移動して、RET (gdb-edit-value) とタイプするか、値を mouse-2でクリックして、それを編集します。どちらの方法も、ミニバッファーを使って新しい値を読み取ります。

変数 gdb-show-changed-valuesを非 nil値 (デフォルト) にセットした場合、Emacs は最近変化した値を font-lock-warning-faceでハイライトし、スコープから外れた変数は目立たない shadowフェイスで表示します。変数がスコープから外れた場合、値を変更することはできません。

変数 gdb-delete-out-of-scopeが非 nil(デフォルト) の場合、Emacs はスコープから外れたときウォッチ式を自動的に削除します。この変数を nilにしておけば、プログラムが同じ関数に複数回再入したとき、新たにウォッチ式を作成しなくてよいので便利かもしれません。

変数 gdb-use-colon-colon-notationが非 nilの場合、Emacs は '*function*::*variable*' というフォーマットを使います。これにより同じ変数名を共有するウォッチ式を表示することができます。デフォルト値は nilです。

ウォッチ式の表示が更新されるたびに、自動的に speedbar を前に表示するには、gdb-speedbar-auto-raiseを非 nilにセットします。これは Emacs フレームを全画面表示にしてデバッグしているとき便利です。

24.6.5.8 マルチスレッドのデバッグ

GDB の *all-stop mode* では、プログラムが停止すると、すべてのスレッドの実行が停止します。同様に、プログラムを再開すると、すべてのスレッドが実行を開始します。Section "All-Stop Mode" in *The GNU debugger* を参照してください。マルチスレッド化されたいくつかのターゲットにたいして、GDB はこれを超える操作のためのモードをサポートします。これは *non-stop mode* と呼ばれ、他のスレッドが自由に実行を継続している間に、デバッガーで停止したプログラムのスレッドを調べることができます。Section "Non-Stop Mode" in *The GNU debugger* を参照してください。GDB のバージョン 7.0 以前では、non-stop mode はサポートされておらず、すべてのターゲットにたいしては機能しません。

変数 gdb-non-stop-settingは、Emacs が GDB を all-stop mode と non-stop mode のどちらで実行するかを決定します。デフォルトは tで、これは利用可能な場合は non-stop mode を使うことを意味します。値を nilに変更した場合、または non-stop mode が利用不可の場合、Emacs は GDB を all-stop mode で実行します。この変数は Emacs がデバッグセッションを開始したときに効果をもちます。値を変更した場合、アクティブなデバッグセッションを再起動する必要があります。

non-stop mode モードでスレッドが停止すると、通常 Emacs はそのスレッドに切り替えます。すでに選択したスレッドから停止した他のスレッドへの切り替えを行わないようにするには、変数 gdb-switch-when-another-stoppedを nilに変更してください。

Emacs が停止したスレッドに切り替えるかどうかの決定は、そのスレッドが停止した理由に依存します。変数 gdb-switch-reasonsをカスタマイズすることにより、スレッドの切り替えを行う停止理由を選択できます。

変数 gdb-stopped-functionsには、あるスレッドが停止したときに実行する関数を指定できます。

non-stop mode では、GUD の実行制御コマンドのための異なるモードを切り替えることができます。

Non-stop/A
: gdb-gud-control-all-threadsが t(デフォルト) の場合、中断および継続のためのコマンドはすべてのスレッドに適用されるので、gud-stop-subjobまたは gud-cont の 1 コマンドで、すべてのスレッドを停止または継続できます。少なくとも 1 つのスレッドが停止している場合、ツールバーに 'Go'ボタンが表示されます。また、少なくとも 1 つのスレッドが実行中の場合、'Stop'ボタンが表示されます。

Non-stop/T
: gdb-gud-control-all-threadsが nilの場合、カレントスレッドだけを停止または継続します。GUD ツールバーの 'Go'および 'Stop'のボタンの表示は、カレントスレッドの状態に依存します。

gdb-gud-control-all-threadsのカレント値は、ツールバーまたは 'GUD->GDB-MI'メニューで変更できます。

ステップコマンドは常にカレントスレッドに適用されます。

non-stop mode では、スレッドを選択せずにスレッドを中断または継続できます。threads バッファーで、ポイント位置のスレッドにたいして iをタイプすると中断、cで継続、sでステップ実行します。今後さらにそのようなコマンドが追加されるかもしれません。

スレッドを中断した場合、停止理由は 'signal received'になることに注意してください。この理由が gdb-switch-reasonsに含まれている場合 (デフォルトでは含まれています)、Emacs はそのスレッドに切り替えます。

24.7 Lisp 式の実行

Emacs には、Lisp のいくつかの変種のためのメジャーモードがあります。これらは他のプログラミング言語のモードと同じ編集コマンドを使用します (Chapter 23 [Programs], page 285 を参照してください)。それに加えて、Lisp 式を実行するための特別なコマンドを提供します。

Emacs Lisp モード
: Emacs Lisp のソースファイルを編集するためのモードです。このモードはカレントのトップレベルの Lisp 式を評価する C-M-xを定義します。Section 24.9 [Lisp Eval], page 329 を参照してください。

Lisp Interaction モード
: Emacs Lisp との対話的なセッションのためのモードです。このモードはポイントの前の式を評価して、その値をバッファーに挿入する C-jを定義します。Section 24.10 [Lisp Interaction], page 330 を参照してください。

Lisp モード
: Emacs Lisp ではない、他の Lisp を実行するプログラムのソースファイルを編集するためのモードです。このモードは、カレントのトップレベルの式を外部の Lisp で評価する C-M-xを定義します。Section 24.11 [External Lisp], page 330 を参照してください。

Inferior Lisp モード
: Emacs のサブプロセス (または *inferior process:* 下位プロセス) として実行される外部 Lisp と、対話的にセッションするためのモードです。

Scheme モード
: Lisp モードと同様ですが、Scheme プログラムのためのモードです。

Inferior Scheme モード
: Inferior Lisp モードと同様ですが、Scheme のためのモードです。

24.8 Emacs のための Lisp コードによるライブラリー

Emacs Lisp のコードは、慣習として`.el`で終わる名前のファイルに保存されます。このようなファイルは、自動的に Emacs Lisp モードで visit されます。

Emacs Lisp のコードは、load が速く省スペースで、実行も速いバイトコードにコンパイルできます。慣習により、コンパイルされた Emacs Lisp のコードは '`.elc`'で終わる名前の別のファイルに保存されます。たとえば、`foo.el`をコンパイルしたコードは`foo.elc`になります。Section "Byte Compilation" in *the Emacs Lisp Reference Manual* を参照してください。

Emacs Lisp コードをネイティブコード *(native code)* にコンパイルすることもできます。これは C や Fortran コンパイラーが生成するマシンコードと違いはありません。ネイティブコードはバイトコードよりもさらに高速に実行されます。ネイティブにコンパイルされた Emacs Lisp コードは、名前が '`.eln`'で終わるファイルに格納されます。Section "Native Compilation" in *the Emacs Lisp Reference Manual* を参照してください。

Emacs Lisp ファイルをロード *(load)* するには、`M-x load-file`とタイプします。このコマンドはミニバッファーを使ってファイル名を読み取り、そのファイル内容を Emacs Lisp コードとして実行します。最初にファイルを visit しておく必要はありません。このコマンドは、既存の Emacs バッファーからではなく、ディスクからファイルを直接読み込みます。

Emacs Lisp ファイルが、Emacs Lisp のロードパス *(load path:* 以下で定義*)* にインストールされている場合、`M-x load-file`ではなく `M-x load-library`とタイプしてロードできます。`M-x load-library`コマンドは、ファイル名ではなくライブラリー名 *(library name)* の入力を求めます。これは Emacs Lisp のロードパスの各ディレクトリーを検索して、そのライブラリー名にマッチするファイルを見つけようと試みます。ライブラリー名が '*foo*'の場合、ファイル名 *foo*`.elc`、*foo*`.el`、*foo*を見つけようと試みます (ネイティブコンパイルを有効にして Emacs をビルドした場合には、`load-library`は *foo*`.el`に対応する *foo*`.elc`を探す)。デフォルトの動作では、最初に見つかったファイルをロードします。このコマンドは`.el`より`.elc`、`.elc`より`.eln`を優先します。それはコンパイルされたファイルの方が、ロードと実行が速いからです。*lib*`.el`が *lib*`.elc`より新しい場合、警告を発します。この場合、誰かが`.el`を変更したもののリコンパイルを忘れたようだが、ともかく`.elc`をロードする、という警告です (この振る舞いにより、編集が終わっておらず、まだリコンパイルする準備ができていない Emacs Lisp のソースファイルを保存することができます)。しかしオプション`load-prefer-newer`を非`nil`値にセットした場合、上記の手順ではなく、Emacs は新しいファイルのバージョンをロードします。ネイティブコンパイルとともに Emacs をビルドして、*lib*`.el`に対応する '`.eln`'ファイルが見つからない場合には、*lib*`.elc`をロードしてバックグラウンドで *lib*`.el`のネイティブコンパイルを開始、コンパイルが終わるとその '`.eln`'ファイルをロードします。

Emacs Lisp プログラムは通常、`load`関数を使用して Emacs Lisp ファイルをロードします。これは`load-library`と似ていますが、より低レベルで追加の引数を指定できます。Section "How Programs Do Loading" in *the Emacs Lisp Reference Manual* を参照してください。

Emacs Lisp のロードパスは、変数`load-path`により指定されます。この変数の値は、ディレクトリー (文字列) のリストです。これらのディレクトリーは、`M-x load-library`コマンド、低レベルの`load`関数、その他の Emacs Lisp ライブラリーを探す Emacs 関数により、指定された順に検索されます。`load-path`のリストの要素には、特別な値`nil`も指定できます。これはカレントのデフォル

トディレクトリーを意味しますが、その意味するところはEmacsが`load-path`を使用する際のカレントディレクトリーに依存するはずなので、これを使うのは大抵間違っています (リストに`nil`を含めたいと思うとき、大抵の場合は、本当に望んでいるのは`M-x load-file`を使用することです)。

`load-path`のデフォルト値は、Emacs自身がLispコードを格納するディレクトリーのリストです。他のディレクトリーに独自のライブラリーがある場合、ロードパスにそのディレクトリーを追加できます。このマニュアルで説明されている他の大半の変数とは異なり、`load-path`はCustomizeインターフェース (Section 33.1 [Easy Customization], page 499 を参照してください) を通じての変更はできません。しかしinitファイルに以下のような行を記述して、ディレクトリーを追加できます (Section 33.4 [Init File], page 528 を参照してください):

```
(add-to-list 'load-path "/path/to/my/lisp/library")
```

通例ではローカルにインストールしたライブラリーは、すでに`load-path`のデフォルト値である`site-lisp`ディレクトリーか、`site-lisp`のサブディレクトリーに配置します。この方法なら`load-path`のデフォルト値を変更する必要はありません。

`load-path`と同じように、Emacsがネイティブコンパイルされた Lispコードをもつ`*.eln`ファイルを探すディレクトリーのリストは、変数`native-comp-eln-load-path`で指定します。

いくつかのコマンドは、自動ロード *(autoload)* されます。これらを実行するとき、Emacsは最初に関連するライブラリーを自動的にロードします。たとえば`M-x compile`コマンド (Section 24.1 [Compilation], page 309 を参照してください) は、自動ロードされます。これを呼び出した場合、Emacsは最初に、自動的に`compile`ライブラリーをロードします。対照的にコマンド`M-x recompile`は、自動ロードされません。そのため、このコマンドは`compile`ライブラリーをロードするまで利用できません。

自動的なロードは、自動ロードされたコマンドのドキュメントを探すとき (Section 7.2 [Name Help], page 45 を参照してください) にも発生します。それは、ドキュメントがライブラリーの他の関数や変数を参照する場合です (ライブラリーのロードにより`*Help*`バッファーのハイパーリンクが適切にセットアップされます)。この機能を無効にするには、変数`help-enable-autoload`を`nil`に変更してください。

`describe-variable`と`describe-function`での名前補完時にも、接頭辞が補完されたかにもとづいて、自動ロードが発生します。この機能を無効にするには、変数`help-enable-completion-autoload`を`nil`に変更してください。

Emacsが検索、ロード可能なディレクトリーにライブラリーを一度配置したら、スタートアップ時に利用可能にしたいと思うかもしれません。これはライブラリーがオンデマンドで自動的に利用可能になる必要がある機能を定義していて、手動でのライブラリーが不便な際に有用です。この場合にはinitファイルに適切なフォーム (スタートアップ時にライブラライーのロードが常に必要なら`load`か`require`、あるコマンドや関数の呼び出し時にEmacsがライブラリーのロードを要するなら`autoload`) を追加して確実にライブラリーがロードされるようにしてください。たとえば:

```
;; 無条件で my-shining-package.elcをロード
(require 'my-shining-package)
;; my-func呼び出し時に my-shining-package.elcをロード
(autoload 'my-func "my-shining-package")
```

`package-install` (Section 32.3 [Package Installation], page 493 を参照) を使用したパッケージのインストールでは、Emcasが探すディレクトリーにパッケージのLispファイルを配置するように配慮するとともに、上述した手動によるカスタマイズが不要となるように必要な初期化コードをinitファイルに書き込むことに注意してください。

24.9 Emacs Lisp 式の評価

Emacs Lisp モードは Emacs Lisp を編集するためのメジャーモードです。これのモードコマンドは `M-x emacs-lisp-mode`です。

Emacs は、Emacs Lisp 式を評価するためのコマンドをいくつか提供します。記述している Emacs Lisp コードをテストするために、これらのコマンドを Emacs Lisp モードで使用できます。たとえば、関数を書き換えた後、以降の関数呼び出しでそれを有効にするために、関数定義を評価します。これらのコマンドはグローバルに利用可能で、Emacs Lisp モード以外でも使用できます。

`M-:` 1 つの Emacs Lisp 式をミニバッファーで読み取り、それを評価して、値をエコーエリアに出力します (`eval-expression`)。

`C-x C-e` ポイントの前の Emacs Lisp 式を評価して、値をエコーエリアに出力します (`eval-last-sexp`)。

`C-M-x` (Emacs Lisp モード)

`M-x eval-defun`
: ポイントの後またはポイントを含む defun を評価して、値をエコーエリアに出力します (`eval-defun`)。

`M-x eval-region`
: リージョンのすべての Emacs Lisp 式を評価します。

`M-x eval-buffer`
: バッファーのすべての Emacs Lisp 式を評価します。

`M-:` (`eval-expression`) は、ミニバッファーを使って式を読み取り、それを評価します (式を評価する前に、カレントバッファーは、式をタイプするためのミニバッファーではなく、`M-:`をタイプしたときカレントだったバッファーに切り替わります)。

コマンド `C-x C-e` (`eval-last-sexp`) は、そのバッファーのポイントの前にある Emacs Lisp 式を評価して、その値をエコーエリアに表示します。評価した結果が整数のときは、他のフォーマット (8 進、16 進、`eval-expression-print-maximum-character`の制限を超えなければ文字) とともに値を表示します。

`M-:`および `C-x C-e`にプレフィクス引数を与えた場合、値をエコーエリアに表示するのではなく、カレントバッファーのポイント位置に値を挿入します。プレフィクス引数が 0 の場合、整数出力は他のフォーマット (8 進、16 進、文字) と一緒に挿入されます。プレフィクス引数は、`eval-expression-print-level`および `eval-expression-print-length`にしたがった出力の省略も防ぎます (以下参照)。同様にプレフィックス引数`-1`は、`eval-expression-print-length`の効果をオーバーライドします。

`C-x C-e` (`eval-last-sexp`) は特に `defvar`式を特別に扱います。通常、`defvar`式を評価しても、それが定義する変数がすでに値をもっている場合は、何も起こりません。しかし、このコマンドは無条件に `defvar`で指定された初期値に変数をリセットします。これは Emacs Lisp プログラムをデバッグするとき便利です。式 `defcustom`および `defface`も同様に扱われます。このセクションで説明している `eval-defun`以外のコマンドは、この特別な機能をもっていません。

Emacs Lisp モードでは、`eval-defun`コマンドは `C-M-x`にバインドされています。これはポイント以降のトップレベルの Lisp 式を評価して、値をエコーエリアにプリントします。このコンテキストにおけるトップレベル式は “defun” を意味しますが、実際の `defun` (関数定義) である必要はありません。

このコマンドは `eval-last-sexp`が行うのと同じ方法で、`defvar`/`defcustom`/`defface`を処理します。

プレフィクス引数を指定すると、C-M-xは Emacs Lisp デバッガーの Edebug のために関数定義をインストルメント (instrument: 処置) します。Section “Instrumenting” in *the Emacs Lisp Reference Manual* を参照してください。

コマンド M-x eval-regionは、リージョンのテキストを 1 つ以上の Lisp 式として解析して、それらを 1 つずつ評価します。M-x eval-bufferも同様ですが、これはバッファー全体を評価します。

オプション eval-expression-print-levelおよび eval-expression-print-lengthは、評価コマンドが結果を出力する前に省略する、リストの最大の深さと長さを制御します。eval-expressionまたは eval-last-sexpにプレフィクス引数 0 を指定すると、リストはすべて出力されます。eval-expression-debug-on-errorは、これらのコマンドが使用されるとき、評価エラーによりデバッガーを呼び出すかを制御します、デフォルトは tです。eval-expression-print-maximum-characterは、文字として表示される最大の整数を超える値を抑制します。

24.10 Lisp Interaction バッファー

Emacs を開始したとき、*scratch*という名前のバッファーが含まれます、これは Emacs Lisp 式の対話的な評価を提供します。このバッファーのメジャーモードは、Lisp Interaction モードです。M-x lisp-interaction-modeとタイプしても、Lisp Interaction モードを有効にできます。

*scratch*バッファーを kill してしまった場合には、コマンド M-x scratch-bufferで再作成できます。

*scratch*バッファー、およびその他の Lisp Interaction モードのバッファーでは、C-j (eval-print-last-sexp) はポイントの前の Lisp 式を評価して、値をポイント位置に挿入します。したがってバッファーに式をタイプするたび、その後ろで C-jをタイプすることにより、そのバッファーは式の評価とその値を記録した写しになります。その他すべての Lisp Interaction モードのコマンドは、Emacs Lisp モードと同じです。

起動時には、*scratch*バッファーは、それが何かを説明する Lisp コメント形式の短いメッセージを含んでいます。このメッセージは変数 initial-scratch-messageにより制御され、値にはドキュメント文字列または nil(メッセージを抑止するという意味) を指定します。

Emacs Lisp 式を対話的に評価する別の方法は、Inferior Emacs Lisp モードを使う方法です。これは Emacs Lisp 式の評価に Shell モード (Section 31.5.3 [Shell Mode], page 460 を参照してください) に似たインターフェースを提供します。M-x ielmとタイプすることにより、このモードを使用する*ielm*というバッファーが作成されます。詳細は、コマンドのドキュメントを参照してください。

24.11 外部 Lisp の実行

Lisp モードは、Common Lisp のような一般用途のための Lisp 方言で記述されたプログラムを記述するためのメジャーモードです。これのモードコマンドは M-x lisp-modeです。Emacs は名前が.l、.lsp、.lispで終わるファイルにたいして、自動的に Lisp モードを使用します。

外部 Lisp セッションを Emacs のサブプロセス、または下位プロセス *(inferior process)* として実行して、式を評価するために渡すことができます。外部 Lisp セッションを開始するには、M-x run-lispとタイプします。これは lispという名前のプログラムを実行して、*inferior-lisp*という名前の Emacs バッファーを通じて入出力を行うようにセットアップします。M-x run-lispで実行される Lisp プログラムの名前を変更するには、変数 inferior-lisp-programを変更してください。

*lisp*バッファーのためのメジャーモードは Inferior Lisp モードで、これは Lisp モードの性質と Shell モード (Section 31.5.3 [Shell Mode], page 460 を参照してください) の性質をあわせ持っ

ています。Lisp セッションに入力を送るには、`*lisp*`バッファーの最後に移動して、入力をタイプしてから `RET`をタイプします。Lisp セッションからの端末出力は、自動的にそのバッファーに挿入されます。

Lisp プログラムを Lisp モードで編集する場合、`C-M-x` (`lisp-eval-defun`) とタイプして、Lisp モードのバッファーから、`M-x run-lisp`で開始した Lisp セッションに式を送ることができます。送信される式はポイント位置、またはポイントの後ろのトップレベルの Lisp 式です。結果となる値は通常どおり、`*inferior-lisp*`バッファーに送られます。Lisp モードでの `C-M-x`の効果は、それが評価される Emacs に送られるのではなく、異なる Lisp 環境に送られる点を除けば、Emacs Lisp モード (Section 24.9 [Lisp Eval], page 329 を参照してください) での効果とよく似ていることに注意してください。

Scheme コードを編集して、式を Scheme サブプロセスに送る機能は、よく似ています。Scheme ソースファイルは、Scheme モードで編集されます。このモードは `M-x scheme-mode`で明示的に有効にできます。`M-x run-scheme`とタイプすることにより、Scheme セッションを開始し、`C-M-x`とタイプして式の送ることができます (Scheme と対話するためのバッファーは、`*scheme*`という名前です)。

25 大きなプログラムの保守

このチャプターでは、中規模から大規模のプログラムやパッケージを保守するための Emacs の機能を説明します。これらの機能には、以下が含まれます:

- ソースファイルへの変更履歴を記録するバージョンコントロールシステム (VCS: Version Control Systems) にたいする、統一されたインターフェースのサポート。
- プログラミングプロジェクトを扱うためのコマンド。
- プログラムの変更にたいして日時順のログを提供する、`ChangeLog`の保守に特化したモード。
- シンボル定義 (“識別子 (identifiers)” とも言います) を表示する一連のコマンドである、Xref。
- Emacs 自身の IDE である、EDE。
- バグリファレンスのハイライト、および issue tracker(問題追跡システム) 内で参照されるバグレポートの visit。

Lisp の大きなプログラムをメンテナンスしている場合は、ここで説明している機能に加え、ERT(Emacs Lisp Regression Testing) ライブラリーも便利だと思うかもしれません (Section “ERT” in *Emacs Lisp Regression Testing* を参照してください)。

25.1 バージョンコントロール

バージョンコントロールシステム *(version control system)* は、ソースファイルの複数のバージョンを記録したり、それらのバージョンの作成日時などや、誰が作成したか、何が変更されたかの説明などを記録できるプログラムのことです。

Emacs のバージョンコントロールのためのインターフェースは、*VC* と呼ばれます。VC コマンドは、複数の異なるバージョンコントロールシステムで機能します。現在のところ、Bazaar、CVS、Git、Mercurial、Monotone、RCS、SRC、SCCS/CSSC、Subversion がサポートされます。これらの中で、GNU プロジェクトのディストリビューションは CVS、RCS、Bazaar です。

バージョン管理システムで生成されたファイルを visit すると、自動的に VC が有効になります。VC を無効にするには、カスタマイズ可能な変数 `vc-handled-backends`を `nil`をセットします (Section “Customizing VC” in *Specialized Emacs Features* を参照してください)。

カレントバッファーで visit されているファイルにたいする VC 状態の情報を更新するには、コマンド `vc-refresh-state`を使用します。Emacs の外 (たとえばシェルプロンプト) でバージョンコントロールコマンドを実行したときや、そのバッファーのファイルを別のバージョンコントロールシステム下に置いたとき、バージョンコントロールからそのファイルを完全に削除したときに、このコマンドは有用です。

VCS の制御下にあるファイルを含んだディレクトリーを表示している Dired バッファー (Chapter 27 [Dired], page 380 を参照) では、VC も自動的に有効になります。そのような Dired では、このセクションで説明するすべての VC コマンドを呼び出すことができます。Dired でマークされたすべてのファイル (Section 27.6 [Marks vs Flags], page 385 を参照) はカレントのファイルセットに属しているものとみなされて、そのファイルセットのファイルにたいして VC コマンドが処理を行うのです。これによりファイルの VC 状態 (VC state) に関わらず、含めたい任意のファイルから VC ファイルセットを構築することができます (Dired バッファーから VC コマンドを呼び出した際にマークされたファイルがなければ、バッファーのカレント行に表示されているファイルがそのファイルセットで唯一のファイルとみなされる)。

25.1.1 バージョンコントロールの紹介

VC は、Emacs からのバージョンコントロールシステムの使用、およびバージョンコントロールの操作と編集をスムーズに統合します。VC は、多くのバージョンコントロールシステムでの一般的な操作にたいする、統一されたインターフェースを提供します。

レポジトリ設定の変更など、非一般的または複雑なバージョンコントロールの操作は、VC ではサポートされません。そのようなタスクは、VC の外 (たとえばコマンドライン) で処理する必要があります。

このセクションは、バージョンコントロールの一般的な概観を提供し、VC がサポートするバージョンコントロールを説明します。すでに使いたいバージョンコントロールシステムをよく知っている場合、このセクションはスキップできます。

25.1.1.1 問題の背景を理解する

バージョンコントロールシステムは、3 つの重要な能力を提供します。

- *Reversibility(*可逆性*)*: ある変更が間違い、または間違えた考えにもとづくものだと気づいたときに、前の状態に戻す能力です。
- *Concurrency(*並列性*)*: 多くの人が同じファイルコレクションを変更するとき、変更の衝突を検知して解決する能力です。
- *History(*履歴*)*: それを変更した背後の意図を説明するコメントなど、履歴データをデータに付す能力です。1 人で作業するプログラマーでさえ、変更履歴は記憶を助けるのに重要です。複数人のプロジェクトでは、開発者間のコミュニケーション形式として、とても重要です。

25.1.1.2 サポートされるバージョンコントロールシステム

VC は現在のところ、多くの異なるバージョンコントロールで機能し、それらをバックエンド *(back ends)* として参照します:

- Git は、最初に Linus Torvalds により Linux(彼の kernel) の開発をサポートするために考案されました。VC は多くの Git 操作をサポートしますが、その他のレポジトリの同期などはコマンドラインを使わなければなりません。
- CVS は、フリーなバージョンコントロールシステムであり、2008 年 y 頃まではフリーソフトウェアプロジェクトの大半で使われていました。それ以降は、新しいシステムに徐々に置き換えらました。CVS ではローカルまたはネットワーク越しの、複数ユーザーによる並列開発が可能です。新しいシステムとは異なり、アトミックなコミットとファイルの移動・リネームにたいするサポートがありません。VC は CVS 下での基本的な編集操作をサポートします。
- Subversion(svn) は、CVS と同じようにデザインされた、フリーなバージョンコントロールシステムですが、CVS のもつ問題はありません (たとえば、これはファイルセットのアトミックなコミット、ディレクトリーのバージョニング、シンボリックリンク、メタデータ、リネーム、コピー、削除をサポートします)。
- SCCS は、これまでに構築された一番最初のバージョンコントロールシステムで、ずっと以前に、もっと進んだものに取って代わられました。SCCS にない特定の機能のための VC コンポーネントは、それら自身により実装されています。複数ブランチなど、その他の VC 機能は単に利用不可です。SCCS はフリーではないので、これを避けることを推奨します。
- CSSC は SCCS のフリーな置き換えです。何らかの理由により、もっと新しい、より良いデザインのバージョンコントロールシステムを使うことができないときだけ、CSSC を使うべきです。

- RCS は、VC が最初に構築された頃の、フリーなバージョンコントロールシステムです。これは比較的に初期のものです。これはネットワークを超えて使うことはできず、ファイルに個別のレベルで機能します。RCS でできるほとんどのことは、VC を通じて行うことができます。
- Mercurial(hg) は、Git に酷似した分散化されたバージョンコントロールシステムです。VC は、レポジトリの同期操作を除いて、ほとんどの Mercurial コマンドをサポートします。
- Bazaar(bzr) は、レポジトリベースと分散化されたバージョニングの両方をサポートする、分散化されたバージョンコントロールシステムです。VC は Bazaar 下でのほとんどの基本的な編集操作をサポートします。
- SRC (src) は RCS の再実装で、1 人による単一ファイルのプロジェクトために特化してデザインされたバージョンコントロールシステムです。これは 1 つのディレクトリーの中に、独立したバージョンコントロール履歴をもつ複数ファイルを許容します。したがって、これは小さなドキュメントやスクリプト、ドットファイルを保守するのに適しています。これは RCS のリビジョンストレージを使用するので、非 lock 操作 (lockless operation) や、シーケンシャルな整数リビジョン番号をもつ、現代的なユーザーインターフェースを提供します。

25.1.1.3 バージョンコントロールの概念

ファイルがバージョンコントロールの配下にある時、それがバージョンコントロールシステムに登録されている *(registered)* と言います。バージョンコントロールシステムは、レポジトリ *(repository)* をもちます。これはファイルの現在の状態、および古いバージョンから現在のバージョンを再構成するのに充分な変更履歴の、両方を保存します。レポジトリには、各ファイルに行われた変更の説明であるログエントリー *(log entries)* など、その他の情報も含まれます。

実際に編集する、バージョンコントロールされたファイルのコピーのことを、作業ファイル *(work file)* と呼びます。作業ファイルは、通常のファイルを同じように変更できます。一連の変更を終えた後、その変更をコミット *(commit)* またはチェックイン *(check in)* します。これにより、その変更はログエントリーの記述とともに、レポジトリに記録されます。

作業ファイルのディレクトリツリーを、作業ツリー *(working tree)* と呼びます。

commit するたびに、レポジトリに新しいリビジョン *(revision)* が作成されます。バージョンコントロールシステムは、過去のすべてのリビジョンと、各リビジョンで行われた変更を記録します。各リビジョンには、リビジョン *ID(revision ID)* により名前がつけられます。リビジョン ID のフォーマットは、バージョンコントロールシステムに依存します。もっとも簡単なケースでは、リビジョン ID は単なる整数です。

これらの基本的な概念を超えるにつれ、各バージョンコントロールシステムの違いの、3 つの様相を理解する必要がでてくるでしょう。以降の 3 つのセクションで説明するように、各バージョンコントロールシステムには、ロックベース (lock-based) とマージベース (merge-based)、ファイルベース (file-based) と変更セットベース (changeset-based)、集中型 (centralized) と分散型 (decentralized) の違いがあります。VC はこれらすべてのモードの操作を処理しますが、それらの違いを隠蔽することはできません。

25.1.1.4 バージョンコントロールにおけるマージベースとロックベース

バージョンコントロールシステムは通常、同じファイルを変更したい複数ユーザーを調整するために、何らかのメカニズムをもちます。これを行うには 2 つの方法— マージとロック — があります。

マージを使うバージョンコントロールシステムでは、各ユーザーはいつでも作業ファイルを変更します。バージョンコントロールシステムは、コミットされていない変更を含むユーザーの作業ファイルを、他のユーザーによりコミットされた最新の変更とマージします。

古いバージョンコントロールシステムは、かわりにロック *(locking)* を使います。この場合、作業ファイルは通常は読み取り専用です。ファイルを編集するには、それをロックすることにより書き込み可能にできるか、バージョンコントロールシステムに尋ねます。ある時点で、そのファイルをロックできるユーザーは 1 人だけです。この手順は、通常のファイルの同時編集を Emacs が検知する方法と類似しているようですが、異なります (Section 15.3.4 [Interlocking], page 155 を参照してください)。変更をコミットすると、ファイルはアンロック (unlocks) され、作業ファイルは再び読み取り専用になります。他のユーザーは、変更するためにそのファイルをロックすることができます。

ロックおよびマージの両方のシステムは、複数ユーザーが同じときに同じファイルの変更を試みたときに問題が発生し得ます。ロックを使うシステムには、ロックの衝突 *(lock conflicts)* があります。あるユーザーはファイルのチェックアウトを試みますが、それがすでにロックされている場合はロックできません。マージを使うシステムには、マージの衝突 *(merge conflicts)* があります。これはファイルに行った変更をコミットするとき、それが後からチェックアウトした他の誰かによる変更のコミットと衝突するときに発生します。どちらの衝突も、人間の判断と意思疎通により解決する必要があります。経験から、開発者に取っての利便性と、実際に発生する衝突の重大性と数を最小にするという両方の点で、マージはロックに優ります。

SCCS は常にロックを使います。RCS はデフォルトではロックベースですが、マージスタイルで処理するように指示できます。CVS と Subversion はデフォルトではマージベースですが、ロックモードで処理するように指示できます。Git、Mercurial のような分散型のバージョンコントロールシステムは、マージベースだけです。

VC はロックとマージの両方のバージョンコントロールをサポートします。“commit(コミット)” と “update(更新)” という用語は、新しいバージョンコントロールシステムで使用されます。古いロックベースのシステムでは、“check in(チェックイン)” と “check out(チェックアウト)” という用語が使用されます。VC はこれらの違いをできる限り隠蔽します。

25.1.1.5 バージョンコントロールに置ける変更セットベースとファイルベース

SCCS、RCS、CVS、およびその他の初期のバージョンコントロールシステム (SRC も含む) では、バージョンコントロールの操作はファイルベース *(file-based)* です。各ファイルは、他のすべてのファイルとは別に、ファイル自身のコメントとリビジョン履歴をもちます。Subversion で始まる新しいシステムは、変更セットベース *(changeset-based)* です。コミットは複数ファイルにたいする変更を含むときがあり、一連の変更全体を 1 つの単位として扱います。変更にたいするコメントは 1 つのファイルではなく、変更セットに属します。

変更セットベースのバージョンコントロールは、ファイルベースのバージョンコントロールより、柔軟で強力です。通常、複数ファイルの変更を元に戻す必要がある時、それを簡単に識別してすべてを削除できます。

25.1.1.6 レポジトリにおける分散型と集中型

初期のバージョンコントロールシステムは、集中型 *(centralized)* モデルでデザインされていて、各プロジェクトはすべての開発者が使用するただ 1 つのレポジトリをもちます。SCCS、RCS、CVS、Subversion、SRC はこの種のモデルを共有します。このモデルの欠点の 1 つは、レポジトリが信頼性と効率の要衝となることです。

GNU Arch は、分配型 *(distributed)* または分散型 *(decentralized)* のバージョンコントロールの先駆で、後に Git、Mercurial、Bazaar で実装されました。プロジェクトは複数の異なるレポジトリをもつことができ、これらのシステムはレポジトリ間で変更履歴を調停する、ある種のスーパーマージをサポートします。開発者ごとに 1 つのレポジトリがあり、レポジトリのマージがコミット操作を代行します。

VC は個人の作業ファイルと、レポジトリとの間で行われる通信を管理する手助けをします。VC は、レポジトリが唯一のマスターなのか、それともネットワークのピアーのレポジトリの 1 つなのかは関知しません。

25.1.1.7 ログファイルのタイプ

バージョンコントロールシステムを使うプロジェクトは、変更にたいする 2 つのタイプのログをもつことができます。1 つはバージョンコントロールシステムで保守されるログです。変更をコミットするたびに、変更にたいするログエントリー *(log entry)* を入力します (Section 25.1.4 [Log Buffer], page 339 を参照してください)。これはバージョンコントロールログ *(version control log)* と呼ばれます。

もう 1 つの種類のログは、`ChangeLog`です (Section 25.3 [Change Log], page 355 を参照してください)。これはプログラムの大きな部分 (通常は 1 つのディレクトリーと、そのサブディレクトリー) にたいする変更の記録を、年代順に記録します。小さなプログラムは、1 つの `ChangeLog`を使用するでしょう。大きなプログラムは、主要なディレクトリーごとに `ChangeLog`をもつかもしれません。Section 25.3 [Change Log], page 355 を参照してください。プログラマーは、バージョンコントロールシステムのずっと前から ChangeLog を使ってきました。

変更セットベースのバージョンシステムは、通常システム全体にたいして変更セットベースの修正ログを保守します。これは ChangeLog を冗長なものにします。ChangeLog を残す利点の 1 つは、他のディレクトリーとは別に、1 つのディレクトリーの履歴トランザクションが見れるのは便利なときがあるからです。他の利点として、多くのバージョンコントロールシステムはコミットログを特定できないからです。

バージョンコントロールで管理されるプロジェクトは、バージョンコントロールログだけを使用するか、両方の種類のログを使用します。あるファイルにたいしては 1 つのログを使い、他のファイルにたいしては別の方法を使うこともできます。プロジェクトごとに、したがうべきポリシーがあります。

両方を使うというポリシーの場合は通常、変更にたいして 1 度だけエントリーを書いて、それを両方のログに書き込みたいと思うでしょう。`ChangeLog`にエントリーを記述して、変更をコミットするときに `C-c C-a`でログバッファーにコピーできます。または変更をコミットするときログバッファーにエントリーを記述して、(`C-c C-w`の助けを借りて) 後で `C-x v a`コマンドでそれを `ChangeLog`にコピーできます (Section “Change Logs and VC” in *Specialized Emacs Features* を参照してください)。

25.1.2 バージョンコントロールとモードライン

バージョンコントロール配下のファイルを visit したとき、Emacs はモードラインにそれを示します。たとえば ‘`Bzr-1223`’と表示された場合、そのファイルに Bazaar が使用され、カレントのリビジョン ID が 1223 であることを示します。

バックエンド名とリビジョン ID の間の文字は、作業ファイルのバージョンコントロール状態 *(version control status)* を示します。マージベースのバージョンコントロールシステムでは、文字 ‘`-`’は作業ファイルが変更されていないことを示し、文字 ‘`:`’は作業ファイルが変更されていることを示します。文字 ‘`!`’は最新のマージ処理の結果により、そのファイルに衝突が含まれていることを示すか (Section 25.1.11.3 [Merging], page 351 を参照してください)、そのファイルがバージョンコントロールから削除されたことを示します。最後に、文字 ‘`?`’は、そのファイルがバージョンコントロールの配下にあるが、作業ツリーにないことを示します。

ロックベースのシステムでは、‘`-`’はファイルがロックされていないことを示し、‘`:`’はファイルが他のユーザー (たとえば ‘`jim`’) にロックされていることを示し、‘`RCS:jim:1.3`’のように表示されます。‘`@`’は、そのファイルがローカルに追加されたが、まだマスターレポジトリにコミットされていないことを示します。

グラフィカルなディスプレイでは、モードラインのインジケーターの上にマウスを移動すると、ツールチップが表示され、それにはバージョンコントロール状態の、より多くの説明が表示されます。インジケーターを `mouse-1`でクリックすると、メニューバーの '`Tools / Version Control`'と同じ VC コマンドがのメニューがポップアップします。

Auto Revert モード (Section 15.4 [Reverting], page 157 を参照してください) がバージョンコントロール配下のバッファーをリバートするとき、これはモードラインのバージョンコントロール情報を更新します。しかし、Auto Revert モードは、作業ファイルの変更をともなわない、カレントの Emacs セッションの外でのバージョンコントロール状態の変化の情報は、正しく更新しないかもしれません。`auto-revert-check-vc-info`を `t`にセットすると、Auto Revert モードは、作業ファイル自身を変更していなくても、`auto-revert-interval`秒ごとにバージョンコントロール状態の情報を更新します。CPU 使用率はバージョンコントロールシステムに依存しますが、通常はそれほど高価な処理ではありません。

25.1.3 バージョンコントロール下での基本的な編集

ほとんどの VC コマンドは、*VC* ファイルセット *(VC filesets)* にたいして処理を行います。VC ファイルセットは、VC 操作が機能する 1 つ以上のファイルのコレクションです。バージョンコントロールされたファイルを visit しているバッファーで VC コマンドをタイプすると、VC ファイルセットは単にそのファイル 1 つだけになります。VC Directory バッファーで、いくつかのファイルをマークしているときに VC コマンドをタイプしたときは、VC ファイルセットはマークされたファイルです (Section 25.1.10 [VC Directory Mode], page 346 を参照)。Dired バッファーから VC コマンドを呼び出した際にも、マークされたファイル (Section 27.6 [Marks vs Flags], page 385 を参照) から VC ファイルセットが構成されます (マークされたファイルがなければカレント行に表示されているファイルがデフォルト)。

現代的な変更セットベースのバージョンコントロールシステム (Section 25.1.1.5 [VCS Changesets], page 335 を参照してください) では、VC コマンドは複数ファイルからなる VC ファイルセットを 1 つのグループとして扱います。たとえば複数ファイルからなる VC ファイルセットをコミットすると、それらすべてのファイルにたいする変更を含む、1 つのリビジョンが生成されます。CVS のような古いファイルベースのバージョンコントロールシステムでは、複数ファイルからなる VC ファイルセットの各ファイルは個別に処理されます。たとえば、コミットにより、変更されたファイル後とに 1 つのリビジョンが生成されます。

`C-x v v`　カレント VC ファイルセットにたいして、次の適切なバージョンコントロール操作を処理します。

重要な VC コマンド `C-x v v` (`vc-next-action`) は多目的なもので、カレントの VC ファイルセットにたいして、もっとも適切な操作を処理します。このコマンドは、それをバージョンコントロールシステムに登録するか、コミットするか、ロックを外すか、変更をマージします。正確な動作の詳細は、以下のサブセクションで説明します。`C-x v v`は、ファイルを visit しているバッファー、Dired バッファー、または VC Directory バッファーのいずれかで使うことができます。後者の 2 つのバッファーにおいては、マークされたファイルから構成されるファイルセットにたいして処理を行います。

VC ファイルセットは、ファイルを閲覧したり visit するためのグループ機能として使用される、名前つきファイルセット (named filesets) とは別の物だということに注意してください (Section 15.20 [Filesets], page 174 を参照してください)。名前つきファイルセットとは異なり、VC ファイルセットは名前をもたず、セッションをまたいで持続しません。

25.1.3.1 マージでの基本的なバージョンコントロール

マージベースのバージョンコントロールシステム (たとえばもっとも現代的な方法の 1 つ。Section 25.1.1.4 [VCS Merging], page 334 を参照してください) では、`C-x v v`は以下のことを行います:

- VC ファイルセットに 1 つ以上のファイルがあり、そのファイルがバージョンコントロール状態と矛盾する場合、エラーをシグナルします (しかしファイルセットには、新しく追加されたファイルと、変更されたファイルを含むことができることに注意してください。Section 25.1.5 [Registering], page 340 を参照してください)。
- VC ファイルセットに、バージョンコントロールシステムに登録されたファイルがない場合、VC ファイルセットを登録 (たとえばバージョンコントロールの配下に置く) します。Section 25.1.5 [Registering], page 340 を参照してください。Emacs が登録すべきシステムを見つけられない場合、レポジトリタイプの入力を求め、新しいレポジトリを作成して、VC ファイルセットをそれに登録します。
- VC ファイルセットの、すべての作業ファイルが変更されていない場合は、何もしません。
- VC ファイルセットの各作業ファイルが変更されている場合、変更をコミットします。これを行うには、Emacs が`*vc-log*`バッファーをポップアップするので、新しいリビジョンのログエントリーをタイプしてから、`C-c C-c`でコミットします。Section 25.1.4 [Log Buffer], page 339 を参照してください。

  共有レポジトリにコミットする場合、最後に更新したときからレポジトリが変更されているときは、コミットが失敗するでしょう。このような場合、再試行する前に更新をしなければなりません。分散型のバージョンコントロールシステムでは、`C-x v +`(Section 25.1.11.2 [Pulling / Pushing], page 350 を参照してください)、または`C-x v m`を使用します (Section 25.1.11.3 [Merging], page 351 を参照してください)。集中型のバージョンコントロールシステムでは、レポジトリに変更をマージするために、再度`C-x v v`とタイプしてください。
- 最後に、集中型のバージョンコントロールシステムでは、VC ファイルセットの各ファイルが最新かチェックします。レポジトリで変更されたファイルがある場合、更新を提案します。

これらのルールは、変更はレポジトリから自動的にマージされない点をのぞき、RCS を非ロックモードで使用している場合も適用されます。あなたがファイルの編集を始めた後に、他のユーザーが同じファイルをコミットした場合、何の情報も与えられません。あなたのリビジョンをコミットしたとき、他のユーザーの変更は失われます (しかし、それはレポジトリには残るので、決定的に失われるわけではありません)。したがって、変更をコミットする前に、カレントリビジョンが変更されていないことを調べなければなりません。それに加えて、このモードでも RCS でロックすることが可能です。変更されていないファイルでの`C-x v v`は、RCS の通常のロックモードのようにそのファイルをロックします。(Section 25.1.3.2 [VC With A Locking VCS], page 338 を参照してください)。

25.1.3.2 ロックでの基本的なバージョンコントロール

ロックベースのバージョンコントロールシステム (SCCS や RCS のデフォルトモードなど) では、`C-x v v`は以下のことを行います:

- VC ファイルセットに 1 つ以上のファイルがあり、そのファイルがバージョンコントロール状態と矛盾する場合、エラーをシグナルします。
- VC ファイルセットに、バージョンコントロールシステムに登録されたファイルがない場合、VC ファイルセットを登録します。Section 25.1.5 [Registering], page 340 を参照してください。Emacs が登録すべきシステムを見つけられない場合、レポジトリタイプの入力を求め、新しいレポジトリを作成して、VC ファイルセットをそれに登録します。

- 各ファイルが登録されていてロックされていなければ、それを書き込み可能にするためにロックして、編集を開始できるようにします。
- 各ファイルのロックを獲得していて、それらが変更を含む場合、その変更をコミットします。これを行うことにより、Emacs は*vc-log*バッファーをポップアップするので、新しいリビジョンのログエントリーをタイプしてから、C-c C-c でコミットします (Section 25.1.4 [Log Buffer], page 339 を参照してください)。
- 各ファイルのロックをすでに獲得していて、変更がない場合、ファイルを再び読み取り専用にするために、ロックを開放します。
- 各ファイルが他のユーザーによりロックされている場合、ロックを横取りする (steal the lock) か、確認を求めます。これに yes と応えると、あなたがロックを獲得して、その前にファイルをロックしていたユーザーに、警告メッセージが送られます。

このルールは、CVS がロックの横取りをサポートしない点をのぞき、CVS をロックモードで使用している場合も適用されます。

25.1.3.3 C-x v vの高度な制御

vc-next-actionにプレフィクス引数を与えた場合 (C-u C-x v v)、論理的に次のバージョンコントロール操作を処理する点に変わりはありませんが、どのように操作を行うかを正確に指定するために、追加の引数を指定できるようになります。

- バージョンコントロールシステムの名前を指定できます。これは、ファイルセットが複数のバージョンコントロールシステムで管理されていて、Emacs が正しいものを検知するのに失敗するとき便利です。
- そうでない場合で、CVS、RCS、SRC を使用しているときは、リビジョン ID を指定できます。
  ファイルセットが変更 (またはロック) されている場合、Emacs は指定したリビジョン ID でそれをコミットします。適切なリビジョン ID を与えることにより、新しいブランチを作成できます (Section 25.1.11 [Branches], page 349 を参照してください)。
  ファイルセットが変更 (またはロック) されていない場合、これは指定したリビジョンを作業ツリーにチェックインします。リビジョン ID またはブランチ ID を与えることにより、他のブランチのリビジョンを指定できます (Section 25.1.11.1 [Switching Branches], page 349 を参照してください)。空の引数 (例: C-u C-x v v RET) は、カレントブランチの最新のリビジョン (head) をチェックアウトします。
  これは、分散型のバージョンコントロールシステムでは単に無視されます。これらのシステムでは、独自のリビジョン ID は指定できず、個別のファイルを “チェックアウト” するという概念も使いません。

25.1.4 Log Entry バッファーの機能

VC に変更のコミットを指示したとき、*vc-log*という名前のバッファーがポップアップします。このバッファーには、行った変更を説明するログエントリー *(log entry)* を書き込みます。書き込んだら C-c C-c (log-edit-done) とタイプして, そのバッファーを抜けて、ログエントリーとともに変更をコミットします。

*vc-log*バッファーのためのメジャーモードは Log Edit モードで、これは Text モードの変種です (Section 22.8 [Text Mode], page 261 を参照してください)。Log Edit モードに入ると、Emacs はフック text-mode-hookおよび vc-log-mode-hookを実行します (Section 33.2.2 [Hooks], page 509 を参照してください)。

*vc-log*バッファーでは、1 行以上のヘッダー行 *(header lines)* を記入できます。これにはバージョンコントロールシステムにより提供される、追加の情報を指定します。各ヘッダー行は、そのバッ

ファーの最初の 1 行を占めなければなりません。ヘッダー行でない最初の行は、ログエントリーの開始として扱われます。たとえば以下のヘッダー行は、その変更が他の開発者によるものだということを示します:

```
Author: J. R. Hacker <jrh@example.com>
```

'Author'ヘッダーとは別に、Emacs は'Summary'ヘッダー (changeset の 1 行要約)、'Date'ヘッダー (手入力で指定されたコミット日時)、'Fixes'ヘッダー (変更によるバグフィクスへの参照) を認識します。すべてのバージョンコントロールシステムが、すべてのヘッダーを認識するわけではありません。そのシステムでサポートされていないヘッダーを指定した場合、それはログエントリーの一部として扱われます。

*vc-log*バッファーにいる間は、カレント VC ファイルセット (current VC fileset) とは、C-c C-c とタイプすることによりコミットされるファイルセットだと考えられます。その VC ファイルセットのファイル一覧を閲覧するには、C-c C-f (log-edit-show-files) とタイプします。その VC ファイルセットと、編集を開始したバージョンとの diff を閲覧するには、C-c C-d (log-edit-show-diff) とタイプしてください。

VC ファイルセットの diff にもとづいて変更されたすべてのファイルと関数をリストする雛形 (skeleton) となる ChangeLog エントリーを生成して ChangeLog エントリー生成の助けとするには、C-c C-w (log-edit-generate-changelog-from-diff) とタイプします。空のままの連続したエントリーは M-q (fill-paragraph) によって結合されます。この雛形には、デフォルトではディレクトリー部分がないファイル名だけが含まれます。VC ルートまでのディレクトリー部分を前置したければ、diff-add-log-use-relative-namesをカスタマイズしてください。

VC ファイルセットが 1 つ以上の ChangeLog(Section 25.3 [Change Log], page 355 を参照してください) ファイルを含む場合、C-c C-a (log-edit-insert-changelog) とタイプすると、関連するエントリーを、*vc-log*バッファーに引用します。各 ChangeLogファイルの一番上のアイテムが、今日の日付でユーザー名があなたの場合、このコマンドはコミットされるファイルにマッチするエントリーのアイテムを検索して、それを挿入します。

コミットを中止するには、そのバッファーで単に C-c C-cをタイプしないで、バッファーを切り替えて他の編集を行うことができます。他のコミットを試みなければ、編集していたエントリーは*vc-log*に残っているので、後でそのバッファーに戻ってコミットを完了できます。

コミットコメントをコピーするために、以前のログエントリーの履歴を閲覧することもできます。これは、似たようなコメントで複数のコミットを行いたいとき便利です。これを行うコマンド M-n、M-p、M-s、M-rは、これらがミニバッファーの外で使用される点を除けば、ミニバッファーのヒストリーコマンド (Section 5.5 [Minibuffer History], page 36 を参照してください) と同様です。

25.1.5 バージョンコントロールへのファイルの登録

C-x v i　　visit しているファイルを、バージョンコントロールに登録します。

コマンド C-x v i (vc-register) は、カレント VC ファイルセットの各ファイルを、バージョンコントロールの配下に置きます。これは登録されていない VC ファイルセットにたいして、C-x v v が行う動作と基本的に同じですが、VC ファイルセットがすでに登録されているとき (Section 25.1.3 [Basic VC Editing], page 337 を参照してください)、C-x v vは他の操作を処理しますが、C-x v i はエラーをシグナルする、という点が異なります。

ファイルを登録するために、Emacs はバージョンコントロールシステムを選択しなければなりません。複数ファイルの VC ファイルセットにたいしては、VC Directory バッファーが使用するシステムを指定します (Section 25.1.10 [VC Directory Mode], page 346 を参照してください)。VC ファイルセットのファイルが 1 つで、そのファイルのディレクトリーがすでにバージョンコントロー

ルシステムに登録されたファイルを含むか、そのディレクトリーがバージョンコントロールシステムにより制御される作業ツリーの一部の場合、Emacs はそのシステムを選択します。複数のバージョンコントロールシステムが当てはまる場合、Emacs は変数 `vc-handled-backends`に最初に表れるシステムを使用します。Emacs がファイルを登録するバージョンコントロールシステムを見つけられなかった場合、レポジトリタイプの入力を求め、新しいレポジトリを作成して、そのレポジトリにファイルを登録します。

ほとんどのバージョンコントロールシステムでは、`C-x v i`または `C-x v v`でファイルを登録することにより、それを作業ツリー (working tree) に追加しますが、レポジトリには追加しません。そのようなファイルは、VC Directory バッファーでは '`added`'とラベル付けされ、モードラインのリビジョン ID には '`@@`'が表示されます。レポジトリにたいして登録を有効にするには、コミットを行わなければなりません (Section 25.1.3 [Basic VC Editing], page 337 を参照してください)。1 つのコミットに、ファイルの追加と、既存ファイルの編集の両方を含むことができるのに注意してください。

ロックベースのバージョンコントロールシステム (Section 25.1.1.4 [VCS Merging], page 334 を参照してください) では、ファイルの登録により、ファイルはロックされていない読み取り専用の状態に留まります。編集を開始するには、`C-x v v`とタイプします。

25.1.6 古いリビジョンの調査と比較

`C-x v =` カレント VC ファイルセットの作業ファイルと、編集を開始したバージョンを比較します (`vc-diff`)。プレフィクス引数を指定した場合、カレント VC ファイルセットの 2 つのリビジョンの入力を求め、それらを比較します。このコマンドを Dired バッファーから呼び出すこともできます (Chapter 27 [Dired], page 380 を参照してください)。

`C-x v D` 編集を開始したリビジョンの、作業ツリー全体を比較します (`vc-root-diff`)。プレフィクス引数を指定した場合、2 つのリビジョンの入力を求め、それらのツリーを比較します。

`C-x v ~` カレントファイルのリビジョンの入力を求め、それを別のバッファーで visit します (`vc-revision-other-window`)。

`C-x v g` カレントファイルの、注釈付きのバージョンを表示します。各行には、その行が変更された最新のリビジョンが表示されます (`vc-annotate`)。

`C-x v =` (`vc-diff`) は、*diff* を表示します。これはカレント VC ファイルセットの各作業ファイルを、編集を開始したときのバージョンと比較します。diff は別のウィンドウに、`*vc-diff*`という名前の Diff mode モードのバッファー (Section 15.10 [Diff Mode], page 164 を参照してください) で表示されます。このバッファーでは、通常の Diff モードコマンドが利用可能です。特に `g` (`revert-buffer`) コマンドは、ファイル比較を再び行い、新しい diff を生成します。

カレント VC ファイルセットの、任意の 2 つのリビジョンを比較するには、`C-u C-x v =`のようにプレフィクス引数を指定して、`vc-diff`を呼び出します。これは 2 つのリビジョン ID(Section 25.1.1.3 [VCS Concepts], page 334 を参照してください) の入力を求め、ファイルセットのそれらのバージョンの間の diff を表示します。このコマンドは、バージョンコントロールシステムが変更セットベースではなくファイルベースの場合 (たとえば CVS) に、複数ファイルの VC ファイルセットにたいして確実に動作しません。なぜなら同じリビジョン ID をもつ異なるファイル同士を、意味のある方法で関連させることができないからです。

リビジョン ID ではなく、他のフォーマットでリビジョンを指定するバージョンコントロールシステムもいくつかあります。たとえば Bazaar では、`C-u C-x v =`(および関連するコマンド) に '`date:yesterday`'と入力でき、これは昨日以降コミットされた最初のリビジョンを指定します。詳細については、バージョンコントロールシステムのドキュメントを参照してください。

Dired バッファー (Chapter 27 [Dired], page 380 を参照してください) で C-x v =または C-u C-x v =を呼び出すと、カレント行にリストされたファイルが、カレント VC ファイルセットとして扱われます。VC ファイルセットにはマークした複数のファイルを含めることもできます。

C-x v D (vc-root-diff) は、C-x v =と似ていますが、カレント作業ツリー全体の変更を表示します (たとえばカレント VC ファイルセットを含む作業ツリー)。このコマンドを Dired バッファーから呼び出すと、そのディレクトリーを含む作業ツリーに適用されます。

ツリー全体の任意の 2 つのリビジョンを比較するには、C-u C-x v =のようにプレフィクス引数を指定して vc-root-diffを呼び出します。これは 2 つのリビジョン ID(Section 25.1.1.3 [VCS Concepts], page 334 を参照) の入力を求めて、バージョンコントロールされたディレクトリー全体のバージョン間の diff を表示します (RCS、SCCS、CVS、SRC はこの機能をサポートしない)。

C-x v =および C-x v Dが、diff を生成するために使用する diffオプションをカスタマイズできます。オプションには、変数 vc-*backend*-diff-switches、vc-diff-switches、diff-switches(Section 15.9 [Comparing Files], page 163 を参照) の順に、最初の非 nilの値が使用されます。ここで *backend*は、関連するバージョンコントロールシステムで、たとえば Bazaar の場合は bzrです。nilは順番に次の変数をチェックすることを意味するので、スイッチを指定しない場合は最初の 2 つの値を tにします。ほとんどの vc-*backend*-diff-switches変数のデフォルトは nilですが、いくつかの backend のデフォルトは tです。Subversion のように、これらのバージョンコントロールシステムの diff実装は、一般的な diff オプションを受け付けません。

ファイルの古いバージョンを直接調べるには、作業ファイルを visit して、C-x v ~ *revision* RET (vc-revision-other-window) とタイプします。これは *revision*に対応するバージョンのファイルを取得して、それを *filename*.~*revision*~に保存してから、別のウィンドウで visit します。

多くのバージョンコントロールシステムでは、C-x v g (vc-annotate) とタイプして、行ごとにリビジョン情報の注釈付き *(annotated)* でファイルを閲覧できます。これは新しい “annotate” バッファーを作成して、各行に古さを示すカラーをつけて、ファイルのテキストを表示します。赤いテキストは新しく、古いものは青、その中間色は中間のバージョンを示します。デフォルトでは、一番古い変更を青、一番新しい変更を赤で、すべてのバージョンレンジにカラーをスケーリングします。変数 vc-annotate-background-modeが非 nilの場合、各行の世代を表すカラーはバックグラウンドカラーに適用され、フォアグラウンドカラーはデフォルトのカラーのままです。

C-x v gが使用する annotateオプションは vc-*backend*-annotate-switchesと vc-annotate-switchesでカスタマイズできます。これらの関数は上述した vc-*backend*-diff-switchesおよび vc-diff-switchesと同じように機能します。

C-x v gにプレフィクス引数を指定した場合には、Emacs はミニバッファーを使って、表示および注釈つけする (カレントファイル内容のかわりの) リビジョンと、カラーレンジがカバーすべきタイムスパンという 2 つの引数を読み取ります。

“annotate” バッファーでは、‘VC-Annotate’メニューから、これら、または他のカラースケールオプションが利用可能です。このバッファーでは、過去のリビジョンの注釈の表示、diff の閲覧、ログエントリーの閲覧を行うために、以下のキーを使うこともできます:

p 前のリビジョン (たとえば現在の注釈付きのリビジョンの 1 つ前のリビジョン) に注釈を付けます。数引数は繰り返し回数となるので、C-u 10 pは 10 個前のリビジョンに注釈を付けます。

n 次のリビジョン (たとえば現在の注釈付きのリビジョンの 1 つ後のリビジョン) に注釈を付けます。数引数は繰り返し回数です。

j カレント行に示されたリビジョンに解釈を付けます。

`a` カレント行に示されたリビジョンの、1 つ前のリビジョンに注釈を付けます。これはカレント行が変更される前の状態のファイルを見るとき便利です。

`f` カレント行に示されたリビジョンのファイルを、バッファーに表示します。

`d` カレント行のリビジョンと、その前のリビジョンの間の diff を表示します。これはカレント行のリビジョンが実際にどのように変更されたか、ファイルを見るとき便利です。

`D` カレント行のリビジョンと、その前のリビジョンの間で、(変更セットをサポートするバージョンコントロールシステムの) 変更セットのすべてのファイルの diff を表示します。これはカレント行のリビジョンが、実際にどのように変更されたかツリー内を見るとき便利です。

`l` カレント行のリビジョンのログを表示します。これはカレント行のリビジョンの変更にたいする執筆者 (author) の説明を見るのに便利です。

`w` 作業中のリビジョン (編集中のもの) に注釈を付けます。`p`や `n`を使って他のリビジョンを表示している場合、このキーで作業中のリビジョンに戻ることができます。

`v` 注釈の表示・非表示を切り替えます。これは邪魔になる注釈抜きでファイル内容だけを見たいとき便利です。

25.1.7 VC Change Log

`C-x v l` カレントファイルセットの変更履歴を表示します (`vc-print-log`)。

`C-x v L` カレントレポジトリの変更履歴を表示します (`vc-print-root-log`)。

`C-x v b l` 別ブランチの変更履歴を表示します (`vc-print-branch-log`)

`C-x v I` pull 操作が取り込む変更を表示します (`vc-log-incoming`)。

`C-x v O` push 操作が送信する変更を表示します (`vc-log-outgoing`)。

`C-x v h` カレントバッファーが visit 中のファイルのリージョンにたいして行われた変更の履歴を表示します (`vc-region-history`)。

`M-x vc-log-search RET`
指定したパターンの変更履歴を検索します。

`C-x v l` (`vc-print-log`) は、`*vc-change-log*`という名前のバッファーを表示して、誰が変更したのか、その日時、各変更のログエントリー (これらは`*vc-log*`バッファーを通じて入力したログエントリーと同じです。Section 25.1.4 [Log Buffer], page 339 を参照) を含む、カレントファイルセットに行われた変更の履歴を長い形式で表示します。ファイルを visit しているバッファーから呼び出した際には、そのファイル単独でカレントファイルセットが構成されるとともに、表示中の`*vc-change-log*`バッファーでは、そのファイルのリビジョンの中央にポイントが配置されます。VC Directory バッファー (Section 25.1.10 [VC Directory Mode], page 346 を参照) や Dired バッファー (Chapter 27 [Dired], page 380 を参照) から呼び出された場合には、マークされているすべてのファイル (マークされているファイルがなければディレクトリーバッファーのカレント行に表示されているファイルがデフォルト) からファイルセットが構成されることになります。

ファイルセットに 1 つ以上のディレクトリーが含まれる場合には、VC バックエンドがサポートしていれば短い変更ログ、サポートしていなければ長い形式のログを表示する`*vc-change-log*`バッファーが得られます。

プレフィックス引数を指定すると、`*vc-change-log*`バッファーで中央に表示するリビジョン、および表示するリビジョンの最大数の入力を求めます。

C-x v L (vc-print-root-log) はバージョンコントロールされたディレクトリーツリー全体の履歴を示す*vc-change-log*バッファーを表示します (RCS、SCCS、CVS、SRC はこの機能をサポートしない)。コマンドにプレフィクス引数を与えると表示するリビジョンの最大数の入力を求めます。プレフィクス数引数は入力を求めることなくリビジョンの最大数を指定します。C-1 C-x v L や C-u 1 C-x v L のようにプレフィクス数引数が 1 ならコマンドはリビジョン ID の入力を求めて、そのリビジョンで導入された変更 (diff) とともにリビジョンのログエントリーを表示します (RCS や CVS のように機能に劣るバージョンコントロールシステムには diff とともにリビジョンログを表示するコマンドがなく、このコマンドはそれらにたいしてはログエントリーだけを表示するので、以下の dや Dをタイプして diff を要求できる)。

C-x v Lでは履歴は簡略化された形式で表示され、通常は各ログエントリーの最初の行だけが表示されます。しかし*vc-change-log*バッファーで RET (log-view-toggle-entry-display) とタイプすると、ポイント位置のリビジョンのログエントリー全体を表示します。2 回目の RETで、再びそれを隠します。

C-x v b l *branch-name* RET (vc-print-branch-log) は vc-print-root-logと同様にバージョンコントロールされているディレクトリーツリーの履歴を*vc-change-log*バッファーに表示しますが、カレントブランチではなく別ブランチの履歴を表示します。そのためこのコマンドは履歴を表示するブランチの入力を求めます。

分散型のバージョンコントロールシステムでは、C-x v I (vc-log-incoming) コマンドは、次回にバージョンコントロールの pull コマンドを実行するときに、他所のリモートから受け取る新しいリビジョンにより適用される変更をログバッファーに表示します (Section 25.1.11.2 [Pulling / Pushing], page 350 を参照)。ここで他所のリモートとは、バージョンコントロールシステムで定義された変更を pull するデフォルトのリモートのことです。プレフィクス引数を指定すると、vc-log-incoming は特定のリモートの入力を求めます。同様に、C-x v O (vc-log-outgoing) は、次回に push コマンドを実行するときに、リモートに送る変更を表示します。プレフィクス引数を指定すると、送信先となる特定のリモート入力を求めます。一部のバージョンコントロールシステムではブランチ名かもしれません。

*vc-change-log*バッファーでは、リビジョンのログまたはファイル間の移動や、過去のリビジョン (Section 25.1.6 [Old Revisions], page 341 を参照してください) を調べたり比較するために、以下のキーを使うことができます:

p 前のリビジョンエントリーに移動します (log バッファーのリビジョンエントリーは通常、日時の降順になっているので、前のリビジョンアイテムは通常、もっと新しいリビジョンに対応します)。数引数は繰り返し回数です。

n 次のリビジョンエントリーに移動します。数引数は繰り返し回数です。

a カレント行のリビジョンに注釈を付けます (Section 25.1.6 [Old Revisions], page 341 を参照してください)。

e ポイント位置に表示された変更コメントを修正します。すべてのバージョンコントロールシステムが、変更コメントの修正をサポートするわけではないことに注意してください。

f カレント行に示されたリビジョンを visit します。

d ポイント位置のリビジョンと、次に古いリビジョンとの間で、特定のファイルにたいする diff を表示します。

D ポイント位置のリビジョンと、次に古いリビジョンとの間で、変更セットの diff を表示します。これは、そのリビジョンですべてのファイルにたいして行われた変更を表示します。

RET 簡略形式の log バッファー (たとえば C-x v Lで作成されたバッファー) で、ポイント位置のログエントリーにたいして、完全なログエントリーの表示・非表示を切り替えます。

多くのログエントリーを取得するには時間がかかるので、*vc-change-log*バッファーは、デフォルトで 2000 を超えるリビジョンは表示しません。変数 vc-log-show-limitはこの制限を指定します。この値を 0 にセットすると、制限が削除されます。既存の*vc-change-log*で、バッファーの最後のボタン 'Show 2X entries'または 'Show unlimited entries'をクリックして、表示するリビジョン数を増やすこともできます。しかし RCS、SCCS、CVS、SRC はこの機能をサポートしません。

変更履歴を確認する有用な変種はコマンド vc-region-history (デフォルトでは C-x v hにバインド) により提供されます。これはカレントバッファーのファイルのポイントとマーク (Chapter 8 [Mark], page 52 を参照) の間にあるリージョンに行われた変更の履歴を*VC-history*バッファーに表示します。変更履歴にはコミットログメッセージ (commit log messages) と変更自体の Diff 形式も含まれます。

カレントバッファーの興味がある変更にたいしてリージョンをマークした後に、このコマンドを呼び出します。このコマンドがポップアップする*VC-history*バッファーでは、上述した*vc-change-log*バッファーで利用可能なコマンドすべてと、Diff モード (Section 15.10 [Diff Mode], page 164 を参照) で定義されたコマンドも使用できます。

このコマンドは現在のところ Git と Mercurial(hg) だけで利用可能です。

コマンド vc-log-searchにより変更ログのパターンによる検索が可能になります。これはパターン (正規表現) の入力を求めて、パターンにマッチするログメッセージをもつ変更履歴内のすべてのエントリーを表示します。プレフィクス引数とともに呼び出された際には、コマンドはこの目的にたいして実行する固有の VCS シェルコマンドの入力も求めます。

25.1.8 バージョンコントロール操作のアンドゥ

C-x v u カレント VC ファイルセットの作業ファイルを、最後のリビジョンにリバートします (vc-revert)。

カレント VC ファイルセットにたいするすべての変更を破棄したい場合、C-x v u (vc-revert) とタイプします。これは変更を破棄する前に同意を求めます。同意するとそのファイルセットはリバートされます。

vc-revert-show-diffが非 nilならは、このコマンドは作業ファイル (複数可) と編集を開始したリビジョンとの間の diff を表示します。その後で diff バッファーは (この変数が killなら)kill、あるいは (それ以外の非 nil値なら) バリーされます。C-x v uで diff を表示させたくない場合は、この変数に nilをセットしてください (この設定をしても C-x v =で直接 diff を表示できる。Section 25.1.6 [Old Revisions], page 341 を参照)。

ロックベースのバージョンコントロールシステムでは、C-x v uはファイルをロックしないまま残します。編集を再開するには、再度ロックしなければなりません。ファイルをロックしてから、やはりそれを変更しないと決めたときも、C-x v uでファイルのロックを開放できます。

25.1.9 バージョンコントロールファイルを無視する

C-x v G カレントのバージョンコントロールシステム配下のファイルを無視します (vc-ignore)。

ソースツリーの多くは、エディターのバックアップや、オブジェクトファイル、バイトコードファイル、ビルドされるプログラムなどの、バージョン管理する必要のないファイルを含みます。これらは単に追加しないだけでも構いませんが、常に不明なファイルとして現れるでしょう。ツリーのトッ

プの、無視するファイルのリストにこれらのファイルを追加して、それらを無視するようにバージョンコントロールシステムに指示することもできます。これを行うには C-x v G (vc-ignore) が助けとなるでしょう。プレフィクス引数を指定すると、無視するファイルリストからファイルを削除できます。

25.1.10 VC Directory モード

VC Directory バッファーは、ディレクトリーツリーにあるファイルのバージョンコントロール状態を見て、それらのファイルにバージョンコントロール操作を実行するために特化したバッファーです。特に複数ファイルの VC ファイルセットにたいして、C-x v v のようなコマンドを適用するのに使用されます (Section 25.1.10.2 [VC Directory Commands], page 347 を参照してください)。

VC Directory バッファーを使用するには、C-x v d (vc-dir) とタイプします。これはミニバッファーを使用してディレクトリーの名前を読み取り、そのディレクトリーにたいする VC Directory バッファーに切り替えます。デフォルトでは、バッファーの名前は*vc-dir*です。その内容については、以下で説明します。

vc-dirコマンドは、指定したディレクトリーで使用されているバージョンコントロールシステムを自動的に検知します。そのディレクトリーにたいして複数のバージョンコントロールシステムが使用されている場合、C-u C-x v dのようにプレフィクス引数を指定して、このコマンドを呼び出す必要があるでしょう。これは VC Directory バッファーが使用すべきバージョンコントロールシステムの入力を求めます。

Dired バッファー (Chapter 27 [Dired], page 380 を参照) から VC コマンドを呼び出すこともできます。この場合には、呼び出した VC コマンドはマークされているファイル (マークされているファイルがなければカレント行のファイルがデフォルト) をカレントファイルセットとみなします (Section 25.1.3 [Basic VC Editing], page 337 を参照)。

25.1.10.1 VC Directory バッファー

VC Directory バッファーは、バージョンコントロールされたファイルと、それらのバージョンコントロール状態を含みます。これは、(C-x v dを呼び出すことにより指定される) カレントディレクトリーの、注目すべき状態のファイルとサブディレクトリーをだけリストします。最新のファイル (レポジトリのものと同じ) は省略されます。サブディレクトリーのファイルがすべて最新の場合、そのサブディレクトリーもリストされません。例外として、VC コマンドの直接の結果として最新になったファイルはリストされます。

以下は VC Directory バッファーのリストの例です:

```
                    ./
    edited          configure.ac
*   added           README
    unregistered    temp.txt
                    src/
*   edited          src/main.c
```

2 つの作業ファイル、カレントディレクトリーの configure.acと、サブディレクトリー src/の main.cは、変更されていますがコミットされていません。README という名前のファイルは追加されましたが、まだコミットされていません。そして temp.txtはバージョンコントロールの配下にありません (Section 25.1.5 [Registering], page 340 を参照してください)。

エントリー READMEおよび src/main.cの隣の '*'という文字は、ユーザーがそれらのファイルをカレント VC ファイルセットとしてマークしたことを示します (以下を参照してください)。

上記は、Bazaar、Git、Mercurial のような分散型のバージョンコントロールシステムでの典型的な例です。他のシステムでは、他の状態も見られます。たとえば CVS は、レポジトリが変更されて

いて、それがまだ作業ファイルに適用されていないときは、‘needs-update’という状態を表示します。RCS と SCCS は、ロックされているファイルの状態に、ロックしているユーザーの名前を表示します。

VC Directory バッファーは、変数 vc-directory-exclusion-listにリストされているサブディレクトリーを省略します。この変数のデフォルト値には、バージョンコントロールシステムにより内部的に使用されるディレクトリーが含まれています。

25.1.10.2 VC Directory コマンド

Emacs は VC Directory バッファーの操作と、カレント VC ファイルセットに属させるために、ファイルをマークするためのコマンドをいくつか提供します。

n
SPC 次のエントリーにポイントを移動します (vc-dir-next-line)。

p 前のエントリーにポイントを移動します (vc-dir-previous-line)。

TAB 次のディレクトリーエントリーに移動します (vc-dir-next-directory)。

S-TAB 前のディレクトリーエントリーに移動します (vc-dir-previous-directory)。

RET
f カレント行にリストされたファイル、またはディレクトリーを visit します (vc-dir-find-file)。

o カレント行にリストされたファイル、またはディレクトリーを別のウィンドウで visit します (vc-dir-find-file-other-window)。

m カレント行のファイルまたはディレクトリーをマークして、それをカレント VC ファイルセットに加えます (vc-dir-mark)。リージョンがアクティブのときは、リージョンの中のすべてのファイルをマークします。

すでにマークされたディレクトリーの中のファイル、またはそのサブディレクトリーは、このコマンドではマークされません。同様に、ツリーの中のいくつかのファイルがマークされているディレクトリーは、このコマンドではマークされません。

M ポイントがファイルエントリーにあるときは、同じ状態のすべてのファイルをマークします。ポイントがディレクトリーエントリーにあるときは、そのディレクトリーツリーのすべてのファイルをマークします (vc-dir-mark-all-files)。プレフィクス引数を指定した場合、リストされたファイルとディレクトリーのすべてをマークします。

% m
* % regexp でファイルをマークするためにこのコマンドを使用できます (vc-dir-mark-by-regexp)。プレフィックス引数が与えられた場合にはマークではなくマークを解除します。

* r このコマンドは編集済み (edited)、追加済み (added)、削除済み (removed) を含む登録済み (registered) のいずれかの状態にあるファイルをマークするために用いることができます (vc-dir-mark-registered-files)。

G ポイントの下にあるファイルを VC が無視 (ignore) すべきファイルのリストに追加します (vc-dir-ignore)。たとえば VC が Git なら、そのファイル.gitignoreファイルに追加します。プレフィクスが与えられたら、マークしたすべてのファイルにこれを行います。

`q` VC Directory バッファーを終了して、隠します (`quit-window`)。

`u` カレント行のファイル、またはディレクトリーのマークを外します (`vc-dir-unmark`)。リージョンがアクティブのときは、リージョンの中のすべてのファイルのマークを外します。

`U` ポイントがファイルエントリーにあるときは。同じ状態のすべてのファイルのマークを外し、ポイントがディレクトリーエントリーにあるときは、そのディレクトリーツリーのすべてのファイルのマークを外します (`vc-dir-unmark-all-files`)。プレフィクス引数を指定した場合、すべてのファイルおよびディレクトリーのマークを外します。

`x` 状態が '`up-to-date`'、または '`ignored`'のファイルを隠します (`vc-dir-hide-up-to-date`)。プレフィクス引数を指定した場合、状態がポイント位置のアイテムと同じアイテムを隠します。

VC Directory バッファーでは、`m` (`vc-dir-mark`) または `M` (`vc-dir-mark-all-files`) でマークしたすべてのファイルが、カレント VC ファイルセットになります。ディレクトリーエントリーを `m`でマークした場合、そのディレクトリーツリーにリストされたすべてのファイルが、カレント VC ファイルセットになります。カレント VC ファイルセットに属するファイルとディレクトリーは、VC Directory ではバージョンコントロール状態の隣に、文字 '`*`'が示されます。この方法により `C-x v v` (Section 25.1.3 [Basic VC Editing], page 337 を参照してください)、`C-x v =` (Section 25.1.6 [Old Revisions], page 341 を参照してください)、`C-x v u` (Section 25.1.8 [VC Undo], page 345 を参照してください) のような VC コマンドが作用する、複数ファイルの VC ファイルセットをセットアップできます。

VC Directory バッファーは、`C-x v`というプレフィクスをもつコマンドを、1 キーで入力するショートカット (`=`、`+`、`l`、`i`、`D`、`L`、`G`、`I`、`O`、`v`) を定義します。

たとえば、VC Directory バッファーで開いて編集された一連のファイルは、'`edited`'という状態でリストされ、それらのファイルをマークして、`v`または `C-x v v` (`vc-next-action`) でコミットできます。バージョンコントロールシステムが変更セットベースの場合、Emacs はそれらのファイルを 1 つのリビジョンとしてコミットします。

VC Directory バッファーでは、以下のコマンドによりカレント VC ファイルセットの検索と置換を処理することもできます:

`S` ファイルセットを検索します (`vc-dir-search`)。

`Q` ファイルセットにたいして、正規表現による問い合わせ置換を行います (`vc-dir-query-replace-regexp`)。

`M-s a C-s` ファイルセットにたいして、インクリメンタル検索を行います (`vc-dir-isearch`)。

`M-s a C-M-s`
ファイルセットにたいして、インクリメンタルな正規表現検索を行います (`vc-dir-isearch-regexp`)。

複数ファイルに作用する点を除けば、これらのコマンドは 1 つのバッファーに作用する同等のコマンドに似ています (Chapter 12 [Search], page 105 を参照してください)。

VC Directory バッファーは追加でブランチ関連のコマンドを定義しており、それらはプレフィックス `b`で始まります:

`b c` 新たにブランチを作成します (`vc-create-branch`)。Section 25.1.11.4 [Creating Branches], page 351 を参照してください。

`b l` ブランチ名の入力を求め、そのブランチの変更履歴を表示します (`vc-print-branch-log`)。

`b s` ブランチを切り替えます (`vc-switch-branch`)。Section 25.1.11.1 [Switching Branches], page 349 を参照してください。

`d` マークされたファイル、マークされたファイルがなければカレントファイルを削除します (`vc-dir-clean-delete`)。バージョンコントロールシステムではマークされたファイルが削除される訳ではないので、これは主としてバージョンコントロールシステムに未登録のファイルにたいして有用です。

上記のコマンドは、メニューバーおよび`mouse-2`によるコンテキストメニューを通じても利用可能です。さらに VC のバックエンドのいくつかは、そのバックエンド特有のコマンドを提供するメニューを使用します。たとえば Git と Bazaar では、*stashes(*隠してあるもの*)* と *shelves(*棚*)*(コミットされていない変更を一時的に除外して、後でそれを戻すコマンド) を操作できます。

25.1.11 バージョンコントロールのブランチ

バージョンコントロールの活用法の 1 つとして、ブランチ *(branches)* と呼ばれる複数の独立した開発ラインのサポートがあります。中でもとりわけブランチは、プログラムの安定版 (stable) と開発版 (development) を個別に保守したり、関係のない機能を他の版から隔離して開発するのに使用されます。

現在のところ VC のブランチ操作にたいするサポートは、かなり制限されています。分散型のバージョンコントロールシステムにたいしては、あるブランチを他のブランチのコンテンツで更新するコマンドと、2 つの異なるブランチの変更をマージするコマンドを提供します。集中型のバージョンコントロールシステムにたいしては、異なるブランチからチェックアウトして、新規または異なるブランチにコミットするコマンドを提供します。

25.1.11.1 ブランチ間の切り替え

さまざまなバージョンコントロールシステムにおいて、ブランチが実装される方法は異なり、VC はこれらの違いを完全に隠蔽することはできません。

Bazaar と Mercurial を含む分散型バージョンコントロールシステムのいくつかは、ノーマルモードの操作では、各ブランチは自身の作業ディレクトリーツリーをもつので、ブランチの切り替えは単にディレクトリーを切り替えるだけです。Git では、ブランチは通常、同じディレクトリーの共通ロケーション *(co-located)* を使用し、ブランチの切り替えは、作業ツリーの内容をそのブランチに一致するように変更する `git checkout`を使用して行われます。Bazaar も共通ロケーションをサポートしており、この場合はコマンド `bzr switch`によりカレントディレクトリーでブランチを切り替えます。Subversion では他のブランチへの切り替えにはコマンド `svn switch`を使用します。Mercurial では他のブランチへの切り替えにはコマンド `hg update`を使用します。

カレントディレクトリーの他のブランチに切り替える VC コマンドは、`C-x v b s` *`branch-name`* `RET` (`vc-switch-branch`) です。

集中型のバージョンコントロールシステムでは、最新の作業ファイルで `C-u C-x v v`とタイプして (Section 25.1.3.3 [Advanced C-x v v], page 339 を参照してください)、他のブランチのリビジョン ID を入力することにより、ブランチ間を切り替えることもできます。たとえば CVS では、*trunk(*幹の意。開発の主要ラインを示します*)* のリビジョンは通常、1.1、1.2、1.3、. . . という形式をもち、最初のブランチがリビジョン 1.2 から作成された場合、リビジョン 1.2 は 1.2.1.1、1.2.1.2、. . . というリビジョン ID をもち、さらに 2 番目のブランチが同じくリビジョン 1.2 から作成された場合、それは 1.2.2.1、1.2.2.2、. . . という形式になります。ブランチのリビジョン ID から最後の部分を除いた

(たとえば 1.2.1)、ブランチ *ID(branch ID)* を指定して、そのブランチの最新のリビジョンに切り替えることもできます。

ロックベースのシステムでは、他のブランチに切り替えることにより、作業ツリーのロックが解除 (書き込み禁止) になります。

1 度ブランチを切り替えると、そのブランチを他に切り替えるまで、VC コマンドはそのブランチに適用されます。たとえば任意の VC ファイルセットをコミットすると、そのブランチにコミットされるようになります。

25.1.11.2 ブランチへ/からの変更の pull/push

`C-x v P` 分散型のバージョンコントロールシステムでは、カレントブランチからの変更により、その変更で他のロケーションを更新 (変更を “push” する、とも言います) します。この概念は集中型のバージョンコントロールシステムには存在しません。

`C-x v +` 分散型のバージョンコントロールシステムでは、他のロケーションから変更を “pull” することにより、カレントのブランチを更新します。

集中型のバージョンコントロールシステムでは、カレント VC ファイルセットを更新します。

分散型のバージョンコントロールシステムでは、コマンド `C-x v P` (`vc-push`) は、カレントブランチからの変更により他のロケーションを更新します。プレフィクス引数を与えた場合、このコマンドは実行する正確なバージョンコントロールコマンドの入力をもとめます。これにより変更をどこに push するか指定できます。デフォルトは、Bazaar では `bzr push`、Git では `git push`、Mercurial では `hg push`です。デフォルトのコマンドは常に、ブランチ設定からバージョンコントロールシステムにより決定されるデフォルトのロケーションに push します。

pull する前に `C-x v O` (`vc-log-outgoing`) を使用して、送信される変更の log バッファーを閲覧できます。Section 25.1.7 [VC Change Log], page 343 を参照してください。

現在のところ、このコマンドは Bazaar、Git、Mercurial だけでサポートされます。“push” という概念は集中型のバージョンコントロールシステムには存在しません。なぜなら、この操作は変更セットのコミットの一部なので、集中型の VCS でこのコマンドを呼び出すと、エラーをシグナルします。Bazaar の *bound branch* でこのコマンドを試みたときもエラーをシグナルします。変更セットのコミットは、自動的に変更を (ローカルのブランチがバインドされている) リモートのレポジトリに push するからです。

分散型のバージョンコントロールシステムでは、コマンド `C-x v +` (`vc-pull`) は、カレントブランチと作業ツリーを更新します。これは通常、リモートのブランチのコピーを更新するのに使用されます。プレフィクス引数を与えた場合、このコマンドは使用する正確なバージョンコントロールコマンドの入力をもとめます。これにより変更をどこから pull するか指定できます。プレフィクス引数を指定しない場合は、バージョンコントロールシステムにより決定される、デフォルトのロケーションから pull します。

分散型のバージョンコントロールシステムの中で、現在`C-x v +`がサポートするのは Bazaar、Git、Mercurial だけです。Bazaar では、これは通常のブランチにたいしては、(マスターブランチをミラーリングされたブランチに pull するために) `bzr pull`を呼び出し、バインドされたブランチにたいしては、(中心となるレポジトリから pull するために) `bzr update`を呼び出します。Git では、これはリモートのレポジトリから変更を取得して、それをカレントブランチにマージするために `git pull` を呼び出します。Mercurial では、デフォルトのリモートレポジトリから変更を取得して、作業ディレクトリーを更新するために `hg pull -u`を呼び出します。

pull する前に C-x v I (vc-log-incoming) を使用して、適用される変更の log バッファーを閲覧できます。Section 25.1.7 [VC Change Log], page 343 を参照してください。

CVS のような集中型のバージョンコントロールシステムでは、C-x v +はレポジトリからカレント VC ファイルセットを更新します。

25.1.11.3 ブランチのマージ

C-x v m　分散型のバージョンコントロールシステムでは、カレントのブランチに他のブランチの変更をマージします。

集中型のバージョンコントロールシステムでは、カレント VC ファイルセットに他のブランチの変更をマージします。

ブランチで開発している場合、すでに他のブランチで行われた変更をマージ *(merge)* する必要があるときがあります。これは 2 つのブランチでの変更が重なる場合もあるため、些細な操作とはいえません。

分散型のバージョンコントロールシステムでは、マージはコマンド C-x v m (vc-merge) により行われます。Bazaar では、これは bzr mergeに渡す正確な引数の入力を求めます。そのとき、可能であれば目的にかなったデフォルトを提示します。Git では、これはマージするブランチ名の入力を求めます。このとき、(カレントレポジトリが知っているブランチ名にもとづく) 補完を行います。Mercurial では hg mergeに渡す引数の入力を求めます。マージコマンドの実行による出力は、他のバッファーに表示されます。

CVS のような集中型のバージョンコントロールシステムでは、C-x v mはブランチ ID、または 2 つのリビジョン ID の入力を求めます。コマンドはそのブランチからの変更点、または指定した 2 つのリビジョン間の差分を探して、それらの変更をカレント VC ファイルセットにマージします。RET だけをタイプした場合、Emacs は単にそのファイルをチェックアウトしたブランチに行われた変更をマージします。

マージを処理した直後は、作業ツリーだけが変更されており、C-x v Dおよび関連するコマンドで、マージにより生成された変更をレビューできます (Section 25.1.6 [Old Revisions], page 341 を参照してください)。2 つのブランチが重なった変更をもつ場合、マージは衝突 *(conflict)* を生成します。その場合、マージコマンドの出力には警告が現れ。影響のある作業ファイルの、衝突する 2 つの変更の周囲に、衝突マーカー *(conflict markers)* が挿入されます。衝突を解決するには、衝突するファイルを編集しなければなりません。編集が終わったら、マージが効果を発揮するように、通常の方法により変更したファイルをコミットしなければなりません (Section 25.1.3 [Basic VC Editing], page 337 を参照してください)。

25.1.11.4 新しいブランチの作成

CVS のような集中型のバージョンコントロールシステムでは、Emacs はコミット操作の一部として、新しいブランチの作成をサポートします。変更された VC ファイルセットをコミットするとき、C-u C-x v v (vc-next-actionのようにプレフィクス引数を指定します。Section 25.1.3.3 [Advanced C-x v v], page 339 を参照してください). すると、Emacs は新しいリビジョンのリビジョン ID の入力を求めます。ここでカレントリビジョンから開始するブランチの。適切なブランチ ID を指定する必要があります。たとえば、カレントリビジョン ID が 2.5 の場合、ブランチ ID は 2.5.1、2.5.2、... となるべきでしょう。ブランチ ID は、その時点での既存のブランチの番号に依存します。

この手順は git や Mercurial のような、分散型バージョンコントロールシステムでは機能しないでしょう。これらのシステムでは、かわりにコマンド vc-create-branch (C-x v b c *branch-name* RET) を使用する必要があります。

(すでにブランチのヘッドではない) 古いリビジョンに新しいブランチを作成するには、最初にそのリビジョンを選択します (Section 25.1.11.1 [Switching Branches], page 349 を参照してください)。その後の手順は、ロックベースのバージョンコントロールシステムを使っているか、マージベースのものを使っているかで異なります。

ロックベースのバージョンコントロールシステムでは、C-x v vで古いリビジョンのブランチを選択します。古いリビジョンを選択する場合、本当に新しいブランチを作成したいのか確認を求めます。これに no と応えた場合、かわりに最新のリビジョンをロックする機会が与えられます。マージベースのバージョンコントロールシステムでは、このステップはスキップします。

変更を行なってから、再び C-x v vとタイプして、新しいリビジョンをコミットします。これは選択されたリビジョンから始まる、新しいブランチを作成します。

ブランチが作成された後は、それ以降のコミットは、そのブランチに新しいリビジョンを作成します。ブランチを離れるには、C-u C-x v vで明示的に異なるリビジョンを選択しなければなりません。

25.2 プロジェクトで作業する

プロジェクト *(project)* とは 1 つ以上のプログラムを生成するために使用するファイルのコレクションです。プロジェクトに属するファイルは、通常はディレクトリー階層に格納されます。この階層のトップレベルのディレクトリーはプロジェクトルート *(project root)* として知られています。

与えられたディレクトリーが何らかのプロジェクトのルートかどうかは、プロジェクトバックエンド *(project back-end)* として知られているプロジェクト固有のインフラストラクチャーにより判断されます。Emacs が現在のところサポートするバックエンドは VCS レポジトリをプロジェクトとみなした VC を考慮するプロジェクト (Section 25.1 [Version Control], page 332 を参照)、および EDE (Section 25.5 [EDE], page 369 を参照) の 2 つです。将来的には追加のプロジェクトタイプをサポートするように拡張される予定です。

ファイルがプロジェクトに属するかどうかもプロジェクトバックエンドにより判断されます。たとえば VC を考慮するバクエンドは “無視 (ignored)” されたファイル (Section 25.1.9 [VC Ignore], page 345 を参照)、更にデフォルトでは “未追跡 (untracked)” のファイルもプロジェクトの一部とはみなしません。この挙動は変数 project-vc-include-untrackedで制御できます。

25.2.1 ファイルを操作するプロジェクトコマンド

C-x p f msgid "Visit a file that belongs to the current project ." カレントプロジェクトに属するファイルを visit します (project-find-file)。

C-x p g カレントプロジェクトに属するすべてのファイルにたいして regexp にたいするマッチを検索します (project-find-regexp)。

M-x project-search
カレントプロジェクトに属するすべてのファイルにたいしてインタラクティブに regexp にたいするマッチを検索します。

C-x p r カレントプロジェクトに属するすべてのファイルにたいして、regexp にたいする問い合わせつき置換を行います (project-query-replace-regexp)。

C-x p d カレントプロジェクトのルートディレクトリーにたいして Dired を実行します (project-dired)。

C-x p v カレントプロジェクトのルートディレクトリーで vc-dirを実行します (project-vc-dir)。

`C-x p s` カレントプロジェクトのルートディレクトリーで下位シェルを開始します (`project-shell`)。

`C-x p e` カレントプロジェクトのルートディレクトリーで Eshell を開始します (`project-eshell`)。

`C-x p c` カレントプロジェクトのルートディレクトリーでコンパイルを実行します (`project-compile`)。

`C-x p !` カレントプロジェクトのルートディレクトリーでシェルコマンドを実行します (`project-shell-command`)。

`C-x p &` カレントプロジェクトのルートディレクトリーでシェルコマンドを非同期に実行します (`project-async-shell-command`)。

Emacs はプロジェクトファイルを手軽に扱うためのコマンドを提供します。このサブセクションではそれらのコマンドについて説明します。

ここで説明するすべてのコマンドはカレントプロジェクト *(current project)* という概念を共有します。カレントプロジェクトはコマンド呼び出し時にカレントだったバッファーの `default-directory` (Section 15.1 [File Names], page 145 を参照) により判断されます。そのディレクトリーが認識可能なプロジェクトに属していないようなら、これらのコマンドはプロジェクトディレクトリーの入力を求めます。

コマンド `C-x p f` (`project-find-file`) はカレントプロジェクトに属するファイルを visit(Section 15.2 [Visiting], page 146 を参照) する手軽な手段です。`C-x C-f`とは異なり、このコマンドは visit するファイルの完全な名前のタイプを要求せず、ファイルのベース名 (先頭のディレクトリー部分を省略) だけでタイプできます。それに加えてコマンドが補完候補とみなすのはカレントプロジェクトに属するファイルだけであり、それ以外は含まれません。ポイント位置にファイル名があれば、このコマンドは “将来のヒストリー (future history)” の最初の要素としてそのファイルを提案します。プレフィックス引数が与えられた場合には、`vc-directory-exclusion-list`にリストされている VCSディレクトリーを除く、プロジェクトルート配下のすべてのファイルが含められます。

コマンド `C-x p g` (`project-find-regexp`) は `rgrep` (Section 24.4 [Grep Searching], page 313 を参照) と似ていますが、カレントプロジェクトに属するファイルだけを検索します。このコマンドは検索する正規表現の入力を求めて、検索結果を Xref モードのコマンドを使用してマッチの選択が可能な Xref モードのバッファーでポップアップします。このコマンドをプレフィクス引数とともに呼び出した際には、検索を開始するベースディレクトリーの入力を追加で求めます。これはたとえばプロジェクトルートの特定のサブディレクトリー配下のファイルに検索を制限することを可能にします。このコマンドがマッチを表示する方法は `xref-auto-jump-to-first-xref`の値の影響を受けます (Section 25.4.1.3 [Identifier Search], page 360 を参照)。

`M-x project-search`は`project-find-regexp`の一連の変種です。これはカレントプロジェクトのファイルを検索するために正規表現の入力を求めますが、すべてのマッチを探して表示するかわりに、マッチしたファイルを編集できるようにマッチを見つけたら停止してマッチしたファイルのマッチした locus を visit します。マッチの残りを探すには `M-x fileloop-continue RET` とタイプしてください。

`C-x p r` (`project-query-replace-regexp`) は `project-search`と似ていますが、`query-replace` (Section 12.10.4 [Query Replace], page 124 を参照) が行うように見つかったそれぞれのマッチにたいして置き換えるかどうかを尋ねて、それに応答した後は次のマッチへと継続します。その応答により Emacs が query-replace ループを exit してしまったら、`M-x fileloop-continue RET` で後から継続することができます。

コマンド C-x p d (project-dired) は補完付きでカレントプロジェクト内部のディレクトリーの選択を求めます。そしてその中にあるファイルをリストする Dired バッファー (Chapter 27 [Dired], page 380 を参照) をオープンします。

コマンド C-x p D (project-dired) はカレントプロジェクトのルートディレクトリーのファイルをリストする Dired バッファー (Chapter 27 [Dired], page 380 を参照) をオープンします。

コマンド C-x p v (project-vc-dir) はカレントプロジェクトのルートディレクトリー配下にあるディレクトリーツリー内のファイルのバージョンコントロール状態をリストする VC Directory バッファー (Section 25.1.10 [VC Directory Mode], page 346 を参照) をオープンします。

コマンド C-x p s (project-shell) はカレントプロジェクトのルートディレクトリーを作業ディレクトリーとするシェルセッション (Section 31.5 [Shell], page 457 を参照) を新たなバッファーで開始します。

コマンド C-x p e (project-eshell) はカレントプロジェクトのルートディレクトリーを作業ディレクトリーとする Eshell セッションを新たなバッファーで開始します。Section “Eshell” in *Eshell: The Emacs Shell* を参照してください。

コマンド C-x p c (project-compile) はカレントプロジェクトのルートディレクトリーでコンパイル (Section 24.1 [Compilation], page 309 を参照) を実行します。

コマンド C-x p ! (project-shell-command) はカレントプロジェクトのルートディレクトリーで shell-commandを実行します。

コマンド C-x p & (project-async-shell-command) はカレントプロジェクトのルートディレクトリーで async-shell-commandを実行します。

25.2.2 バッファーを操作するプロジェクトコマンド

C-x p b　カレントプロジェクトに属する他のバッファーに切り替えます (project-switch-to-buffer)。

C-x p C-b　プロジェクトバッファーをリストします (project-list-buffers)。

C-x p k　カレントプロジェクトに属するすべての生きたバッファーを kill します (project-kill-buffers)。

プロジェクトでの作業ではプロジェクトに属するファイルを visit するバッファーや、(project-compileが作成する*compilation*バッファーのように) プロジェクトには属するものの何のファイルも visit しないバッファーを多数もつ可能性があります。コマンド C-x p b (project-switch-to-buffer) はカレントプロジェクトに属するバッファーのみを補完候補とみなしてバッファーの入力を求めることにより、カレントプロジェクトに属するバッファー間の切り替えを助けます。

C-x p C-b (project-list-buffers) はコマンド list-buffers (see Section 16.2 [List Buffers], page 178) と同じように既存のバッファーのリストを表示しますが、カレントプロジェクトに属するバッファーだけをリストするコマンドです。

プロジェクトの作業が終わった際には、Emacs セッションを小さく保つためにプロジェクトに属するすべてのバッファーを kill したいと思うかもしれません。コマンド C-x p k (project-kill-buffers) によりこれを行うことができます。これはカレントプロジェクトに属する project-kill-buffer-conditionsのいずれかを満足するすべてのバッファーを kill します。project-kill-buffers-display-buffer-listが非 nilの場合には、まず kill されるバッファーを表示します。

25.2.3 プロジェクトの切り替え

`C-x p p` 他のプロジェクトにたいして Emacs コマンドを実行します (`project-switch-project`)。

プロジェクトファイルを処理するコマンド (Section 25.2.1 [Project File Commands], page 352 を参照) は、カレントプロジェクトがなければ、利便性のためにプロジェクトディレクトリーの入力を求めます。何らかのプロジェクト内部にいるものの別プロジェクトを操作したいときには、`C-x p p`コマンド (`project-switch-project`) を使用します。このコマンドは既知のプロジェクトルートの中からディレクトリーの選択を求めて、選択したプロジェクトを操作するために利用可能なコマンドのメニューを表示します。変数 `project-switch-commands`はメニュー内で利用できるコマンドと、各コマンドを呼び出すキーを制御します。

変数`project-list-file`は Emacs が既知プロジェクトのリストを記録するファイルを指定します。デフォルトは `user-emacs-directory`にあるファイル `projects`です (Section 33.4.4 [Find Init], page 533 を参照)。

25.2.4 プロジェクトリストファイルの管理

`M-x project-forget-project`
`project-list-file`から既知のプロジェクトを削除する。

Emacs は通常は自動的に `project-list-file`へのプロジェクトの追加と削除を行いますが、利用可能なプロジェクトを手編集したいときもあるかもしれません。`M-x project-forget-project`は利用可能なプロジェクトからプロジェクトの入力を求めて、そのプロジェクトをファイルから削除します。

25.3 変更ログ

多くのソフトウェアプロジェクトでは、変更ログ *(change log)* を管理します。これは通常、いつどのようにして、そのプログラムが変更されたかの日付順の記録を含む、`ChangeLog`という名前のファイルです。これらのファイルは、バージョンコントロールシステムに保存された変更ログエントリーから自動的に生成されたり、それらの変更ログエントリーを自動的に生成するのに使われる場合もあります。複数の変更ログファイルがあり、それぞれが 1 つのディレクトリー、またはディレクトリーツリーに対応する場合もあります。

25.3.1 変更ログコマンド

Emacs コマンド `C-x 4 a`は、編集しているファイルにたいする新しいエントリーを、変更ログファイルに追加します (`add-change-log-entry-other-window`)。そのファイルが実際にはバックアップファイルの場合、このコマンドはそのファイルの元のファイルのエントリーを適切に作成します—これはカレントバージョンから削除された関数のログエントリーを作成するとき便利です。

`C-x 4 a`は変更ログファイルを visit して、一番最近のエントリーが今日の日付であなたの名前でない場合は、新しいエントリーを作成します。これはカレントファイルにたいする、新しいアイテムも作成します。このコマンドは多くの言語にたいして、変更された関数またはその他のオブジェクトを推測することすらできます。

変更ログファイルを探すために、Emacs は編集中ファイルのディレクトリーからディレクトリー構造を上方に検索します。デフォルトでは、バージョンコントロールディレクトリーのルートと思われるディレクトリーが見つかると、検索はストップします。これを変更するには、`change-log-directory-files`をカスタマイズしてください。

変数 add-log-keep-changes-togetherが非 nilの場合、C-x 4 aは新しいアイテムを開始せず、そのファイルにたいする既存のアイテムに追加します。

同じ性質の複数の変更を 1 つにまとめることができます。最初の C-x 4 aの後にテキストを何も入力せずに、続けて C-x 4 aをタイプしていくと、他のシンボルが変更ログエントリーに追加されます。

add-log-always-start-new-recordが非 nilの場合、最後のエントリーが同じ日付のあなたによる変更だったときでも、C-x 4 aは常に新しいエントリーを作成します。

変数 change-log-version-info-enabledの値が非 nilの場合、C-x 4 aは、ファイルのバージョン番号を変更ログのエントリーに追加します。これは変数 change-log-version-number-regexp-listの正規表現を使用して、ファイルの最初の 10%から、バージョン番号を探します。

変更ログファイルは、Change Log モードで visit されます。このメジャーモードでは、グループ化されたアイテムの集まりは 1 つのパラグラフと扱われ。各エントリーはページとみなされます。これはエントリーの編集を容易にします。C-jおよび auto-fill は、新しい行を前の行と同様にインデントします。これはエントリーの内容を入力するとき便利です。

Change Log モードがオンのときには、ポイント近傍の変更ログエントリーのソース位置に移動するために、(デフォルトで C-c C-cにバインドされている) change-log-goto-sourceを使用できます。その後の next-errorコマンド (デフォルトで M-g M-nと C-x `にバインドされている) は、変更ログ間のエントリーを移動します。次の変更ログエントリーだけではなく、変更されたファイルの実際のサイトにジャンプするでしょう。ヘンコウログエントリーも戻るために previous-errorも使用できます。

コマンド M-x change-log-mergeを使用して、他のログファイルを、エントリーの日付順を保持したまま、Change Log モードのバッファーにマージできます。

バージョンコントロールシステムはプログラム変更の追跡と変更ログを維持する別の手段です。VCS を使用する多くのプロジェクトは、今日ではバージョンごとの変更ログファイルを個別に保持しないので、そのようなファイルをレポジトリ内に保有することを避けたいと思うかもしれません。add-log-dont-create-changelog-fileの値が非 nilなら、C-x 4 a (add-change-log-entry-other-window) のようなコマンドは、すでにそのようなファイルが存在しなければファイルではなく適切に命名された一時バッファーに変更を記録します。

変更ログファイルかある場合や変更ログに一時バッファーを使用する場合のいずれにおいても、それらが存在すれば関連する変更ログエントリーを挿入するために、VC Log バッファで C-c C-a (log-edit-insert-changelog) をタイプすることができます。Section 25.1.4 [Log Buffer], page 339 を参照してください。

25.3.2 ChangeLog の書式

変更ログエントリーは、現在の日付、名前 (変数 add-log-full-nameより取得)、電子メールアドレス (変数 add-log-mailing-addressより取得) を含むヘッダー行から開始されます。ヘッダー行を除いた変更ログの各行は、スペースまたはタブで開始されます。エントリーの大部分は、空白文字とアスタリスクで行が開始される、アイテム *(items)* から構成されます。以下は 2 つのアイテムおよび 1 つのアイテムをもつ、日付が 1993 年 5 月の、2 つのエントリーの例です。

```
1993-05-25  Richard Stallman  <rms@gnu.org>

        * man.el: Rename symbols 'man-*' to 'Man-*'.
        (manual-entry): Make prompt string clearer.

        * simple.el (blink-matching-paren-distance):
        Change default to 12,000.
```

```
1993-05-24  Richard Stallman  <rms@gnu.org>

        * vc.el (minor-mode-map-alist): Don't use it if it's void.
        (vc-cancel-version): Doc fix.
```

1 つのエントリーで複数の変更を記述できます。変更はそれぞれアイテム、またはアイテムの中の行を占めます。アイテムの間には通常、空行があります。アイテムが関連している場合 (異なる場所での同じ変更など)、それらの間に空行を置かずにそれらをグループ化します。

変更ログファイルの最後には、著作権表示と使用許諾を配すべきです。以下は例です:

```
Copyright 1997, 1998 Free Software Foundation, Inc.
Copying and distribution of this file, with or without modification, are
permitted provided the copyright notice and this notice are preserved.
```

これはもちろん、正しい年と版権所有者に置き換えて使う必要があります。

25.4 識別子のリファレンスを探す

識別子 *(identifier)* とは、プログラムの構文的なサブユニットの名前であり、関数 (function)、サブルーチン (subroutine)、メソッド (method)、クラス (class)、データ型 (data type)、マクロ (macro) などが該当します。プログラミング言語では、識別子はその言語の構文をもつシンボルです。識別子はタグ *(tags)* という名前でも知られています。

プログラムの開発と保守では、プロジェクト全体を横断して識別子をリネームする等から、識別子がどこで定義されているか (defined)、どこから参照されているか (referenced) を素早く見つける能力が求められます。これらの能力は、プログラミング言語をサポートするよう定義されたモード以外のメジャーモードでも、リファレンスを見つけるために有用です。たとえばテキストや TEX ドキュメントのチャプター (chapters)、セクション (sections)、アペンディクス (appendices) も同様にサブユニットとなり得るし、それらの名前も識別子として使用できます。このチャプターでは、プログラムのソース、同様に他の種類のテキストの中の、任意の種類のサブユニットの名前を正確に参照するために、“識別子 (identifiers)” という用語を使用します。

これらの能力のために、Emacs は ‘`xref`’と呼ばれる統一されたインターフェースを提供します。

`xref`が処理を行なうには、そのメジャーモード特有の情報とメソッドを使用する必要があります。どのファイルから識別子を検索するか、識別子にたいするリファレンス (references: 参照) を探す方法、識別子を補完する方法、これら (およびそれ以上のこと) は、モード固有の知識です。`xref`は処理のモード固有な部分を、そのモードにより提供されるバックエンド *(backend)* に委託します。これには、いくつかのコマンドにたいするデフォルトや、そのモード自体が提供しないモードにたいするデフォルトも含まれます。

バックエンドはこれらの能力をさまざまな方法で実装できます。いくつかの例を示します:

a. その言語のシンボルを探すために、ビルトインの方法を提供するモードがいくつかあります。たとえば Emacs Lisp のシンボルは、パッケージのロード履歴からの検索から識別され、Emacs Lisp インタープリターにより保守され、ビルトインのドキュメント文字列 (built-in documentation strings) で調べます。シンボル定義を探すために、Emacs Lisp モードは、モードのバックエンドの中の、これらの機能を使用します (この種のバックエンドの不利な点の 1 つは、インタープリターにロードされたサブユニットしか認識しないことです)。

b. カレントバッファーのプロジェクト (Section 25.2 [Projects], page 352 を参照) とメジャーモードにたいして Eglot が有効なら、Eglot はプロジェクト内の識別子の定義について外部の言語サーバープログラムに照会を行い、そのサーバーから得たデータを提供します。Section “Eglot Features” in *Eglot: The Emacs LSP Client* を参照してください。

c. 外部プログラムは関連するファイルをスキャンしてリファレンスを抽出して、これらにたいするデータベースをビルドすることができます。リファレンスをリストしたり調べるために、バックエンドは必要なときにこのデータベースにアクセスできます。Emacs のディストリビューションには `etags`が含まれています。これはプログラム中の識別子の定義にタグ付けするコマンドで、リファレンスをタグテーブル *(tags tables)* に抽出することにより、多くのプログラミング言語、および HTML のようなその他のモードをサポートします。Section 25.4.2.2 [Create Tags Table], page 365 を参照してください。`etags`によりサポートされた言語にたいするメジャーモードは、そのバックエンドの基準でタグテーブルを使用できます (この種のバックエンドの不利な点の 1 つは、タグテーブルは有効性を維持するために最新である必要があり、度々リビルドしなければならないことです)。

25.4.1 識別子を探す

このサブセクションでは識別子にたいするリファレンスを探したり、識別子にたいしてさまざまな問い合わせを行なうコマンドを説明します。リファレンスは識別子を定義 *(define)* するかもしれません (プログラム内のサブユニットを実装、ドキュメントのセクションのテキストなど)。あるいは識別子を使用 *(use)* するかもしれません (関数やメソッドの呼び出し、変数への値の割り当て、クロスリファレンスで引用されるチャプターなど)。

25.4.1.1 識別子の照合

`xref`が可能にすることの中で一番重要なのは、指定した識別子の定義を探すことです。

`M-.` 識別子の最初の定義を探します (`xref-find-definitions`)。

`C-M-.` *pattern* `RET`
*pattern*にマッチする名前の識別子を探します (`xref-find-apropos`)。

`C-x 4 . RET`
識別子の最初の定義を探して、他のウィンドウに表示します (`xref-find-definitions-other-window`)。

`C-x 5 . RET`
識別子の定義を探して、それを新しいフレームに表示します (`xref-find-definitions-other-frame`)。

`M-x xref-find-definitions-at-mouse`
マウスでクリックした識別子の最初の定義を探します。

`M-,` 前に `M-.` または同種のコマンドを呼び出した場所に戻ります (`xref-go-back`)。

`C-M-,` 前に `M-.` または同種のコマンドを呼び出した場所に戻ります (`xref-go-forward`)。

`M-x xref-etags-mode`
`etags`バックエンドを使用するように、`xref`を切り替えます。

`M-.` (`xref-find-definitions`) は、ポイント位置の識別子の定義を表示します。プレフィックス引数を与えた、またはポイント位置に識別子がない場合は、識別子の入力を求めます (常に識別子を尋ねるようにしたい場合は、`xref-prompt-for-identifier`を `t`にカスタマイズする)。

`M-.`の引数に識別子を入力する際には、既知の識別子の名前を補完候補として通常のミニバッファーの補完コマンドを使用できます (Section 5.4 [Completion], page 31 を参照)。

バッファーを切り替えるほとんどのコマンドと同様に、`xref-find-definitions`は新しいバッファーを他のウィンドウ、または新しいバッファーのために新しいフレームを作成する変種をもっていま

す。前者は C-x 4 . (xref-find-definitions-other-window)、後者は C-x 5 . (xref-find-definitions-other-frame) です。

コマンド xref-find-definitions-at-mouseは xref-find-definitionsと同じように機能しますが、マウスイベントがあった箇所周辺の識別子を探します。このコマンドはたとえばC-M-mouse-1のようなマウスイベントにバインドされることを意図しています。

コマンド C-M-. (xref-find-apropos) はtags用の apropos (Section 7.3 [Apropos], page 46 を参照) のようなコマンドです。これは指定された *regexp*にマッチする名前の選択済み tags テーブル内の識別子リストを表示します。これは M-.と似ていますが、シンボル名 (固定文字列) ではなく識別子 regexp マッチングを行う点が異なります。デフォルトでは M-.のように*xref*バッファーをポップアップしますが、変数 tags-apropos-additional-actionsをカスタマイズして追加の出力を表示できます。詳細はこの変数のドキュメントを参照してください。

上記コマンドのいずれかがマッチする 1 つ以上の定義を見つけると、デフォルトではマッチ候補を表示する*xref*バッファーをポップアップします (C-M-.は少なくともマッチを 1 つ見つけたら常に*xref*バッファーをポップアップする)。候補は通常はファイル名とそのファイル内のマッチした識別子として、そのバッファーに表示される。そのバッファーでは表示する任意の候補を選択でき、Section 25.4.1.2 [Xref Commands], page 359 で説明する追加のコマンドもあります。ただし変数 xref-auto-jump-to-first-definitionの値が moveなら、これらの中から最初の候補が*xref*バッファー内で自動的に選択されて、tや showなら最初の候補が自身のウィンドウ内に自動的に表示されます。tの場合には更に最初の候補を表示中のウィンドウの選択も行います。デフォルト値の nilでは、*xref*バッファーに候補を表示しますが、どの候補も選択しません。

定義を表示した場所に戻るには、M-, (xref-go-back) を使用します。これは最後に M-.を呼び出したポイントにジャンプします。したがって、M-.により何らかの定義を見つけて確認したら、M-,で戻ることができます。変数 xref-marker-ring-lengthで決定される深さ (デフォルトは 16) まで、M-,を用いてステップを再トレースすることができます。M-,を使えばあなたが辿ってきた場所の履歴を逐一再トレースすることによって最初に M-.を呼び出した場所、あるいはそこに至るまでの任意の場所まで遡ることができます。

M-,で戻りすぎてしまったり、あるいは戻った場所から再検証したい場合には、再度履歴を進むために C-M-, (xref-go-forward) を使うことができます。これは M-.と似ていますが、定義を調べたい識別子にいちいちポイントを移動する必要はありません。C-M-,を使えばあなたが遡ってきた場所の履歴を逐一再トレースすることによって履歴の最後の場所、あるいはそこに至るまでの任意の場所まで辿ることができます。

いくつかのメジャーモードは、特定の識別子の検索に失敗するかもしれない xrefサポート機能をインストールするかもしれません。たとえば、Emacs Lisp モード (Section 24.9 [Lisp Eval], page 329 を参照) では、デフォルトでは M-.はカレント Emacs セッションにロードされた、あるいは自動ロード (Section “Autoload” in *The Emacs Lisp Reference Manual* を参照) される Lisp パッケージの関数と変数だけを検索します。M-.が何らかの識別子の検索に失敗する場合は、xrefに etagsバックエンド (Section 25.4 [Xref], page 357 を参照) の使用を強制することができます。これを行うには、M-x xref-etags-mode を呼び出して Xref Etags マイナーモードをオンに切り替えて、再度 M-.を呼び出します (これが機能するためには、ソースファイルのディレクトリーツリー内で tag テーブルを作成するために、必ず etagsを実行すること。Section 25.4.2.2 [Create Tags Table], page 365 を参照されたい)。

25.4.1.2 *xref*バッファーで利用可能なコマンド

以下のコマンドは*xref*バッファーの XREF モードにより提供されるコマンドです:

RET
mouse-1 カレント行のリファレンスを表示します (xref-goto-xref)。プレフィックス引数を指定すると*xref*バッファーも隠し (bury) ます。

mouse-2 mouse-1と同様ですが*xref*を表示中のウィンドウを選択されたウィンドウにします (xref-select-and-show-xref)。

n
. 次のリファレンスに移動して、それを別のウィンドウに表示します (xref-next-line)。

N 次のリファレンスグループの最初のリファレンスに移動して、別ウィンドウに表示します (xref-next-line)。

p
, 前のリファレンスに移動して、それを別のウィンドウに表示します (xref-prev-line)。

P 前のリファレンスグループの最初のリファレンスに移動して別ウィンドウに表示します (xref-prev-line)。

C-o カレント行のリファレンスを別のウィンドウに表示します (xref-show-location-at-point)。

r *pattern* RET *replacement* RET
*pattern*にマッチするリファレンスにたいしてインタラクティブな問い合わせつき置換 (query-replace) を行ない、マッチを *replacement*に置換します。このコマンドは関係のあるすべてのファイルにおける識別子への全マッチを表示している*xref*バッファーでのみ使用できます。Section 25.4.1.3 [Identifier Search], page 360 を参照してください。

g *xref*バッファーのコンテンツをリフレッシュします (xref-revert-buffer)。

M-, *xref*バッファーを表示しているウィンドウを quit して、Xref の前のスタック位置にジャンプします (xref-quit-and-pop-marker-stack)。

q *xref*バッファーを表示しているウィンドウを quit します (xref-quit)。

これらに加えて、リファレンスを表示せずにバッファー内を移動するために、C-nや C-pのような、通常のナビゲーションコマンドも利用可能です。

25.4.1.3 識別子の検索と置換

このセクションのコマンドは、識別子自身、またはファイル内の識別子にたいするリファレンスにたいして、様々な検索と置換を行ないます。

M-? ポイント位置の識別子にたいする、すべてのリファレンスを探します。

r
M-x xref-query-replace-in-results RET *replacement* RET
C-u M-x xref-query-replace-in-results RET *regexp* RET *replacement* RET
*xref*バッファーに表示されているすべての識別子の名前にたいして、*regexp*を *replacement*にインタラクティブに置換します。

M-x xref-find-references-and-replace RET *from* RET *to* RET
識別子 *from*のすべてのインスタンスを、新たな名前 *to*にインタラクティブにリネームします。

`M-x tags-search` RET *regexp* RET
: 選択されたタグテーブルのファイルから、*regexp*を検索します。

`M-x tags-query-replace` RET *regexp* RET *replacement* RET
: 選択されたタグテーブルの各ファイルにたいして、`query-replace-regexp`を実行します。

`M-x fileloop-continue`
: ポイントのカレント位置から、上記コマンドの最後の 2 つを再開します。

`M-?`は、ポイント位置の識別子にたいして、必要に応じて補完つきで識別子の入力を促しつつ、すべてのリファレンスを探します。カレントのバックエンド (Section 25.4 [Xref], page 357 を参照) に依存して、ポイント位置に有効な識別子を見つけた場合でも、このコマンドは識別子の入力を促すかもしれません。プレフィクス引数が指定された場合、このコマンドは常に識別子の入力を求めます (常に入力を求めるようにしたい場合は `xref-prompt-for-identifier`を `t`、ポイント位置に利用できる識別子がない場合のみ入力を求めるようにするには `nil`にカスタマイズすればよい)。それからこのコマンドは、その識別子にたいするすべてのリファレンスについて、ファイル名と識別子が参照されている行を、`*xref*`バッファーに表示します。このバッファーでは XREF モードのコマンドが利用可能です。Section 25.4.1.2 [Xref Commands], page 359 を参照してください。

変数 `xref-auto-jump-to-first-xref`の値が `t`なら、`xref-find-references`は最初の結果に自動的にジャンプして、それを表示するウィンドウを選択します。値が `show`ならまず結果を表示しますが、`*xref*`バッファーを表示するウィンドウが選択されたままになります。値が `move`なら`*xref*`バッファーで最初の結果を選択しますが表示はしません。デフォルト値の `nil`では`*xref*`バッファーに結果を表示するだけで、結果を何も選択しません。

`r` (`xref-query-replace-in-results`) は通常の `M-x query-replace-regexp`のように、*replacement*文字列を読み取ります。そして`*xref*`バッファーに表示されたその識別子を参照するすべてのファイル、すべての箇所において、その識別子を新たな名前 *replacement*に置き換えます。これはリファクタリングの一環として識別子の名称を変更する際に役に立つでしょう。このコマンドは `M-?`で生成された`*xref*`バッファーで呼び出す必要があります。このコマンドはデフォルトでは識別子の名前全体を *replacement*で置き換えますが、プレフィックス引数とともに呼び出した場合には識別子の名前にマッチさせる regexp の入力を求めて、識別子の名称から regexp にマッチした部分だけを *replacement*で置き換えます。

`M-x xref-find-references-and-replace`は `xref-query-replace-in-results`と似ていますが、これは名前 *from*で指定される単一の識別子をリネームしたい場合に特に便利なコマンドです。

`M-x tags-search`は、ミニバッファーを使用して regexp を読み取り、選択されたタグテーブルのすべてのファイルから、1 ファイルずつマッチを検索します。これは検索しているファイル名を表示するので、進行状況を確認することができます。マッチが見つかった場合、`tags-search`はリターンします。このコマンドには利用可能なタグテーブル (Section 25.4.2 [Tags Tables], page 363 を参照してください) が必要です。

`tags-search`で 1 つのマッチが見つかったら、おそらく残りのすべてについても検索したいと思うでしょう。`M-x fileloop-continue`は、多くのマッチを探すために `tags-search`を再開します。これはカレントバッファーの残りの部分を検索して、その後タグテーブルの残りのファイルを検索します。

`M-x tags-query-replace`は、タグテーブルのすべてのファイルにたいして、1 つの `query-replace-regexp`を実行します。これは、通常の `M-x query-replace-regexp`と同様、検索する regexp と、それを置換する文字列を読み取ります。この検索はむしろ `M-x tags-search`

に似ていますが、入力へのマッチを繰り返し処理します。問い合わせ付き置換については、Section 12.10.4 [Query Replace], page 124 を参照してください。

変数 tags-case-fold-searchの値をカスタマイズすることにより、タグ検索コマンドの大文字小文字の扱いを制御できます。デフォルトには、case-fold-searchの値と同じ設定が使用されます (Section 12.9 [Lax Search], page 120 を参照してください)。

1 回の M-x tags-query-replaceの呼び出しで、タグテーブルのすべてのファイルを検索することが可能です。しかし、一時的に検索を抜けられると便利なときもあります。これは、問い合わせ付き置換として特別な意味をもたない入力イベントで行うことができます。つづけて問い合わせ付き置換を再開するには、M-x fileloop-continueとタイプします。このコマンドは、最後のタグ検索または置換コマンドを再開します。たとえばカレントファイルの残りをスキップするには、M-> M-x fileloop-continue とタイプします。

上記で説明したコマンドは、xref-find-definitions系の検索より広範な検索を行うことに注意してください。xref-find-definitionsコマンドは、部分文字列または正規表現にマッチする識別子の定義だけを検索します。コマンド xref-find-references、tags-search、tags-query-replaceは、通常の検索および置換コマンドがカレントバッファーにたいして行うように、識別子または regexp にたいするマッチを検索します。

xref-find-referencesや tags-searchのかわりに、サブプロセスとして grepを実行して、Emacs にマッチした行を 1 つずつ表示させることができます。Section 24.4 [Grep Searching], page 313 を参照してください。

25.4.1.4 識別子の照会

C-M-i
M-TAB
: タグテーブルがロードされているときは、できるだけ選択されたタグテーブルを使って、ポイント周囲のテキストの置換を行います (completion-at-point)。

M-x list-tags RET *file* RET
: プログラムファイル *file*で定義されている識別子のリストを表示します。

C-M-. *regexp* RET
: *regexp*にマッチする、すべての識別子のリストを表示します (xref-find-apropos)。Section 25.4.1.1 [Looking Up Identifiers], page 358 を参照してください。

M-x tags-next-file
: 選択されたタグテーブルに記録されているファイルを visit します。

プログラミング言語のモードのほとんどでは、C-M-iまたは M-TAB (completion-at-point) とタイプして、ポイント位置のシンボルを補完できます。このコマンドのために、モードに特化した補完候補を提供するモードもあります。補完候補を提供しないモードでは、選択されたタグテーブルがある場合、補完候補を生成するためにこのコマンドを使用することができます。Section 23.8 [Symbol Completion], page 302 を参照してください。

M-x list-tagsは、選択されたタグテーブルでカバーされたファイルの名前を 1 つ読み取り、そのファイルで定義されたタグのリストを表示します。タグテーブルに記録されたファイル名にディレクトリーが含まれない場合は、ファイル名にディレクトリーを含めないでください。このコマンドはバックエンドが etags のときだけ機能し、そのプロジェクトのために利用可能なタグテーブルが必要です。Section 25.4.2 [Tags Tables], page 363 を参照してください。インタラクティブに使用すると、カレントバッファーのファイル名がデフォルトタグとして使用されます。

M-x tags-next-fileは、選択されたタグテーブルでカバーされるファイルを visit します。最初に呼び出したとき、テーブルでカバーされた最初のファイルを visit します。続けて呼び出すことによ

り、次のカバーされたファイルを visit していきます。プレフィクス引数を指定した場合、最初のファイルに戻ります。このコマンドには、選択されたタグテーブルが必要です。

25.4.2 tags テーブル

タグテーブル *(tags table)* は、特定のプログラムまたはドキュメントのソースコードをスキャンすることにより抽出されたタグ [1] を記録します。生成されたファイルから抽出されたタグは、タグ抽出の際にスキャンされる生成されたファイルではなく、その元になるファイルを参照します。生成されたファイルの例として、Cweb ソース、Yacc パーサー、Lex スキャナー定義から生成された C ファイルや、プリプロセスされた C ファイルの`.i`、`.fpp`ソースファイルをプリプロセスすることにより生成される Fortran ファイルがあります。

タグテーブルを生成するには、ドキュメントまたはソースコードファイルにたいして、シェルコマンド`etags`を実行します。‘`etags`’プログラムは、タグテーブルファイル *(tags table file)*、または略記してタグファイル *(tags file)* にタグを書き込みます。タグファイルは慣習的に`TAGS`という名前です。Section 25.4.2.2 [Create Tags Table], page 365 を参照してください (同じフォーマットでこのようなテーブルを生成できる他のコマンドを使用して、タグテーブルを作成することも可能です)。

Emacs は、`xref`にたいするサポートされたバックエンドとして、`etags`パッケージを通じてタグテーブルを使用します。タグテーブルは Emacs ディストリビューションの一部である`etags`コマンドにより生成されるので、ここではタグテーブルについて、より詳細に説明します。

Ebrowse 機能は`etags`に似ていますが、C++に特化したものです。Section “Ebrowse” in *Ebrowse User's Manual* を参照してください。Semantic パッケージは、`etags`機能とは別の、タグを生成して使用する他の方法を提供します。Section 23.10 [Semantic], page 303 を参照してください。

25.4.2.1 ソースファイルタグの構文

以下は、もっともポピュラーな言語でタグ構文が定義される方法です:

- C コードでは、C の関数や typedef はタグなので、`struct`、`union`、`enum`の定義もタグです。タグテーブルを作成するとき、‘`--no-defines`’を指定しなければ、`#define`マクロ定義、`#undef`および`enum`定数もタグになります。同様に、‘`--no-globals`’を指定しなければグローバル変数もタグで、‘`--no-members`’を指定していなければ構造体のメンバーもタグです。‘`--no-globals`’、‘`--no-defines`’、‘`--no-members`’を使用することにより、タグテーブルを小さくすることができます。

  `etags`に‘`--declarations`’オプションを与えることにより、関数定義 (function definitions) に加えて、関数宣言 (function declarations) と外部変数 (external variables) もタグ付けできます。

- C++コードでは、C コードのすべてのタグ構成に加えて、メンバー関数も認識されます。‘`--no-members`’オプションを使用しなければ、メンバー変数も認識されます。`operator`定義は、‘`operator+`’のような名前をもちます。‘`--class-qualify`’オプションを指定した場合、クラスの変数および関数にたいするタグは、‘*`class`*`::`*`variable`*’および‘*`class`*`::`*`function`*’という名前になります。デフォルトでは、クラスのメソッドとメンバーはクラス修飾されていません。これにより、ソース内でより正確に、それらの名前が識別可能になります。

[1] *tag* は、識別子リファレンスの同義語です。`etags`パッケージにもとづいたコマンドおよび機能では、伝統的に “tag” という用語をこの意味に使用します。以下のサブセクションでは、この伝統にしたがいます。

- Java コードでは、C++で認識されるのすべてのタグ構成に加えて、`interface`、`extends`、`implements`もタグとして認識されます。クラスの変数および関数にたいするタグは、‘*class.variable*’および‘*class.function*’という名前になります。
- LaTeX ドキュメントでは、`\chapter`、`\section`、`\subsection`、`\subsubsection`、`\eqno`、`\label`、`\ref`、`\cite`、`\bibitem`、`\part`、`\appendix`、`\entry`、`\index`、`\def`、`\newcommand`、`\renewcommand`、`\newenvironment`、`\renewenvironment`にたいする引数がタグになります。

  `etags`を呼び出す前に、環境変数`TEXTAGS`で指定することにより、他のコマンドも同様にタグにできます。この環境変数の値には、コロンで区切られたコマンド名のリストを指定します。たとえば、

```
TEXTAGS="mycommand:myothercommand"
export TEXTAGS
```

  これは、(Bourne シェルの構文の使用して) コマンド‘`\mycommand`’と‘`\myothercommand`’もタグとして定義します。
- Lisp コードでは、`defun`で定義された任意の関数、`defvar`および`defconst`で定義された任意の変数、および一般的に列 0 から‘`(def`’で始まる任意の式の最初の引数はタグです。例外として`(defvar` *foo*`)`という形式の式は、宣言として扱われ、‘`--declarations`’オプションが与えられたときだけタグになります。
- Scheme コードでは、`def`で定義されたすべて、または名前が‘`def`’で始まる構成がタグに含まれます。これらは、ファイルのトップレベルで`set!`でセットされる変数も含まれます。

他の言語もいくつかサポートされます:

- Ada コードでは、関数 (functions)、プロシージャー (procedures)、パッケージ (packages)、タスク (tasks)、タイプ (types) がタグです。‘`--packages-only`’オプションを使用することにより、タグをパッケージにたいしてだけ作成できます。

  Ada では、異なる種類のエンティティー (たとえば関数とプロシージャー) に、同じ名前を使うことができます。またパッケージ、プロシージャー、関数と似たものに、スペック (spec、たとえば interface) およびボディー (body、たとえば implementation) があります。欲しい定義を簡単に取り出すために、Ada のタグ名にはエンティティーのタイプを示す接尾辞がつきます:

  ‘`/b`’ パッケージのボディー (package body)。

  ‘`/f`’ 関数 (function)

  ‘`/k`’ タスク (task)。

  ‘`/p`’ プロシージャー (procedure)。

  ‘`/s`’ パッケージのスペック (package spec)。

  ‘`/t`’ タイプ (type)。

  したがって、`M-x find-tag RET bidule RET`は単に`bidule`という任意のタグを検索しますが、`M-x find-tag RET bidule/b RET`は直接パッケージ`bidule`のボディーに移動します。
- アセンブラーコードでは、行の開始に現れ、後にコロンが続くラベルがタグです。
- Bison または Yacc の入力ファイルでは、各構文規則で定義する非終端記号がタグです。ファイル内に含まれる C コードの部分は、C コードとして解析します。
- Cobol コードでは、タグはパラグラフ名なので、列 8 から始まり、後にピリオドが続く任意の単語がタグです。

- Erlang コードでは、ファイルで定義された関数 (functions)、レコード (records)、マクロ (macros) がタグです。
- Fortran コードでは、サブルーチン (subroutines) およびブロックデータ (block data) がタグです。
- Go コードでは、関数 (functions)、タイプ (types) がタグです。
- HTML 入力ファイルでは、`title`、および`h1`、`h2`、`h3`ヘッダーがタグです。アンカー内の`name=`、およびすべての`id=`もタグです。
- Lua 入力ファイルでは、すべての関数 (functions) がタグです。
- makefile では、ターゲット (targets) がタグで、‘`--no-globals`’を指定しなければ変数 (variables) もタグです。
- Objective C コードでは、クラスにたいする Objective C 定義、クラスカテゴリー (class categories)、メソッド (methods)、プロトコル (protocols) が含まれます。クラスの変数および関数にたいするタグの名前は、‘*class*`::`*variable*’および‘*class*`::`*function*’になります。
- Pascal コードでは、ファイル内で定義された関数およびプロシージャーがタグです。
- Perl コードでは、パッケージ、サブルーチン、変数がタグで、キーワード`package`、`sub`、`use constant`、`my`、`local`で定義されます。グローバル変数をタグ付けしたい場合、‘`--globals`’を使用します。サブルーチンにたいするタグの名前は、‘*package*`::`*sub*’になります。デフォルトのパッケージで定義されたサブルーチンの名前は、‘`main::`*sub*’になります。
- PHP コードでは、関数 (functions)、クラス (classes)、定義 (defines) がタグです。‘`--no-members`’オプションを使用しなければ、変数 (vars) もタグです。
- PostScript コードでは、関数がタグです。
- Prolog コードでは、行頭の述語 (predicates) とルール (rules) がタグです。
- Python コードでは、行頭の`def`および`class`はタグを生成します。
- Ruby コードでは、行の先頭にある`def`、`class`、`module`はタグを生成します。定数もタグを生成します。
- Rust コードでは`fn`、`enum`、`struct`、`macro_rules!`で定義されているものはすべてタグです。

他の書式や言語を扱うために、regexp にたいするマッチにもとづいてタグを生成することもできます (Section 25.4.2.3 [Etags Regexps], page 367 を参照してください)。

25.4.2.2 タグテーブルの作成

`etags`プログラムは、タグテーブルファイルを作成するために使用されます。このコマンドは、前のセクションで説明している複数の構文を理解します。以下は`etags`を実行する方法です:

```
etags inputfiles...
```

`etags`プログラムは、指定されたファイルを読み込んで、カレント作業ディレクトリーの`TAGS`という名前のファイルに、タグテーブルを書き込みます。‘`--output=`*file*’オプションを使用して、タグテーブルに異なる名前のファイル名を指定することもできます。ファイル名に`-`を指定すると、タグテーブルを標準出力に出力します。‘`--append`’オプションを使用して、既存のファイルに新たに作成した tag テーブルを追加することもできます。

指定されたファイルが見つからない場合、`etags`はそれらの圧縮されたバージョンを探して、それらを解凍して読み込みます。MS-DOS では、コマンドラインに‘`mycode.c`’が与えられ、‘`mycode.c`’が存在しないとき、`etags`は`mycode.cgz`のような名前のファイルを探します。

ファイルの内容が変更されてタグテーブルが古くなったときは、`etags`を再び実行することにより、タグテーブルを更新できます。タグテーブルにタグが記録されていなかったり、間違ったファイ

ルにたいして記録している場合、タグテーブルを更新するまで、Emacs はそれの定義を見つけることができません。しかしタグに記録されている位置が、(編集により) 少し間違っているようなときは、少しの遅れは生じますが Emacs は正しい位置を見つけることができます。

したがって、編集するたびにタグテーブルを更新する必要はありません。リストしたい新しいタグを定義したときや、タグ定義をあるファイルから他のファイルへ移動したとき、または大幅な変更を施したときは、タグテーブルを更新するべきです。

etagsに '`--include=`*file*'オプションを渡すことにより、タグテーブルに他のタグテーブルをインクルード *(include)* できます。これによりインクルードされたタグファイルでカバーされる、すべてのファイルをカバーできます。

etagsを実行するとき、ソースファイルを相対ファイル名で指定した場合、タグファイルには、そのタグファイルが最初に書き込まれたディレクトリーにたいする相対ファイル名が含まれます。この方法を使えば、ディレクトリーツリー全体を移動しても、タグファイルは正しくソースファイルを参照します。しかしタグファイルが`-`または`/dev`のときは、ファイル名はカレント作業ディレクトリーにたいする相対ファイル名になります。これはタグを標準出力に書き込むときに便利です。

相対ファイル名を使う場合、違うディレクトリーにあるタグファイルを指すシンボリックリンクを指定するべきではありません。なぜならこれは一般的にファイル名を無効にするからです。

etagsの引数に絶対ファイル名を指定した場合、タグファイルには絶対ファイル名が含まれます。この方法では、ソースファイルが同じ場所にある限り、タグファイルを移動してもタグファイルは同じ名前を参照します。絶対ファイル名は '`/`'で開始されるか、MS-DOS および MS-Windows では '*device*`:/`'で開始されます。

非常に大きな数のファイルからタグテーブルを作成したい場合、それをコマンドラインに指定すると問題が発生するかもしれません。なぜならコマンドライン引数の長さに制限のあるシステムもあるからです。この制限は、以下のようにファイル名の場所にダッシュを指定して、etagsにファイル名を標準入力から読み込むように指示して回避することができます。

```
find . -name "*.[chCH]" -print | etags -
```

etagsはファイル名とファイル内容にもとづいて、入力ファイルで使用されている言語を認識します。これは最初に、特定の言語にたいして一般的に使用されるファイル名と拡張子にたいするマッチを試みます。いくつかの言語には、既知の名前のインタープリター (Perl の `perl`、Prolog の `pl` など) があるので、etagsは次に入力ファイルの最初の行でインタープリター指定 '`#!`*interp*'を調べて、既知のインタープリターとのマッチを行います。これが失敗、または言語の自動検知をオーバーライドしたい場合は、'`--language=`*name*'オプションで、明示的に言語を指定できます。このオプションはファイル名に混ぜることができます。各指定はその後に続くファイル名に適用されます。'`--language=auto`'の指定は、ファイル名とファイル内容から言語を推測するよう etagsに指示します。'`--language=none`'を指定すると、言語に特有の処理を完全にオフに切り替えます。この場合、etagsは regexp のマッチングだけでタグを認識します (Section 25.4.2.3 [Etags Regexps], page 367 を参照してください)。これは、etagsがまだサポートしていない言語を使用するファイルが入力の際に、etagsがデフォルト言語として Fortran および C にフォールバックするのを抑止します。

オプション '`--parse-stdin=`*file*'は、etagsをプログラムから呼び出すときに便利です。これは、(1 回だけ) コマンドラインからファイル名を読み取るとき使用できます。etagsは標準入力から読み取り、生成されたタグがファイル *file*に属するとマークします。

GNU コーディング規約を遵守しないソースファイルにおいて、関数や`struct`定義のようなトップレベルの定義だけに許されている列 0 の大カッコ ('`{`'および '`}`') が存在するようなら、etagsが列 0 にある閉じ大カッコを誤って解釈することを防ぐためにオプション '`--ignore-indentation`'を使うことをお勧めします。

‘etags --help’オプションは etagsが認識する言語と、言語を推測するためのファイル名ルールのリストを出力します。これは利用可能な etagsオプションと、簡単な説明のリストも出力します。このオプションの後に、1 つ以上の ‘--language=*lang*’を指定すると、*lang*にたいするタグの生成方法の詳細を出力します。

25.4.2.3 etags の regexp

‘--regex’オプションは、正規表現のマッチにもとづいて etagsがタグを認識できるようにします。このオプションはファイル名と混ぜることができます。オプションは、それぞれのオプション後に続くソースファイルに適用されます。複数の ‘--regex’を指定した場合、それらすべては並列に使用されます。構文は以下のとおりです:

```
--regex=[{language}]/tagregexp/[nameregexp/]modifiers
```

オプション値の肝心な部分は *tagregexp*で、これはタグにマッチする regexp です。これは常に位置が固定されており、行の開始だけにマッチします。インデントされたタグの場合、最初の空白文字にマッチさせるために、‘[\t]*’で始まる regexp を使用します。

これらの正規表現では、‘\’は次の文字をクォートします。また C のエスケープ文字シーケンスのすべて、すなわち ‘\a’(bell)、‘\b’(back space)、‘\e’(escape)、‘\f’(formfeed)、‘\n’(newline)、‘\r’(carriage return)、‘\t’(tab)、and ‘\v’(vertical tab) がサポートされます。これらに加えて、‘\\d’は DEL文字を意味します。

理想的には、*tagregexp*はタグとして認識させるのに必要な文字以上にマッチさせるべきではありません。構文がそれを求める場合、タグより多くの文字にマッチする *tagregexp*を記述して、そのマッチからタグだけをピックアップするために、*nameregexp*を追加するべきです。これは Emacs がより正しくタグを見つけて、タグ名の補完をより確実にすることを可能にします。*nameregexp*内においてはこれは、*tagregexp*内でのカッコによるグループ化にたいする “後方参照 (back references)” (Section 12.7 [Regexp Backslash], page 118 を参照) として有用であり、頻繁に使用されます。たとえば、‘\\1’はそのようなカッコによる最初のグループを参照します。以下でいくつかの例を見つけることができるでしょう。

modifiers(修飾子) は、etagsがマッチングを行う方法を変更するための 0 文字以上の文字シーケンスです。修飾子がない regexp は、大文字小文字を区別する方法で、入力ファイルの各行にたいして順番に適用されます。修飾子とその意味は以下のとおりです:

‘i’ この regexp のマッチングで、大文字小文字を無視します。

‘m’ この正規表現はファイル全体にマッチするので、複数行のマッチが可能です。

‘s’ この正規表現はファイル全体にマッチし、*tagregexp*内の ‘.’は改行にマッチします。

‘-R’オプションは、それの前に ‘--regex’で定義された regexp をすべて取り消します。これは後に続くファイル名にも適用されます。以下は例です:

```
etags --regex=/reg1/i voo.doo --regex=/reg2/m \
    bar.ber -R --lang=lisp los.er
```

この例では、etagsは voo.dooと bar.berにたいして。ファイル内容に一致する解析用の言語を選択します。etagsは、voo.doo内の追加のタグを認識するために *reg1*も使用し、bar.ber内の追加のタグを認識するために *reg1*と *reg2*の両方を使用します。voo.dooと bar.berの各行にたいして大文字小文字を区別せずに *reg1*がチェックされ、bar.berのファイル全体にたいして大文字小文字を区別して *reg2*がチェックされ、これは複数行へのマッチが許されます。los.erのタグの認識にはユーザー指定の regexp マッチは行わず、Lisp のタグルールだけが使用されます。

オプションのプレフィクス {*language*}を使用して、与えられた言語だけに ‘--regex’オプションを制限できます (‘etags --help’で etagsが認識する言語のリストが表示されます)。これはファイ

ルに、etagsにたいして事前に定義された多くの正規表現が含まれている場合に便利です。以下の例は、Emacs の C 言語のソースファイルの DEFVARマクロにたいするタグです:

```
--regex='{c}/[ \t]*DEFVAR_[A-Z_ \t(]+"\([^"]+\)"/\1/'
```

正規表現が複雑な場合、そのリストをファイルに保存することができます。以下のオプション構文は、etagsに正規表現が保存された 2 つのファイルを指示します。2 つ目のファイルに含まれる正規表現は、大文字小文字を区別せずにマッチします。

```
--regex=@case-sensitive-file --ignore-case-regex=@ignore-case-file
```

etagsにたいする regex ファイルは、行ごとに 1 つの正規表現を含みます。空行およびスペースかタブで始まる行は無視されます。表の開始が '@'の場合、etagsはその行の残りを他の正規表現ファイルとみなすので、そのようなファイルを他のファイルをインクルードできます。他のすべての行は正規表現です。最初の非空白文字が '--'の場合、その行はコメントです。

たとえば、以下の内容の 'emacs.tags'という名前のファイルを作成できます:

```
        -- This is for GNU Emacs C source files
{c}/[ \t]*DEFVAR_[A-Z_ \t(]+"\([^"]+\)"/\1/
```

これは以下のように使用します:

```
etags --regex=@emacs.tags *.[ch] */*.[ch]
```

さらに例を示しましょう。regexp はシェルから解釈され内容にクォートされています。

- Octave ファイルのタグ:

```
etags --language=none \
      --regex='/[ \t]*function.*=[ \t]*\([^ \t]*\)[ \t]*(/\1/' \
      --regex='/###key \(.*\)/\1/' \
      --regex='/[ \t]*global[ \t].*/' \
      *.m
```

  タグはスクリプトにたいして生成されるので、そのスクリプトにジャンプしたいときは、あなた自身で '###key *scriptname*'という形式の行を追加する必要があることに注意してください。

- Tcl ファイルのタグ:

```
etags --language=none --regex='/proc[ \t]+\([^ \t]+\)/\1/' *.tcl
```

- VHDL ファイルのタグ:

```
etags --language=none \
  --regex='/[ \t]*\(ARCHITECTURE\|CONFIGURATION\) +[^ ]* +OF/' \
  --regex='/[ \t]*\(ATTRIBUTE\|ENTITY\|FUNCTION\|PACKAGE\
  \( BODY\)?\|PROCEDURE\|PROCESS\|TYPE\)[ \t]+\([^ \t(]+\)/\3/'
```

25.4.3 タグテーブルの選択

Emacs は常に、選択されたタグテーブルを最大で 1 つもちます。タグテーブルにたいして機能するすべてのコマンドは、選択されたタグテーブルをまず使用します。タグテーブルを選択するには引数としてタグテーブルファイル名を読み取る M-x visit-tags-tableをタイプします。この引数のデフォルトはデフォルトディレクトリーからディレクトリーを上方へ再帰的に検索して TAGSという名前のファイルを含む最初のディレクトリーです。

Emacs は、タグテーブルを使用するまでは実際にタグテーブルの内容を読み込みません。visit-tags-tableが行うのは、ファイル名を変数 tags-file-nameに格納することだけです。この変数の初期値は nilです。この変数の値は、タグテーブルにたいして機能するすべてのコマンドに、使用するタグテーブルファイル名を知らせます。

選択されたタグテーブルに加えて、Emacs は共に使用するいくつかのタグテーブルのリストを保守しています。たとえばライブラリーを使うプログラムを扱っているなら、Emacs が両方から容易に識別子を見つけられるように、そのプログラムの他にも利用可能なライブラリーのタグテーブルも欲

しいかもしれません。識別子が選択されたタグテーブルになかったり、タグコマンドに必要となるソースファイルにたいする記述がない場合には、このコマンドはタグテーブルのカレントリストから他のすべてのタグテーブルの使用を試みます。

他のタグテーブルがすでにロードされているときに `visit-tags-table`を使用した場合には選択肢が与えられます。つまり、タグテーブルのカレントリストに新しいタグを追加するか、あるいはカレントリストを破棄して新しいリストを開始することもできます。タグコマンドは、カレントリストのすべてのタグテーブルを使用します。新しいリストを開始した場合、他のもののかわりに、新しいタグテーブルが使用されます。カレントリストに新しいタグテーブルを追加した場合、それは他のものと同じように使用されます。

以下のようにして、変数 `tags-table-list`にディレクトリー名のリストをセットすることにより、タグテーブルのリストを正確に指定できます:

```
(setq tags-table-list
      '("~/.emacs.d" "/usr/local/lib/emacs/src"))
```

This tells the tags commands to look at the `TAGS` files in your `~/.emacs.d` directory and in the `/usr/local/lib/emacs/src` directory. The order depends on which file you are in and which tags table mentions that file.

`tags-file-name`と `tags-table-list`の両方をセットしてはいけません。

25.5 Emacs 開発環境

EDE(*Emacs Development Environment: Emacs* 開発環境) は、Emacs での大きなプログラムの作成、ビルド、デバッグなどのタスクを単純化するパッケージです。これは Emacs において、IDE(*Integrated Development Environment:* 統合開発環境) の機能をいくつか提供します。

このセクションは、EDE の簡単な説明を提供します。EDE の完全な詳細は、`C-h i`とタイプして EDE のマニュアルを選択してください。

EDE は、グローバルなマイナーモードとして実装されています (Section 20.2 [Minor Modes], page 241 を参照してください)。有効にするには `M-x global-ede-mode`とタイプするか、'`Tools`' メニューの '`Project Support (EDE)`'アイテムをクリックします。以下の行を init ファイルに追加することにより、Emacs 開始時に EDE を有効にすることもできます。

```
(global-ede-mode t)
```

EDE を有効にすることにより、メニューバーに '`Development`'という名前のメニューが追加されます。以下で説明するコマンドを含めて、多くの EDE コマンドをこのメニューから呼び出すことができます。

EDE は、ファイルをプロジェクト *(projects)* に編成します。プロジェクトはディレクトリーに対応します。プロジェクトルート *(project root)* は、プロジェクトの最上層のディレクトリーです。新しいプロジェクトを定義するには、プロジェクトルートのファイルを visit して、`M-x ede-new`とタイプします。このコマンドはプロジェクトタイプ *(project type)* の入力を求めます。これは EDE がプロジェクトを背後で管理する方式です (Section "Creating a project" in *Emacs Development Environment* を参照してください)。もっとも一般的なプロジェクトタイプは、Makefiles を使用する '`Make`'、および GNU Automake(*Automake* を参照してください) を使用する '`Automake`'です。どちらの場合も、EDE はプロジェクトに関する情報を格納する、`Project.ede`という名前のファイルを作成します。

プロジェクトには、1 つ以上のターゲット *(targets)* を含めることができます。ターゲットとは、プロジェクトの 1 つ以上のファイルからビルドされるオブジェクトファイル、実行ファイル、またはその他の種類のファイルです。

プロジェクトに新しいターゲット *(target)* を追加するには、C-c . t (M-x ede-new-target) とタイプします。このコマンドは、カレントファイルをそのターゲットに追加するか尋ねます。これはターゲットがそのファイルからビルドされることを意味します。ターゲットを定義した後は、C-c . a (ede-add-file) とタイプすることにより、ターゲットにファイルを追加することができます。

ターゲットをビルドするには、C-c . c (ede-compile-target) とタイプします。プロジェクトのすべてのターゲットをビルドするには、C-c . C (ede-compile-project) とタイプします。EDE はターゲットがどのようにビルドされるべきか推測するために、ファイルタイプを使用します。

25.6 バグリファレンス

ある程度のユーザーをもつプロジェクトのほとんどは、バグレポートを追跡するために一意で短い数値や識別子を割り当てる何らかの問題追跡ソフトウェアでバグレポートを追跡しています。これらはソースコード中でバグの修整コードのコメント、ドキュメントファイル、メーリングリストや IRC チャンネルでの議論において、与えられたバグにたいするリファレンスとして使用されます。

マイナーモード bug-reference-modeと bug-reference-prog-modeはそのようなバグリファレンスをハイライトして、対応するバグレポートをプロジェクトの issue tracker へフォローできるようにします。bug-reference-prog-modeは、ソースコードのコメントおよび文字列内部のバグリファレンスだけをハイライトする bug-reference-modeの変種です。

これが機能するためには、バグリファレンスの構文 (bug-reference-bug-regexp)、およびバグレポートを照会できるトラッカーの URL(bug-reference-url-format) を Bug Reference モードが承知している必要があります。これらは通常はプロジェクト間で異なるので、Section 33.2.5 [Directory Variables], page 516 や Section 33.2.4 [File Variables], page 512 で指定するのは理にかなっています。

たとえば自分たちのプロジェクトではバグレポートへのリファレンスを通常は bug#1234 や Bug-1234 のように記述し、そのバグの issue トラッカーにおけるページが https://project.org/issues/1234なら、これらのローカル変数セクションは以下のようになります。

```
;; Local Variables:
;; bug-reference-bug-regexp: "\\([Bb]ug[#-]\\([0-9]+\\)\\)"
;; bug-reference-url-format: "https://project.org/issues/%s"
;; End:
```

最初の regexp グループがキャプチャーした文字列は、オーバーレイ bug-reference の境界 (ハイライトされてクリック可能になる部分) の作成を定義します。

bug-reference-bug-regexpの 2 つ目の regexp グループがキャプチャした文字列は、bug-reference-url-format内の%sテンプレートの置換に使用されます。

issue とマージリクエストが違う URL になるように、これらの区別にバグリファレンスの異なる部分を使用する必要がある等、より複雑なシナリオに対応するために bug-reference-url-format は関数でもよいことに注意してください。

自動セットアップ

bug-reference-modeがアクティブな場合には、bug-reference-mode-hookが実行して、bug-reference-bug-regexpと bug-reference-url-formatをどちらもセットせず、変数をセットできる関数が見つかるまで bug-reference-auto-setup-functions内の関数を呼び出して、これらの 2 つの変数への適切な値のセットアップを試みます。

セットアップ関数には、現在のところ 3 つのタイプがあります。

1. 変数bug-reference-forge-alistおよびbug-reference-setup-from-vc-alistで設定できるバージョンコントロールされたファイルをセットアップする。デフォルトでは使用する issue

トラッカーに`https://debbugs.gnu.org`、issue は通常は`bug#13`のように参照 (ただし他の多くの表記も考慮) する GNU プロジェクト、そして GitLab、Gitea、SourceHut、GitHub のような複数種類の現代的なソフトウェアフォージをセットアップできる (訳注: FOSS 開発コミュニティーにおいて forge とはコンピューターアプリケーションの開発および共有のためのウェブベースのグループウェアを意味する)。このようなフォージのインスタンスをセルフホスティングでデプロイしている場合には、`bug-reference-forge-alist`を通してそれを bug-reference に伝えるのがもっとも簡単な方法である。

2. メールフォルダーや mbox の名前から推察される email、および変数`bug-reference-setup-from-mail-alist`で設定できるメールヘッダー値をセットアップする。built-in、news-、およびメールヘッダー Section 31.1 [Gnus], page 450 と Chapter 30 [Rmail], page 429 がサポートされている。
3. 変数`bug-reference-setup-from-irc-alist`で設定できる IRC チャンネルをセットアップする。ビルトインの IRC クライアントである Rcirc (*The Rcirc Manual*)、および ERC (*The ERC Manual*) がサポートされている。

これらのモードのほとんどは、すべて`bug-reference-mode`を有効にするだけで充分ですが、Rmail だけは若干異なるセットアップが必要になります。

```
;; バージョンコントロールされたファイルなら vc ベースのセットアップを使用
(add-hook 'prog-mode-hook #'bug-reference-prog-mode)

;; Gnus (summary バッファーと article バッファー)
(add-hook 'gnus-mode-hook #'bug-reference-mode)

;; Rmail
(add-hook 'rmail-show-message-hook #'bug-reference-mode-force-auto-setup)

;; Rcirc
(add-hook 'rcirc-mode-hook #'bug-reference-mode)

;; ERC
(add-hook 'erc-mode-hook #'bug-reference-mode)
```

Rmail の場合には、モードフックのかわりに関数`bug-reference-mode-force-auto-setup` (`bug-reference-mode`をアクティブにして強制的に自動セットアップを行う) と組み合わせて`rmail-show-message-hook`を使う必要があります。その理由は Rmail ではすべてのメッセージが同じバッファーにありますが、セットアップは別のメッセージが表示されるたびに実行される必要があるからです。

サードパーティ製パッケージへのサポートの追加

bug-reference の auto-setup にたいするサポートの追加は、通常ならとても単純です。必要な情報 (MUA の場合には List-Id/To/From/Cc のメールヘッダー値) を集める引数なしのセットアップ関数を記述して、以下のヘルパー関数のいずれかを呼び出します:

- `bug-reference-setup-from-vc-alist`に応じたセットアップを行う `bug-reference-maybe-setup-from-vc`
- `bug-reference-setup-from-mail-alist`に応じたセットアップを行う `bug-reference-maybe-setup-from-mail`
- `bug-reference-setup-from-irc-alist`に応じたセットアップを行う `bug-reference-maybe-setup-from-irc`

セットアップ関数は bug-reference モードをセットアップできるなら非`nil`をリターンします。これは上記のヘルパー関数のいずれかを呼び出すのが、関数が最後に行うこととなるケースです。

最後にそのセットアップ関数を`bug-reference-auto-setup-functions`に追加する必要があります。

これらの auto-setup 関数は最初のステップとして、`major-mode`をチェックする等によりそれらが適用可能かチェックする必要があることに注意してください。

デバッグパッケージとの統合

プロジェクトの issue がサーバー`https://debbugs.gnu.org`でトラックされていれば、Package メニュー (Chapter 32 [Packages], page 490 を参照) でダウンロードできる`debbugs`パッケージを使用して、Emacs から直接レポートの閲覧や回答ができます。このパッケージは、以下のように`bug-reference-mode`と`bug-reference-prog-mode`の上にアクティブ化できるマイナーモード`debbugs-browse-mode`を追加します:

```
(add-hook 'bug-reference-mode-hook 'debbugs-browse-mode)
(add-hook 'bug-reference-prog-mode-hook 'debbugs-browse-mode)
```

26 abbrev(略語)

定義された *abbrev(abbreviation:* 略語の意*)* とは、挿入したとき他のテキストに展開される単語のことです。abbrev は、特別な方法で展開されるようにユーザーにより定義されます。たとえば ‘`foo`’ を、‘`find outer otter`’に展開されるように定義したとします。その後、`f o o SPC`とタイプすることにより、バッファーに ‘`find outer otter` ’を挿入できます。

略語機能の 2 番目の種類は、動的 *abbrev* 展開 *(dynamic abbrev expansion)* と呼ばれます。ポイントの前の文字で始まる単語をバッファーから探して、その文字を展開するために、明示的なコマンドで動的 abbrev 展開を使用します。Section 26.7 [Dynamic Abbrevs], page 377 を参照してください。

3 番目の種類の *hippie expansion(*ヒッピー展開*)* は、略語展開を一般化したものです。Section “Hippie Expansion” in *Features for Automatic Typing* を参照してください。

26.1 abbrev の概念

abbrev とは、特定の展開結果に展開されるために定義された単語のことです。abbrev の後ろに単語の区切りとなる文字を挿入したとき、それは abbrev を展開し、abbrev を展開結果に置き換えます。たとえば ‘`foo`’が ‘`find outer otter`’に展開される abbrev として定義された場合、`f o o .`とタイプすると ‘`find outer otter.`’が挿入されます。

abbrev は、バッファーローカルなマイナーモードの Abbrev モードが有効なときだけ展開されます。Abbrev モードを無効にすると、定義した abbrev は忘れられますが、再び Abbrev モードを有効にすると展開されます。コマンド `M-x abbrev-mode`は、Abbrev モードを切り替えます。数引数を指定した場合、引数が正のときは Abbrev モードをオン、他の場合はオフに切り替えます。Section 20.2 [Minor Modes], page 241 を参照してください。

abbrev は、あるメジャーモードのときだけアクティブになる、モード特有 *(mode-specific)* な定義をもつことができます。abbrev は、すべてのメジャーモードでアクティブになる、グローバル *(global)* な定義をもつこともできます。同じ abbrev が、グローバルな定義と、異なるメジャーモードのための、さまざまなモード特有の定義をもつことができます。カレントのメジャーモードにたいするモード特有の定義は、グローバルな定義をオーバーライドします。

Abbrev モードが有効かにかかわらず、編集セッションの間に対話的に abbrev を定義できます。それ以降のセッションでリロードして使用するために、ファイルに abbrev 定義のリストを保存することもできます。

26.2 abbrev の定義

`C-x a g`
: ポイントの前の 1 つ以上の単語を使用して、それが展開結果となる abbrev を定義します (`add-global-abbrev`)。

`C-x a l`
: 同じですが、カレントメジャーモードに特有の abbrev を定義します (`add-mode-abbrev`)。

`C-x a i g`
: バッファーの単語を abbrev として定義します (`inverse-add-global-abbrev`)。

`C-x a i l`
: バッファーの単語を、モード特有の abbrev として定義します (`inverse-add-mode-abbrev`)。

`M-x define-global-abbrev RET` *`abbrev`* `RET` *`exp`* `RET`
: *abbrev*を、*exp*に展開される abbrev として定義します。

`M-x define-mode-abbrev RET` *abbrev* `RET` *exp* `RET`
: *abbrev*を、*exp*に展開されるモード特有の abbrev として定義します。

`M-x kill-all-abbrevs`
: すべての abbrev 定義を削除して、白紙状態にします。

abbrev を定義する通常の方法は、abbrev に展開させたいテキストを入力して、ポイントをその後ろに配し、`C-x a g` (`add-global-abbrev`) とタイプします。これはミニバッファーを使って abbrev 自身を読み取り、ポイントの前の 1 つ以上の単語にたいする abbrev として定義します。数引数を使用してポイントの前のいくつの単語が展開結果となるかを指定します。たとえば、上述した abbrev の‘`foo`’を定義するには、‘`find outer otter`’とテキストをタイプしてから、`C-u 3 C-x a g f o o RET`とタイプします。

`transient-mark-mode`を使っていれば (デフォルトでは有効)、定義する abbrev の展開結果としてアクティブなリージョンを使用します。もし有効でない場合には、`C-x a g`の数引数に 0 を指定すると、リージョン内容を使用することを意味します。

`C-x a l` (`add-mode-abbrev`) は似ていますが、これはカレントのメジャーモードにたいする、モード特有の abbrev を定義します。引数の機能は `C-x a g`と同じです。

`C-x a i g` (`inverse-add-global-abbrev`) と `C-x a i l` (`inverse-add-mode-abbrev`) は、逆のことを行ないます。abbrev となるテキストがすでにバッファーに存在する場合、これらのコマンドはミニバッファーに展開結果を指定することにより、abbrev を定義します。これらのコマンドは、この定義を使って abbrev テキストを展開します。

abbrev または展開結果をバッファーに入力せずに、コマンド `define-global-abbrev`で定義することができます。これは 2 つの引数— abbrev と展開結果を読み取ります。コマンド `define-mode-abbrev`は、モード特有の abbrev にたいして同様のことを行います。

abbrev の定義を変更するには、単に新しい定義を作成するだけです。abbrev がすでに定義をもつ場合、abbrev 定義コマンドはそれを置換する前に確認を求めます。

abbrev 定義を削除するには、`C-u - C-x a g`や `C-u - C-x a l`のように、abbrev 定義コマンドに負の引数を与えます。前者はグローバルな定義を削除し、後者はモード特有の定義を削除します。`M-x kill-all-abbrevs`は、すべての abbrev にたいしてグローバルとローカルの両方の定義を削除します。

26.3 abbrev 展開の制御

Abbrev モードが有効な場合、バッファーのポイントの前に abbrev があり、そこで自己挿入文字として空白文字か区切り文字 (SPCやカンマなど) を挿入したときは、常に abbrev が展開されます。より正確には、単語を構成しない任意の文字は abbrev を展開し、単語を構成する任意の文字は abbrev の一部となります。もっとも一般的な abbrev の使用法は、まず abbrev を挿入し、それから区切り文字か空白文字を挿入して abbrev を展開する方法です。

abbrev の展開は、大文字小文字を維持します。つまり‘`foo`’は‘`find outer otter`’に、‘`Foo`’は‘`Find outer otter`’に展開されます。デフォルトでは‘`FOO`’は‘`Find Outer Otter`’に展開されますが、変数 `abbrev-all-caps`を非 `nil`値に変更した場合は、‘`FIND OUTER OTTER`’に展開されます。

以下は abbrev の展開を制御するコマンドです:

`M-'`
: プレフィクスと、その後の展開される abbrev を分割します (`abbrev-prefix-mark`)。

`C-x a e`
: ポイントの前の abbrev を展開します。(`expand-abbrev`)。これは Aッb レ v モードが有効でなくても効果があります。

`M-x unexpand-abbrev`
: 最後に展開した abbrev をアンドゥします。

`M-x expand-region-abbrevs`
: リージョンで見つかったいくつか、またはすべての abbrev を展開します。

abbrev を展開して、その展開結果にプレフィクスをつけたい場合があるかもしれません。たとえば、‘`cnst`’が‘`construction`’に展開されるとき、これを使って‘`reconstruction`’を入力したいと思うかもしれません。しかし、`recnst`とタイプしてもうまくいきません。なぜなら、それが abbrev として定義される必要があるからです。これは、プレフィクス‘`re`’と abbrev の‘`cnst`’の間で、コマンド `M-'` (`abbrev-prefix-mark`) を使うことにより行なうことができます。最初にまず‘`re`’を挿入します。そこで `M-'`をタイプします。これにより、コマンドが機能していることを示すためバッファーにハイフンが挿入されます。その後、abbrev の‘`cnst`’を入力します。このときバッファーには‘`re-cnst`’が含まれます。そこで単語を構成しない文字を挿入すると、abbrev の‘`cnst`’が‘`construction`’に展開されます。この展開ステップでは、`M-'`が使用中であることを示していたハイフンも削除されます。結果は期待した通り‘`reconstruction`’となります。

abbrev を展開せずに abbrev のテキストをバッファーに残したい場合、abbrev の後ろの区切り文字を `C-q`で挿入して、これを行なうことができます。したがって `foo C-q ,`とタイプすると、それは展開されず、バッファーには‘`foo,`’が残ります。

間違って abbrev を展開した場合、`C-/` (`undo`) で展開をアンドゥできます。これは abbrev 展開による挿入をアンドゥし、それを abbrev テキストに戻します。期待する結果が展開されない abbrev と終端となる非単語文字の場合、`C-q`でクォートして終端文字を再挿入しなければなりません。`M-x unexpand-abbrev`を使えば終端文字を削除せずに、最後の展開を取り消すことができます。

`M-x expand-region-abbrevs`は、リージョン内の定義された abbrev を検索し、見つかった abbrev それぞれにたいして、abbrev を展開結果に置き換えるか尋ねます。このコマンドは abbrev を使ってテキストを挿入したが、最初に Abbrev モードをオンにするのを忘れたときに便利です。これは特別な abbrev 定義のセットで、複数のグローバルな置き換えを一度に行なうときにも便利です。このコマンドは、Abbrev モードが有効でなくても効果があります。

関数 `expand-abbrev`は、`abbrev-expand-function`が指定する関数を呼び出すことにより展開を行ないます。この関数を変更することにより、abbrev の展開を自由に変更できます。Section “Abbrev Expansion” in *The Emacs Lisp Reference Manual* を参照してください。

26.4 abbrev の提案

カレントでアクティブな定義済み abbrev が存在するテキストを手動でタイプした際に、abbrev の提案を取得できます。たとえば‘`find outer otter`’に展開される‘`foo`’という abbrev があるときに手動で‘`find outer otter`’とタイプすると、Emacs がこれを検知してタイプを中断した際にエコーエリアにヒントを表示します。

abbrev 提案の機能を有効にするには、オプション `abbrev-suggest`を非 `nil`値にカスタマイズしてください。

変数 `abbrev-suggest-hint-threshold`はユーザーにたいして abbrev を提案するタイミングを制御します。この変数は Emacs が abbrev 使用の提案を行うのに要する最小の節約値を定義します (ここで言う節約値とはユーザーによる試行を要さない文字数を意味する)。たとえばユーザーが‘`foo bar`’ (7 文字) をタイプして、‘`fubar`’ (5 文字) という定義済みの abbrev が存在する場合には、このしきい値の数値が 2 より大きければ提案を受け取ることはありません。この例の場合には、デフォルト値の 3 では abbrev の使用による節約値がしきい値を下廻るので、ユーザーが提案を受け取ることはないでしょう。常に abbrev 提案を受け取りたければ、この変数に 0 をセットしてください。

コマンド `abbrev-suggest-show-report`はカレント編集セッション中のすべての abbrev 提案をバッファーに表示します。これは複数の abbrev 提案を受け取り、それらをすべて憶えていない場合に便利かもしれません。

26.5 abbrev のテストと編集

`M-x list-abbrevs`
: すべての abbrev 定義のリストを表示します。数引数を指定した場合は、ローカルな abbrev のリストだけを表示します。

`M-x edit-abbrevs`
: abbrev のリストを編集します。定義の追加、変更、削除ができます。

`M-x list-abbrevs`の出力は以下のようなものです:

```
   various other tables...
   (lisp-mode-abbrev-table)
   "ks"          0    "keymap-set"
   (global-abbrev-table)
   "dfn"         0    "definition"
```

(空行に意味はありません。また他の abbrev テーブルは省略しています。)

カッコで括られた名前を含む行は、特定の abbrev テーブルの abbrev にたいするヘッダーです。`global-abbrev-table`はすべてのグローバルな abbrev を含み、その他のメジャーモードの後ろについた abbrev テーブルは、モード特有の abbrev を含みます。

それぞれの abbrev テーブルで、空行でない行は 1 つの abbrev の定義です。行の先頭の単語は abbrev です。その後ろの数字は、その abbrev が展開された回数です。Emacs はこれを追跡することにより、実際に使用されている abbrev を調べて、ときどきしか使わないものを削除するのを助けます。行の最後の文字列は abbrev の展開結果です。

‘`(sys)`’とマークされている abbrev もいくつかあります。これらは *system abbrevs(*システムの略語*)* で、さまざまなモードにたいして事前に定義されており、ユーザーの abbrev ファイルには保存されません (Section “Abbrevs” in *The Emacs Lisp Reference Manual* を参照してください)。system の abbrev を無効にするには、同じ名前で展開結果が abbrev 自身と同じになる abbrev を定義し、それを abbrev ファイルに保存してください。

`M-x edit-abbrevs`を使うと、Emacs バッファーで abbrev のリストを編集することにより、abbrev 定義の追加、変更、kill ができます。リストの書式は、上記で説明した書式を同じです。abbrev のバッファーは`*Abbrevs*`と呼ばれ、モードは Edit-Abbrevs モードです。このバッファーで `C-c C-c`をタイプすると、そのバッファーで指定された abbrev 定義がインストールされ、リストに定義されていない abbrev は削除されます。

コマンド `edit-abbrevs`は、実際には `list-abbrevs`と同じですが、`list-abbrevs`が単に`*Abbrevs*`を他のウィンドウに表示するのに比べ、このコマンドはそのバッファーを選択する点が異なります。

26.6 abbrev の保存

以下のコマンドにより、編集セッション間で abbrev 定義を維持できます。

`M-x write-abbrev-file RET` *file* `RET`
: すべての abbrev 定義の記述を、ファイル *file*に書き込みます。

`M-x read-abbrev-file RET` *file* `RET`
: ファイル *file*を読み込み、そこで指定されている abbrev を定義します。

`M-x define-abbrevs`
: カレントバッファーの定義から abbrev を定義します。

`M-x insert-abbrevs`
: すべての abbrev とそれらの展開結果を、カレントバッファーに挿入します。

`M-x write-abbrev-file`は、ミニバッファーを使用してファイル名を読み取り、すべてのカレントの abbrev 定義の記述を、そのファイルに書き込みます。これは後のセッションで使用するために、abbrev 定義を保存するのに使われます。ファイルに保存されるテキストは一連の Lisp 式で、それが実行されると保存したときと同じ abbrev を定義します。

`M-x read-abbrev-file`は、ミニバッファーを使用してファイル名を読み取り、ファイル内容に対応する abbrev を定義します。関数`quietly-read-abbrev-file`も同様ですが、これはエコーエリアにメッセージを表示しません。これを対話的に呼び出すことはできず、主に init ファイル (Section 33.4 [Init File], page 528 を参照してください) で使用されます。どちらの関数も引数に`nil`が指定されると、変数`abbrev-file-name`で与えられるファイルを使用します。この変数のデフォルトは`~/.emacs.d/abbrev_defs`です。これは標準の abbrev 定義ファイルで、Emacs は起動時にこのファイルから自動的に abbrev をロードします (例外として Emacs がバッチモードで開始されたときは abbrev ファイルをロードしません。バッチモードについての説明は、Section C.2 [Initial Options], page 576 を参照してください)。

abbrev のどれかを変更した場合、Emacs は (`C-x s`や`C-x C-c`などで)、すべてのファイルの保存するか尋ねるときに abbrev についても尋ねます。これは`abbrev-file-name`で指定したファイルに、それらを保存します。この機能は変数`save-abbrevs`を`nil`にセットすることにより抑止できます。`silently`にセットすることにより、確認なしで abbrev が保存されるようになります。

コマンド`M-x insert-abbrevs`および`M-x define-abbrevs`は、前のコマンドと似ていますが、Emacs バッファーのテキストにたいして機能します。`M-x insert-abbrevs`は、カレントの abbrev 定義の記述をカレントバッファーのポイントの後ろにテキストとして挿入します。`M-x define-abbrevs`は、カレントバッファー全体を解析して、対応する abbrev を定義します。

26.7 動的 abbrev 展開

上記で説明した abbrev 機能は、テキストの挿入にしたがい自動的に処理されますが、すべての abbrev を明示的に定義しなければなりません。対照的に、動的 *abbrev(dynamic abbrevs)* は、バッファーの内容から略語の展開結果を自動的に決定することができます、しかし動的 abbrev の展開は、明示的に要求したときだけ行なわれます。

`M-/`
: バッファーのポイントの前の単語を動的 *abbrev* として、その略語で始まる単語を検索することにより展開します (`dabbrev-expand`)。

`C-M-/`
: ポイントの前の単語を動的 abbrev として補完します (`dabbrev-completion`)。

たとえばバッファーが '`does this follow `'が含んでいて、`f o M-/`とタイプすると、これは '`follow`'を挿入します。なぜならそれが '`fo`'で始まる、そのバッファーの最後の単語だからです。`M-/`に数引数を指定すると、それはポイントから後方に検索して 2 番目、3 番目、... の異なる展開結果を検索します。`M-/`を繰り返すと他の展開結果を後方に検索します。ポイントの前のテキストをすべて検索した後は、ポイントの後のテキストを検索します。変数`dabbrev-limit`が非`nil`の場合、それはバッファーの中で展開結果を探す範囲を指定します。

カレントバッファーを検索した後、通常 M-/は他のバッファーを検索します。dabbrev-check-all-buffersと dabbrev-check-other-buffersは、他のバッファー (もし存在するなら) のどれが検索されたか決定するのに使用できます。dabbrev-ignored-buffer-modesのいずれかのモードから派生したメジャーモードをもつバッファーは無視されます。

どのバッファーを検索するかを制御するには、変数 dabbrev-ignored-buffer-namesおよび dabbrev-ignored-buffer-regexpsをカスタマイズします。前者の値は、スキップするバッファー名のリストです。後者の値は正規表現のリストで、バッファー名がこれらの正規表現のどれかにマッチした場合、動的 abbrev 展開はそのバッファーをスキップします。

C-u - M-/のように M-/に負の引数を指定すると、これは最初にポイントの後ろの展開結果を検索し、その後は他のバッファーを検索し、ポイントの前の展開結果は最後に報告します。他の展開結果を探すために M-/を繰り返す場合は、引数に何もしていしないでください。M-/を繰り返すことにより、ポイントの後、その後はポイントの前の展開結果を巡回します。

動的 abbrev を展開した後、その展開結果の元のコンテキストで展開結果の後ろに続く、追加の単語をコピーすることができます。コピーしたい追加の単語ごとに、単に SPC M-/とタイプします。単語間のスペースおよび区切り文字は、単語とともにコピーされます。

M-/が展開する単語を決定する方法と、それを展開する方法を制御できます。Section 26.8 [Dabbrev Customization], page 378 を参照してください。

コマンド C-M-/ (dabbrev-completion) は、動的 abbrev の補完を行ないます。利用可能な展開結果を 1 つずつ試すかわりに、これはすべてを検索して、それらがもつ共通のテキストを挿入します。共通部分がない場合、C-M-/は、通常の方法で選択することができる補完リストを表示します。Section 5.4 [Completion], page 31 を参照してください。

動的 abbrev 展開は、Abbrev モードとは完全に独立しています。M-/での単語の展開は、その単語が通常の abbrev として定義されていることとは無関係です。

26.8 動的 abbrev のカスタマイズ

動的 abbrev 展開は通常、展開結果の検索で大文字小文字を無視します。したがって、展開結果と展開する単語は、大文字小文字が一致する必要はありません。

この機能は変数 dabbrev-case-fold-searchにより制御されます。この値が tの場合、検索で大文字小文字は無視されます。nilの場合、単語と展開結果は大文字小文字が一致していなければなりません。値が case-fold-search(デフォルト) の場合、変数 case-fold-searchが展開結果の検索で、大文字小文字を無視するかどうかを制御します (Section 12.9 [Lax Search], page 120 を参照してください)。

動的 abbrev 展開は通常、大文字小文字のパターンにしたがって展開結果を変換することにより、展開する動的 *abbrev* の大文字小文字のパターンを維持します。

変数 dabbrev-case-replaceは、動的 abbrev の大文字小文字のパターンを維持するかどうかを制御します。この値が tの場合、動的 abbrev の大文字小文字のパターンはほとんどのケースで維持されます。nilの場合、展開結果は常にそのままコピーされます。値が case-replace(デフォルト) の場合、変数 case-replaceが展開結果をそのままコピーするかどうかを制御します (Section 12.10.3 [Replacement and Lax Matches], page 123 を参照してください)。

しかし、展開結果が複雑にミックスされた大文字小文字のパターンを含む場合、そして動的 abbrev がそのパターンにある程度マッチする場合、これらの変数の値にかかわらず、展開結果は常にそのままコピーされます。したがって、たとえばバッファーが variableWithSillyCasePatternというテキストを含む場合、v a M-/とタイプすると、大文字小文字のパターンを含めて展開結果をコピーします。

変数 dabbrev-abbrev-char-regexpが非 nilの場合、これは動的展開の目的のために、どの文字を単語の一部とするかを制御します。正規表現は 2 文字以上ではなく、ただ 1 文字だけにマッチしなければなりません。同じ正規表現が、どの文字が展開結果の一部となるかも決定します。値が nil(デフォルト) の場合は特別な意味をもちます。動的 abbrev(たとえばポイント位置の単語) は単語構成文字から構成されますが、それらの展開結果は単語とシンボル文字のシーケンスから取得されます。これはプログラムソース内、および多くの言語による人間が読むことができるテキストにたいするシンボルの展開にたいして、一般的にこれは適切ですが、一般的ではない句読点を含むテキストバッファーにおいては、おそらくあなたが望むものではないかもしれません。そのような場合は、値\"\\\\sw\"がよい結果を生成するかもしれません。

シェルスクリプトおよび makefile では、変数名にプレフィクス '$'があるときと、ないときがあります。このタイプのテキストのためのメジャーモードは、オプションのプレフィクスを扱うために、変数 dabbrev-abbrev-skip-leading-regexpをセットして、動的 abbrev 展開をカスタマイズできます。この値には、動的 abbrev 展開が無視すべきオプションのプレフィクスにマッチする正規表現を指定します。デフォルトは nilで、これは文字をスキップしないことを意味します

27 Dired (ディレクトリーエディター)

Diredはディレクトリー、およびオプションでそれのサブディレクトリーのリストを含む、Emacsバッファーを作成します。このバッファー内を移動するために、通常のEmacsコマンドと、リストされたファイルを操作するための、特別なコマンドを使うことができます。Diredはローカルとリモートの両方のディレクトリーで機能します。

通常はDiredバッファーは読み取り専用で、テキストの挿入はできません (とはいえWdiredモードではそれが可能。Section 27.17 [Wdired], page 397 を参照されたい)。`d`や`x`のような通常のプリント文字は、特別なDiredコマンドに再定義されています。Diredコマンドには、カレントファイル (カレント行のファイルのこと) をマークしたり、フラグをつけるものがいくつかあり、他のコマンドは、マークされたファイルやフラグがつけられたファイルにたいして処理を行ないます。最初に特定のファイルをマークして、それらすべてにたいして1つのコマンドで操作を行なうのです。

Dired-Xパッケージは、Diredモードのためのさまざまな特別の機能を提供します。*Dired Extra User's Manual* を参照してください。

ディレクトリーのファイルのリストは、`C-x C-d` (`list-directory`) でも閲覧することができます。Diredとは異なり、このコマンドではリストされたファイルにたいする操作はできません。Section 15.8 [Directories], page 162 を参照してください。

27.1 Diredの起動

Diredを呼び出すには、`C-x d` (`dired`) とタイプします。これはミニバッファーを使用してディレクトリー名を読み取り、そのディレクトリーのファイルを一覧するする *Dired* バッファーを開きます。ミニバッファーの引数に、ワイルドカードによるファイル名パターンを与えることもできます。この場合、Diredバッファーには、そのパターンにマッチする、すべてのファイルが一覧されます。ディレクトリー部分にワイルドカードが出現することもあります。たとえば、

```
C-x d  ~/foo/*.el  RET
C-x d  ~/foo/*/*.el  RET
```

1つ目の例は、ディレクトリー'`foo`'内の、拡張子'`.el`'のすべてのファイルをリストします。2つ目の例は、'`foo`'のすべてのサブディレクトリー内の、拡張子'`.el`'のファイルをリストします。

Posixシステムでシステムシェルが *globstar* (再帰的なグロブ機能) をサポートしていて、それが有効な際にはDiredで再帰的なグロビングを使用できます:

```
C-x d  ~/foo/**/*.el  RET
```

このコマンドは'`foo`'のすべてのサブディレクトリーを再帰的に下降して、拡張子'`.el`'をもつすべてのファイルのディレクトリーリストを生成します。シェル間でのglobstarの実装は微妙に異なることに注意してください。期待する動作が知るにはシェルのマニュアルを確認してください。

globstarをサポートしてもデフォルトではサポートが無効なシェルの場合には、`dired-maybe-use-globstar`を非`nil`値にカスタマイズすることでDiredにこの機能を使用させることができます。これにより、それらのシェルでglobstarを有効にする方法をDiredに知らせて有効にすることができます (これらのシェルのリストについては`dired-enable-globstar-in-shell`を参照)。

ミニバッファー内では、通常のヒストリー、および補完のコマンドが使用できます。特に、`M-n`は(もしあれば)visitされているファイルの名前を配します (Section 5.5 [Minibuffer History], page 36 を参照)。

`C-x C-f` (`find-file`) にディレクトリー名を与えて、Diredを呼び出すこともできます。

`C-x C-j` (`dired-jump`) とタイプすることによって、任意のバッファーのデフォルトディレクトリー (Section 15.1 [File Names], page 145 を参照) でDiredを呼び出すようEmacsに指示できま

す。バッファーがファイルを visit している場合には、Dired バッファーでそのファイルの行にポイントを移動します。それ以外の場合には、リストされたディレクトリーの最初のファイルにポイントが移動します。例外として Dired バッファーで C-x C-jとタイプすると、Emacs は親ディレクトリーのディレクトリーリストを表示して、dired-jumpを呼び出したディレクトリーに対応する行にポイントを配置します。C-x 4 C-j (dired-jump-other-window) も同様の効果をもちますが、新たなウィンドウに Dired バッファーを表示します。

変数 dired-listing-switchesは、ディレクトリーをリストするために、lsに与えるオプションを指定します。この文字列には、'-l'が含まれていなければなりません。diredコマンドにプレフィクス引数を使用した場合、ディレクトリーの指定の前に、lsのためのスイッチを指定できます。スイッチが指定される方法は問いません。lsのスイッチには、引数を要求しない短いオプション (1 文字) と、長いオプション ('--'で始まり、引数は '='で指定される) を含めることができます。

Dired は改行文字が埋め込まれた名前をもつファイルを上手く処理できません。そのようなファイルが多ければ、dired-listing-switchesに '-b'の追加を検討することができます。これはすべてのスペシャル文字をクォートすることで、Dired はそれらをより良好に処理することができます (一時的に '-b'を追加するために C-u C-x dコマンドの使用も可能)。

Dired が ls呼び出しに使用したスイッチの指標をモードラインに表示します。デフォルトでは、そのスイッチによる並べ替えが名前順か日付順を示すものかどうかの判定を Dired が試み、それをモードラインに示します。dired-switches-in-mode-line変数が as-isなら、スイッチをそのまま表示します。この変数の値が整数なら、表示するスイッチをその長さに切り詰めます。この変数の値は関数でもよく、その場合には関数は唯一のパラメーターとして dired-actual-switchesとともに呼び出されて、モードラインに表示する文字列をリターンする必要があります。

システムの lsコマンドが '--dired'オプションをサポートする場合、Dired は自動的にそのオプションを渡します。これにより、Dired が名前を解析できないという特殊なファイル名にたいして、lsが特別なエスケープシーケンスを付します。Emacs セッションで最初に Dired を実行するとき、lsコマンドに '--dired'オプションを指定して呼び出すことにより、そのスイッチをサポートするかどうかチェックします。exit コードが 0 の場合、それ以降 Dired は '--dired'オプションを使用し、それ以外の場合は使用しません。変数 dired-use-ls-diredをカスタマイズすることにより、このチェックを抑止することができます。値 unspecified(デフォルト) は、チェックを行なうことを意味します。その他の非 nil値は、'--dired'オプションを使用することを意味します。nilは、'--dired'オプションを使用しないことを意味します。

MS-Windows や MS-DOS システム、およびいくつかのリモートシステムでは、Emacs が lsをエミュレートします。このエミュレーションのオプションと特性については、Section H.4 [ls in Lisp], page 611 を参照してください。

Dired バッファーを他のウィンドウに表示するには、C-x 4 d (dired-other-window) を使用します。C-x 5 d (dired-other-frame) は、Dired バッファーを別のフレームに表示します。

q (quit-window) とタイプすると、Dired バッファーは隠され (bury) ます。ウィンドウがそのバッファーのためだけに作成された場合は、そのウィンドウを削除します。

27.2 Dired バッファーでの移動

Emacs の通常のカーソル移動コマンドは、Dired バッファーでも利用可能です。キー C-nと C-pはそれぞれ dired-next-lineと dired-previous-lineを実行して、行の先頭ではなくファイル名の先頭にカーソルを配するように、再定義されています。

特に利便性のために、Dired での SPCおよび nは、C-nと等価になっています。pは C-pと等価です (行の移動は Dired では一般的な操作なので、タイプしやすいようにするべきです)。DEL(上に移動し

てフラグを外す) は、単なる上への移動でも便利なときがあります (Section 27.3 [Dired Deletion], page 382 を参照してください)。

j (dired-goto-file) は、ミニバッファーを使用してファイル名の入力を求め、Dired バッファーで、そのファイルが記述されている行にポイントを移動します。

M-s f C-s (dired-isearch-filenames) は、Dired バッファーで前方へのインクリメンタル検索を行ないます。これはファイル名にたいするマッチだけを検索し、それ以外のバッファーのテキストは無視します。M-s f M-C-s (dired-isearch-filenames-regexp) も同じことを行ないますが、これは正規表現による検索です。変数 dired-isearch-filenamesを tに変更した場合、通常の検索コマンドも検索対象がファイル名だけに制限されるようになります。たとえば C-sは、M-s f C-sと同様に振る舞います。変数の値が dwimの場合、ポイントの最初の位置がファイル名にあるときだけ、検索コマンドはファイル名にマッチします。インクリメンタル検索についての情報は、Chapter 12 [Search], page 105 を参照してください。

複数ディレクトリーにたいするものも含めて、Dired バッファーでは追加の移動コマンドが利用可能です。Section 27.13 [Subdirectory Motion], page 395 を参照してください。

27.3 Dired でのファイルの削除について

もっともよく使われる Dired の使用法は、最初にファイルを削除するためにフラグ *(flag)* をつけて、その後にフラグがつけられたファイルを削除をする方法です。

d 削除するために、このファイルにフラグをつけます (dired-flag-file-deletion)。

u 削除フラグを外します (dired-unmark)。

DEL その行の削除フラグを外して、ポイントを前の行に移動します (dired-unmark-backward)。

x 削除のフラグがつけられたファイルを削除します (dired-do-flagged-delete)。

ファイルが記述されている行に移動して、d (dired-flag-file-deletion) とタイプすることにより、ファイルを削除するためのフラグをつけることができます。削除フラグは行頭に 'D'が表示されます。このコマンドはポイントを次の行に移動するので、dコマンドを繰り返すことにより、連続してファイルにフラグをつけることができます。数引数は繰り返し回数を指定します。負の引数は前のファイルにフラグをつけることを意味します。

リージョンがアクティブの場合、dコマンドはリージョンのすべてのファイルに削除のフラグをつけます。この場合、コマンドはポイントを移動せず、プレフィクス引数も無視します。

ファイルを即座に削除せず、削除のフラグをファイルにつける理由は、意図しないファイルを削除する危険を軽減するためです。フラグがついたファイルを Dired に削除をさせる前に、コマンド uまたは DELで、削除フラグを外すことができます。u (dired-unmark) は dと同じように機能しますが、これはフラグをつけるのではなくフラグを外します。DEL (dired-unmark-backward) は、上に移動してフラグを外します。これは uに引数 −1 を指定するのと同じです。どちらのコマンドも、数引数は繰り返し回数で、負の引数は反対方向にフラグを外していくことを意味します。リージョンがアクティブの場合、ポイントを移動せずにリージョン内のすべてのファイルのフラグを外します。

フラグがついたファイルを削除するには、x (dired-do-flagged-delete) とタイプします。このコマンドは削除フラグがついた、すべてのファイルのリストを表示して、確認を求めます。もしこれに yesと応えた場合、Dired はフラグがついたファイルを削除して、それらのファイルにたいする Dired バッファーの行を削除します。Dired バッファーの行数は少し減り、バッファーは選択されたまま残ります。

確認を求められたとき `no`と応えるか、`C-g`で中断した場合は、即座に Dired に戻ります。バッファーの削除フラグはそのまま残り、実際に削除されたファイルはありません。

空ディレクトリーは他のファイルと同様に削除できますが、Dired は通常、空でないディレクトリーは削除できません。しかし変数 `dired-recursive-deletes`が非 `nil`の場合には、Dired はディレクトリーの内容も含めて、空でないディレクトリーを削除できます。`dired-recursive-deletes`を `nil`にセットしている場合でも、ディレクトリーの内容すべてについて問い合わせされることなく、ディレクトリーを再帰的に削除したいこともあるかもしれません。これには若干の危険が伴います。変数の値が `always`の場合は、さらに危険性は増すとはいえ、Dired は空でないディレクトリーを再帰的に削除するでしょう。

`dired-recursive-deletes`を `nil`にセットしている場合でも、ディレクトリーの内容すべてについて問い合わせされることなく、ディレクトリーを再帰的に削除したいこともあるかもしれません。たとえば、多くのディレクトリーにたいして、それらすべてを安全に削除できると確信がもてるときは、それらに削除のマークを付与したいと思うかもしれません。空でないディレクトリーすべてにたいして、確認が求められますが、`all`と応えた場合は、それ以上の問い合わせなしで残りすべてのディレクトリーが削除されます。

変数 `delete-by-moving-to-trash`を `t`に変更した場合、上記の削除コマンドは対象となるファイルおよびディレクトリーを削除するかわりに、システムの Trash(ゴミ箱) に移動します。Section 15.12 [Misc File Ops], page 167 を参照してください。

ファイルを削除する代替手段は、それらを `m`でマークして `D`で削除する方法です。Section 27.7 [Operating on Files], page 387 を参照してください。

27.4 大量のファイルに一度にフラグをつける

コマン`#`、`~`、`.`、`% &`、`% d`は、ファイル名にもとづき多くのファイルにフラグをつけます。

`#` すべての自動保存ファイル (名前の最初と最後が '`#`'のファイル) にたいして、削除のフラグをつけます (Section 15.6 [Auto Save], page 159 を参照してください)。

`~` すべてのバックアップファイル (名前の最後が '`~`'のファイル) にたいして、削除のフラグをつけます (Section 15.3.2 [Backup], page 152 を参照してください)。

`.` (ピリオド)
余分な番号付きバックアップファイルに、削除のフラグをつけます。一番古いものと、一番新しいバックアップのいくつかは除外され、その中間のファイルにフラグがつけられます。

`% &` 簡単にファイルを再作成できることを示唆する、特定の種類の名前をもつすべてのファイルに削除のフラグをつけます。

`% d` *regexp* `RET`
正規表現 *regexp*にマッチする名前の、すべてのファイルに削除のフラグをつけます。

`#` (`dired-flag-auto-save-files`) は、名前が自動保存ファイルに見える — つまり名前の最初と最後が '`#`'の、すべてのファイルにフラグをつけます。Section 15.6 [Auto Save], page 159 を参照してください。

`~` (`dired-flag-backup-files`) は、名前がバックアップファイルであることを告げる — つまり名前の最後が '`~`'の、すべてのファイルにフラグをつけます。Section 15.3.2 [Backup], page 152 を参照してください。

`.` (ピリオド、`dired-clean-directory`) は、いくつかのバックアップファイルだけに削除のフラグをつけます。つまり、最古と最新のいくつかのファイルを除く、すべてのファイルです。残すべき

最新バージョンの数は通常、変数 dired-kept-versions(kept-new-versionsではありません。これは保存時だけ適用されます) で与えられます。残すべき最古バージョンの数は、変数 kept-old-versionsにより与えられます。

C-u 3 .のように、ピリオドに正の数引数を指定すると、dired-kept-versionsをオーバーライドして、残すべき最新バージョンの数を指定します。負の引数は kept-old-versionsをオーバーライドします。引数の値にマイナスをつけると、残すべき最古バージョンの数を指定します。

% & (dired-flag-garbage-files) は、変数 dired-garbage-files-regexpで指定される正規表現に、名前がマッチするファイルにフラグをつけます。デフォルトでは、TEX により生成される‘.bak’ファイルと、patchにより生成される‘.orig’ファイルと‘.rej’ファイルにマッチします。

% dは、指定された正規表現に名前がマッチするすべてのファイルにフラグをつけます (dired-flag-files-regexp)。マッチングにはファイル名の非ディレクトリー部分だけが使用されます。マッチのアンカーとして‘^’と‘$’を使用できます。% dを使用するとき、特定のサブディレクトリーを隠して除外できます。Section 27.14 [Hiding Subdirectories], page 395 を参照してください。

27.5 Dired のファイルを visit する

Dired バッファーにリストされたファイルを visit したり、調べるための Dired コマンドが、いくつかあります。これらのコマンドは、すべてカレント行のファイルに適用されます。そのファイルが実際にはディレクトリーの場合、これらのコマンドは (別の Dired バッファーを作成して)、そのサブディレクトリーにたいして Dired を呼び出します。

f　C-x C-fとタイプしてファイル名を指定したときと同じように、カレント行に記述されたファイルを visit します (dired-find-file)。Section 15.2 [Visiting], page 146 を参照してください。

RET
e　fと等価です。

o　fと同じですが、そのファイルのバッファーに別のウィンドウを使用します (dired-find-file-other-window)。Dired バッファーは、最初のウィンドウに表示されたまま残ります。これはC-x 4 C-fを使用して、ファイルを visit するのと似ています。Chapter 17 [Windows], page 186 を参照してください。

C-o　カレント行に記述されたファイルを visit し、そのバッファーを別のウィンドウで表示しますが、そのウィンドウを選択しません (dired-display-file)。

mouse-1
mouse-2　クリックした名前のファイルを visit します (dired-mouse-find-file-other-window)。これは oコマンドのように、ファイルの表示に別のウィンドウを使います。

v　カレント行に記述されたファイルを、View モードで表示します (dired-view-file)。View モードは、バッファーを移動するための便利なコマンドを提供しますが、バッファーの変更はできません。Section 11.6 [View Mode], page 82 を参照してください。

^　カレントディレクトリーの親ディレクトリーを visit します (dired-up-directory)。これは、..の行に移動して fをタイプするのと等価です。

dired-kill-when-opening-new-dired-buffer [User Option]
Dired で新たにサブディレクトリーを visit する際、(デフォルトでは)Emacs は古い Dired バッファーをそのままにして、その新しいディレクトリーを表示するために新たにバッファー

をオープンします。このユーザーオプションが非 nilなら、新しいディレクトリーを選択後に古い Dired バッファーを kill します。これは Dired でディレクトリー構造を移動する際に、複数の Dired バッファーが開かれないことを意味します。

27.6 Dired でのマークとフラグ

ファイルに 'D'というフラグをつけるのではなく、他の文字 (通常は '*') でファイルをマークすることもできます。ほとんどの Dired コマンドは、'*'でマークされたファイルにたいしてコマンドを処理します。フラグ付けされたファイルだけに作用するコマンドは、それらのファイルを削除する xだけです。

以下は '*'でマークしたり、マークを外したり、マークにたいして操作を行なうコマンドです (ファイルにたいしてフラグをつけたり外すコマンドについては、Section 27.3 [Dired Deletion], page 382 を参照してください)。

m
* m　カレントファイルを '*'でマークします (dired-mark)。リージョンがアクティブの場合、かわりにリージョン内のすべてのファイルをマークします。そうでない場合、数引数 nが与えられたときは、カレント行から数えて、次の n個のファイルまでをマークします (nが負の場合、前の −n個のファイルをマークします)。サブディレクトリーのヘッダー行 (subdirectory header line: Section 27.12 [Subdirectories in Dired], page 394 を参照) で呼び出さた場合、このコマンドはそのサブディレクトリー内のすべてのファイルをマークします。

* N　マークしたファイルの数とサイズを報告します (dired-number-of-marked-files)。

* *　すべての実行ファイルを '*'でマークします (dired-mark-executables)。数引数を指定すると、それらのファイルのマークを外します。

* @　すべてのシンボリックリンクを、'*'でマークします (dired-mark-symlinks)。数引数を指定すると、それらのファイルのマークを外します。

* /　.と..を除く、すべてのディレクトリーを '*'でマークします (dired-mark-directories)。数引数を指定すると、それらのディレクトリーのマークを外します。

* s　.と..を除く、カレントのサブディレクトリー内のすべてのファイルをマークします (dired-mark-subdir-files)。

u
* u　この行の任意のマークを外します (dired-unmark)。リージョンがアクティブの場合は、かわりにリージョン内のすべてのファイルのマークを外します。そうでない場合、数引数 nが与えられた場合は、カレントファイルから数えて、次の n個のファイルのマークを外します (nが負の場合、前の −n個のファイルのマークを外します)。

DEL
* DEL　ポイントを前の行に移動して、その行のマークを外します (dired-unmark-backward)。リージョンがアクティブの場合、かわりにリージョン内のすべてのファイルのマークを外します。そうでない場合、数引数 nが与えられたときは、カレントファイルから数えて、前の n個のファイルのマークを外します (nが負の場合、次の −n個のファイルのマークを外します)。

`* !`
`U` Dired バッファーの、すべてのファイルのマークを外します (`dired-unmark-all-marks`)。

`* ?` *`markchar`*
`M-DEL` 文字 *markchar*を使用する、すべてのマークを外します (`dired-unmark-all-files`)。`M-DEL`で呼び出された場合、このコマンドは *markchar*の入力を求めます。*markchar*は 1 文字です— これを終了させるために `RET`を使用しないでください。以下の`* c`の説明を参照してください。これは 1 つのマーク文字を、他の文字に置換します。

数引数を指定した場合、このコマンドはマークされたファイルごとに、マークを外すか確認します。`y`は yes(はい) で、`n`は no(いいえ)、`!`は残りのファイルにたいして確認を行なわずにマークを外します。

`* C-n`
`M-}` 次にマークされたファイルへ下に移動します (`dired-next-marked-file`)。“マークされた” ファイルとは、任意の種類のマークがついているファイルです。

`* C-p`
`M-{` 前のマークされたファイルへ、上に移動します (`dired-prev-marked-file`)。

`t`
`* t` すべてのマークを切り替えます (`dired-toggle-marks`)。つまり、‘`*`’でマークされたファイルのマークを外し、マークされていないファイルを ‘`*`’でマークします。他の方法でマークされたファイルは影響を受けません。

`* c` *`old-markchar new-markchar`*
文字 *old-markchar*を使ったすべてのマークを、文字 *new-markchar*を使ったマークに置き換えます (`dired-change-marks`)。このコマンドは主に ‘`*`’や ‘`D`’以外の文字を使用したマークを作成するのに使われます。引数は 1 文字です— 終了させるために `RET`を使用しないでください。

このコマンドは、マーク文字としてほとんどの任意の文字を使用できるので、ファイルをさまざまなクラスに分類することができます。*old-markchar*がスペース (‘ ’) の場合、このコマンドは、すべてのマークされていないファイルにたいして処理を行ないます。*new-markchar*がスペースの場合、このコマンドは対象となるファイルのマークを外します。

このコマンドの威力を知る例として、以下ではマークされていないファイルすべてにフラグ ‘`D`’をセットして、すでに ‘`D`’フラグがついているものはフラグを外しています:

```
* c D t  * c SPC D  * c t SPC
```

この例では、すでに ‘`t`’でマークされたファイルがないと仮定しています。

`% m` *`regexp`* `RET`
`* %` *`regexp`* `RET`
名前が正規表現 *regexp*にマッチするすべてのファイルを、(‘`*`’で) マークします (`dired-mark-files-regexp`)。このコマンドは`% d`と似ていますが、‘`D`’のフラグをつけるかわりに、‘`*`’でマークします。

マッチングには、ファイル名のディレクトリー部分以外だけが使用されます。マッチのアンカーとして、‘`^`’と ‘`$`’が使用できます。サブディレクトリーを一時的に隠すことにより、それらを除外できます (Section 27.14 [Hiding Subdirectories], page 395 を参照してください)。

`% g` *regexp* `RET`
: 内容 *(contents)* に正規表現 *regexp*にたいするマッチを含むすべてのファイルを、(‘*’で) マークします (`dired-mark-files-containing-regexp`)。このコマンドは`% m`と似ていますが、ファイル名ではなくファイル内容を検索する点が異なります。ファイルがEmacs バッファーで visit されていて、`dired-always-read-filesystem`が `nil`(デフォルト) なら、このコマンドはファイルを再 visit せずにバッファーの内容を調べるので、ディスク上のファイル内容が、最後に visit したときから変更されている場合は、矛盾した結果になるかもしれません。これを望まない場合は、このコマンドを呼び出す前に、バッファーで visit されているファイルをリバートしたり、それらのバッファーにたいして `Auto-Revert`モードをオンに切り替えたいと思うかもしれません。Section 15.4 [Reverting], page 157 を参照してください。ファイルをリバートしたり Auto-Revert モードをオンにせずに、このコマンドに常にファイルを再 visit させたい場合は、`dired-always-read-filesystem`を非 `nil`にセットしたいと思うかもしれません。

`C-/`
`C-x u`
`C-_`
: Dired バッファーでの、マークの追加や削除などの変更をアンドゥします (`dired-undo`)。このコマンドは実際のファイル操作をリバートしたり、失われたファイルを回復しません*!* これは単にバッファー自身にたいする変更をアンドゥするだけです。

  ファイルを操作するコマンドの後でこれを使用することにより、問題が起こる場合もあります。たとえば1つ以上のファイルをリネームしてから、Dired バッファーで `dired-undo` により元の名前にリストアした場合、Dired バッファーと、そのディレクトリーの実際の内容が、同期しなくなります。

27.7 ファイルにたいする操作

このセクションでは、1 つまたは複数のファイルを操作する、基本的な Dired コマンドを説明します。これらのコマンドはすべて大文字で、引数を読み取ったり、実際に動作する前に確認を求めるのに、ミニバッファーを使用します。これらのコマンドはすべて、以下の方法により操作するファイルを指定します:

- コマンドに数引数 *n*を与えると、カレントファイルから数えて、次の *n*個のファイルを操作します (*n*が負の場合、コマンドはカレント行の前の、−*n*個のファイルを操作します)。
- 上記以外の場合、‘*’でマークされたファイルがあるときは、コマンドはそれらすべてのファイルを操作します。
- 上記以外の場合、コマンドはカレントファイルだけを操作します。

コマンド`!`や‘`%`’のような他の Dired コマンドにも、操作対象となるファイルを決定するのに、同じ規則を使うものがあります。

Dired バッファーに表示されている 1 つ以上のファイルにたいして、ここで説明した Dired コマンドに加えてバージョンコントロール (VC) のコマンドを呼び出すこともできます。Section 25.1 [Version Control], page 332 を参照してください。

ファイルのコピーやリネーム、それらにたいするリンクを作成するような、対象ディレクトリーを尋ねるコマンドは、操作のためにデフォルトの対象ディレクトリーの推論を試みます。これらのコマンドは通常、Dired バッファーのデフォルトディレクトリーを提案しますが、オプション `dired-dwim-target`が非 `nil`で、何らかのウィンドウに別の Dired バッファーが存在しない場合には、その他のバッファーのディレクトリーをかわりに提案します。`dired-dwim-target`により Dired バッファー

を表示する次のウィンドウ、Dired バッファーを表示するもっとも最近使用されたウィンドウ、あるいはそれ以外の関数の使用のどれを優先するかをカスタマイズできます。値が関数ならそれは引数なしで呼び出されて、デフォルト (デフォルトターゲットおよび “future history”) として使用するディレクトリーのリストをリターンすることが期待されます。

以下は、ファイルにたいして操作を行なう Dired コマンドです。

`C` *new* `RET` 指定されたファイルをコピーします (`dired-do-copy`)。引数 *new*はコピー先のディレクトリー、または (1 つのファイルをコピーする場合は) 新しいファイル名です。これはシェルコマンド `cp`と似ています。

オプション `dired-create-destination-dirs`は Dired がファイルのコピーやリネームで対象先に存在しないディレクトリーがある場合に、それを作成するかどうかを制御します。デフォルト値の `nil`はそのような存在しないディレクトリーを作成しないことを意味します。値 `always`は Dired が自動的にそれらを作成すること、値 `ask`はそれらを作成する前に Dired が確認を求めることを意味します。

`dired-create-destination-dirs`に加えてオプション `dired-create-destination-dirs-on-trailing-dirsep`が非 `nil`なら、対象先ディレクトリーの末尾のディレクトリー区切りは特別に扱われます。‘`test/`’にコピーする際にディレクトリー ‘`test`’が存在しなければ作成して、ソースとして指定したファイルやディレクトリーを新たに作成したディレクトリー内にコピーします。

`dired-copy-preserve-time`が非 `nil`の場合、このコマンドでコピーすることにより、‘`cp -p`’でコピーしたときのように、古いファイルの最終修正時刻 (modification time) が保持されます。

変数 `dired-recursive-copies`は、(‘`cp -r`’のように) ディレクトリーを再帰的にコピーするかを制御します。デフォルトは `top`で、これはディレクトリーを再帰的にコピーする前に、確認を求めることを意味します。

変数 `dired-copy-dereference`はシンボリックリンクをリンクとしてコピーするか、あるいは (‘`cp -L`’のように) 参照剥がし (dereference) の後にコピーするかを制御します。デフォルトは `nil`で、これは新たにシンボリックリンクを作成してシンボリックリンクをコピーすることを意味します。

`dired-keep-marker-copy`はこのコマンドがファイルのマークをどのように扱うかを制御するユーザーオプションです。ファイルの新たなコピーすべてをマークするのは、デフォルトでは ‘`C`’です。

`D` 指定されたファイルを削除します (`dired-do-delete`)。これはシェルコマンド `rm`と似ています。

このセクションの他のコマンドと同様、このコマンドはマークされたファイル、または次の *n*個のファイルを操作します。対照的に `x` (`dired-do-flagged-delete`) は、フラグがついたすべてのファイルを削除します。

`R` *new* `RET` 指定されたファイルをリネームします (`dired-do-rename`)。1 つのファイルをリネームする場合、引数 *new*はファイルの新しい名前です。複数のファイルをリネームする場合、引数 *new*はファイルを移動するディレクトリーです (これはシェルコマンド `mv`と似ています)。

オプション `dired-create-destination-dirs`は Dired が *new*内に存在しないディレクトリーを作成するかどうかを制御します。

対象先ディレクトリー末尾のディレクトリー区切りを特別扱いするかどうかを制御するには、`dired-create-destination-dirs`に加えてオプション

dired-create-destination-dirs-on-trailing-dirsepもセットしてください。これをセットした場合にはディレクトリー‘old’を‘new/’にリネームする際にディレクトリー‘new’が存在しなければ作成して、その新たに作成したディレクトリー配下に‘old’を移動します。セットしていなければ‘old’を‘new’にリネームします。

Diredはリネームされたファイルに関連付けられたバッファーにより visit されたファイルの名前を自動的に変更するので、これらのバッファーは新しい名前を参照します。

変数dired-vc-rename-fileの値が非nilなら、vc-rename-file (Section “Deleting and Renaming Version-Controlled Files” in *Specialized Emacs Features* を参照) を通じて背後にある VCS のコマンドを使用してファイルをリネームします。

H *new* RET 指定されたファイルのハードリンクを作成します (dired-do-hardlink)。これはシェルコマンド lnと似ています。引数 *new*はリンクを中に作成するディレクトリー、または (1 つのリンクだけを作成する場合は) リンクに与える名前です。

S *new* RET 指定されたファイルのシンボリックリンクを作成します (dired-do-symlink)。これは‘ln -s’と似ています。引数 *new*はリンクを中に作成するディレクトリー、または (1 つのリンクだけを作成する場合は) リンクに与える名前です。

Y *new* RET 指定されたファイルの相対的なシンボリックリンクを作成します (dired-do-relsymlink)。引数 *new*はリンクを中に作成するディレクトリー、または (1 つのリンクだけを作成する場合は) リンクに与える名前です。これは dired-do-symlinkと似ていますが、相対的なシンボリックリンクを作成します。

```
foo -> ../bar/foo
```

これは下記のような絶対的なシンボリックリンクは作成しません:

```
foo -> /path/that/may/change/any/day/bar/foo
```

M *modespec* RET
指定されたファイルのモード (パーミッションビットとも呼ばれる) を変更します (dired-do-chmod)。*modespec*には、chmodプログラムで扱われる引数のような、8 進かシンボル表記を指定できます。このコマンドはシンボリックリンクをフォローしないので、シンボリックリンクのモードが変更不可なプラットフォームでシンボリックリンクのモード変更を試みるとエラーが報告されるでしょう。

G *newgroup* RET
指定されたファイルのグループを、*newgroup*に変更します (dired-do-chgrp)。

O *newowner* RET
指定されたファイルの所有者を。*newowner*に変更します (dired-do-chown。ほとんどのシステムでは、これを行なうことができるのはスーパーユーザーだけです)。

変数dired-chown-programは、処理を行なうために使用するプログラムの名前を指定します (システムが異なると、chownが違う場所に配されている場合があるので、この変数が必要)。

T *timestamp* RET
指定されたファイルに touch します。これはファイルの修正時刻を *timestamp* (デフォルトは現在時刻) で更新することを意味します。これはシェルコマンド touchに似ています。

P *command* RET
指定されたファイルを印刷します (dired-do-print)。それらを印刷するためのコマンドを指定しなければなりませんが、ミニバッファー開始時には、変数 lpr-commandお

よび `lpr-switches`(`lpr-buffer`が使用するのと同じ変数。Section 31.7 [Printing], page 476 を参照してください) を使用することにより推定された、適切な初期値が示されます。

`Z` 指定されたファイルを圧縮します (`dired-do-compress`)。そのファイルがすでに圧縮済みのようなら解凍します。マークされたファイルはそれぞれ自身のアーカイブへと圧縮されます。利用可能なら `gzip`、そうでなければ `compress`を使用します。

ディレクトリー名にたいしては、このコマンドはユーザーオプション `dired-compress-directory-default-suffix`に応じて圧縮アーカイブを生成します。デフォルトではそのディレクトリーのすべてのファイルを含む、圧縮アーカイブ`.tar.gz`を生成します。圧縮されたディレクトリーの解凍は、アーカイブファイル`.tar.gz`や`.tgz`上で `Z`をタイプすることにより、ファイル名から拡張子を除いた名前のディレクトリー内にアーカイブ内のすべてのファイルが解凍されます。

`c` 指定されたファイルを、ファイルシステム上のどこかにある 1 つのアーカイブに圧縮します (`dired-do-compress-to`)。デフォルトアーカイブはユーザーオプション `dired-compress-directory-default-suffix`によって制御されます。`dired-compress-files-alist`も参照してください。

`:d` 指定されたファイルを復号化します (`epa-dired-do-decrypt`)。Section “Dired integration” in *EasyPG Assistant User's Manual* を参照してください。

`:v` 指定されたファイルのデジタル署名を検証します (`epa-dired-do-verify`)。Section “Dired integration” in *EasyPG Assistant User's Manual* を参照してください。

`:s` 指定されたファイルにデジタル署名します (`epa-dired-do-sign`)。Section “Dired integration” in *EasyPG Assistant User's Manual* を参照してください。

`:e` 指定されたファイルを暗号化します (`epa-dired-do-encrypt`)。Section “Dired integration” in *EasyPG Assistant User's Manual* を参照してください。

`L` 指定された Emacs Lisp ファイルをロードします (`dired-do-load`)。Section 24.8 [Lisp Libraries], page 327 を参照してください。

`B` 指定された Emacs Lisp ファイルをバイトコンパイルします (`dired-do-byte-compile`)。Section “Byte Compilation” in *The Emacs Lisp Reference Manual* を参照してください。

`I` そのファイルにたいして (Info 形式のファイルとみなして)Info を実行します。

`N` そのファイルにたいして (`nroff`形式のファイルとみなして)man を実行します。

`A` *regexp* `RET`
指定されたすべてのファイルにたいして、正規表現 *regexp*を検索します (`dired-do-find-regexp`)。

このコマンドは `xref-find-references` (Section 25.4.1.3 [Identifier Search], page 360 を参照) の変種で、必要に応じて、Section 25.4.1.2 [Xref Commands], page 359 で説明されているコマンドを使用することにより、マッチ間を移動したり、それらを表示することができる`*xref*`バッファーを表示します。

マークされたファイルやディレクトリーがある場合、このコマンドはそれらのディレクトリーにあるすべてのファイルとサブディレクトリーを再帰的に検索します。ただし名前が `grep-find-ignored-files`にマッチするファイルと、`grep-find-ignored-directories`にマッチするサブディレクトリーは除外されます。

`Q` *regexp* `RET` *to* `RET`

指定された各ファイルにたいして`query-replace-regexp`を行い、*regexp*にたいするマッチを、文字列 *to*に置換します (`dired-do-find-regexp-and-replace`)。

このコマンドは`xref-query-replace-in-results`の変種です。これは *regexp*にたいするすべてのマッチをリストする`*xref*`バッファーを表示します。このバッファーでは特別なコマンド (Section 25.4.1.2 [Xref Commands], page 359 を参照) を使用できます。特に、問い合わせつき置換のループ (query replace loop) を exit した場合は、そのバッファーで更に置換を行なうために`r`を使用できます。Section 25.4.1.3 [Identifier Search], page 360 を参照してください。

`dired-do-find-regexp`と同様に、マークされたファイルがディレクトリーの場合、このコマンドはそれらのディレクトリー内、およびそれらのサブディレクトリーのすべてのファイルの置換を再帰的に行います。ただし、名前が`grep-find-ignored-files`にマッチするファイル、および名前が`grep-find-ignored-directories`にマッチするサブディレクトリーは除外されます。

27.8 Dired でのシェルコマンド

Dired コマンド`!` (`dired-do-shell-command`) は、ミニバッファーでシェルのコマンド文字列を読み取り、そのシェルコマンドを 1 つ以上のファイルにたいして実行します。シェルコマンドが処理するファイルは、Dired コマンドが操作するファイルを決定する方法と同じです (Section 27.7 [Operating on Files], page 387 を参照してください)。コマンド`X`は、`!`の別名です。

コマンド`&` (`dired-do-async-shell-command`) は同じことを行ないますが、これはシェルコマンドを非同期で実行する点が異なります (`!`でシェルコマンドの最後に文字 '`&`'を追加しても、同じことを行なうことができます)。コマンドが複数のファイルを処理する場合、ファイルごとに指定したシェルコマンドの複数コピーを並行で実行します。例外として、指定されたシェルコマンドが '`;`'または '`;&`'で終わる場合、バックグラウンドで実行されるシェルコマンドは、各ファイルにたいして順番に実行されます。Emacs は次のコマンドを実行する前に、呼び出したシェルコマンドが終了するのを待ちます。

`!`と`&`のどちらも、シェルコマンドの作業ディレクトリーは、Dired バッファーのトップレベルのディレクトリーです。

`!`または`&`に複数ファイルを処理させる場合、シェルコマンド文字列は、シェルコマンドにそれらのファイルを渡す方法を決定します。

- コマンド文字列の中で、空白文字に囲まれた '`*`'を使用した場合、コマンドは 1 度だけ実行され、'`*`'はファイル名のリストに置換されます。ファイル名の順番は、Dired バッファーでの出現順になります。

  したがって`! tar cf foo.tar * RET`は、ファイル名全体のリストにたいして`tar`を実行し、それらを tar ファイル`foo.tar`に格納します。

  空白文字で囲まれたシェルのワイルドカードとして '`*`'を使用したい場合は、'`*""`'と記述します。シェルではこれは '`*`'と等価です。しかし '`*`'が空白文字で囲まれていないので、Dired はこれを特別に扱うことができるのです。これを行う場合には、`dired-confirm-shell-command`が`nil`でなければ、Emacs は同意を求めるプロンプトを表示します。

- 上記以外の場合で、コマンド文字列が空白文字で囲まれた '`?`'、または '`` ?` ``'を含むとき、Emacs は各ファイルごとに一度シェルコマンドを実行します。その際、'`?`'および '`` ?` ``'はカレントファイルに置換されます。コマンドで複数回 '`?`'または '`` ?` ``'を使用でき、それぞれ同じファイル名に置換されます。これらを '`*`'とともに使用した場合、このコマンドはエラーをシグナルします。

- コマンド文字列が‘*’、‘?’、‘`?`’のいずれも含まない場合、Emacs は各ファイルにたいしてシェルコマンドを 1 度実行し、その際ファイル名はコマンド文字列の最後に追加されます。たとえば `! uudecode RET`は、各ファイルにたいして `uudecode`を実行します。

もっと複雑な方法でファイル名を繰り返すには、明示的なシェルループを使用したいと思うかもしれません。たとえば以下は、各ファイルを uuencode して、入力ファイル名に‘.uu’を追加して出力ファイル名を作成する例です:

```
for file in * ; do uuencode "$file" "$file" >"$file".uu; done
```

以下は、‘`?`’表記による同じ例です:

```
uuencode ? ? > `?`.uu
```

コマンド`!`および`&`は、新しいファイル、または変更されたファイルを表示するために、Dired バッファーを更新しようとはしません。なぜならこれらのコマンドは、どのファイルが変更されたかを知らないからです。Dired バッファーを更新するには、`g`を使用します (Section 27.15 [Dired Updating], page 396 を参照してください)。

Dired の外でシェルコマンドを実行する情報に関しては、Section 31.5.1 [Single Shell], page 457 を参照してください。

27.9 シェルコマンドの推測

Dired はファイルの名前にもとづいて、そのファイルに適用したいシェルコマンドの推測を試みます。たとえば`foo.tar`という名前のファイルにポイントがあるときに`!`を押下すると、Dired は‘`tar xvf`’を実行したいのだろうと推測して、それをデフォルトのシェルコマンドとして提案します。

`M-n`をタイプすれば、編集用にそのデフォルトをミニバッファーに取り込むことができます。与えられたファイルにたいして複数のコマンドが存在する場合には、`M-n`を何回かタイプすることによってマッチするコマンドを順繰りに確認することができます。

Dired が推測を試みるコマンドは単一のファイルにたいしてであり、マークされたファイルのリストに推測を試みることはありません。

`dired-guess-shell-alist-default` [Variable]
この変数は特定のファイルにたいして適切なシェルコマンドを推測するための、事前定義されたルールを指定します。これを `nil`にセットすると推測は行いません。これらのルールは `dired-guess-shell-alist-user`の要素 (ユーザーが定義) によってオーバーライドされます。

`dired-guess-shell-alist-user` [Variable]
この変数が非 `nil`なら、ファイルの regexp とそれらにたいする推奨コマンドからなるユーザー定義の alist を指定します。これらのルールは `dired-do-shell-command`の実行時において変数 `dired-guess-shell-alist-default`に事前定義されたルールより優先されます。デフォルトは `nil`です。

alist の要素はそれぞれ以下のような形式です

```
(regexp command...)
```

ここで *command*は文字列、あるいは文字列に評価される Lisp 式です。コマンドが複数与えられた場合には、一時的にそれらのコマンドすべてがヒストリーに追加されます。

シェルコマンド内の‘*’は、*regexp*にマッチしたファイルの名前を意味します。Emacs が *command*を呼び出す際に、‘*’はそれぞれマッチしたファイル名によって置き換えられます。

‘.foo’および‘.bar’というファイル拡張子にたいするルールを追加するには、以下を init ファイルに追加してください:

```
(setq dired-guess-shell-alist-user
      (list
       (list "\\.foo$" "foo-command")  ; 固定ルール
       ;; possibly more rules...
       (list "\\.bar$"  ; 条件テスト付きルール
             '(if condition
                  "bar-command-1"
                "bar-command-2"))))
```

同じ拡張子にたいする事前定義されたルールは、すべて上記ルールによって置き換えられます。

他にも M-x customize-group RET dired-guess RETで他のユーザーオプションを確認することができます。

27.10 Dired でのファイル名の変更

このセクションでは、ファイル名を系統的な方法で変更する、Dired コマンドを説明します。各コマンドは、既存の名前を変換することにより作成される新しい名前を使って、マークされたファイルのいくつか、またはすべてのマークされたファイルを操作します。

基本的な Dired のファイル操作コマンド (Section 27.7 [Operating on Files], page 387 を参照してください) と同じように、ここで説明するコマンドは次の *n*ファイル、または‘*’でマークされたすべてのファイル、またはカレントファイルにたいして操作を行ないます (ファイルをマークする方法は Section 27.6 [Marks vs Flags], page 385 で説明されています)。

このセクションで説明するすべてのコマンドは、対話的に機能します。つまり候補となるファイルごとに確認を求めます。したがって、実際に操作が必要なファイルより多くのファイルを選択 (たとえば regexp により多くのファイルを選択) して、コマンドが確認を求めるときに yまたは nをタイプすることにより、選択されたファイルを絞り込むことができます。

% u
: 選択されたファイルを、大文字の名前にリネームします (dired-upcase)。古いファイル名が Fooと barの場合、新しい名前は FOOと BARになります。

% l
: 選択されたファイルを、小文字の名前にリネームします (dired-downcase)。古いファイル名が Fooと barの場合、新しい名前は fooと barになります。

% R *from* RET *to* RET
% C *from* RET *to* RET
% H *from* RET *to* RET
% S *from* RET *to* RET
% Y *from* RET *to* RET
: これらの 5 つはそれぞれリネーム、コピー、ハードリンクの作成、ソフトリンク、および相対的なソフトリンクの作成を行うコマンドであり、古いファイル名にたいして正規表現による置換を行い新しい名前を作成します。

正規表現の置換を行なう 4 つのコマンドは、実際には選択されたファイルの名前にたいして検索と置換を行ないます。これらのコマンドは 2 つの引数を読み取ります。それは式 *from*と、置換パターン *to*です。これらは古いファイル名にマッチした *from*を、*to*に置き換えます。replace-regexp (Section 12.10.2 [Regexp Replace], page 122 を参照してください) のように、古い名前にマッチ

したパターンの全体または一部を参照するために、*to*の中で '`\&`'や '`\digit`'を使用できます。正規表現がファイル名の複数の箇所にマッチする場合は、最初のマッチだけが置換されます。

たとえば`% R ^.*$ RET x-\& RET`は、選択された各ファイルの名前の前に '`x-`'を追加してリネームします。逆に各ファイル名の前の '`x-`'を削除することも可能です。1 つは`% R ^x-\(.*\)$ RET \1 RET`とする方法です。他にも`% R ^x- RET RET`とする方法もあります (ファイル名の範囲を指定するために、アンカーへのマッチ '`^`'および '`$`'を使用しています)。

置換処理は通常、ファイルのディレクトリー名は考慮しません。これはディレクトリーにあるファイル名だけを操作します。しかし数引数を 0 にした場合、置換はディレクトリー名を含めた絶対ファイル名に作用します (非 0 の引数は、操作するファイルの個数を指定します)。

一連のファイルを操作対象に選択するために、それらを操作するときに使うのと同じ正規表現 *from* を使いたいと思うかもしれません。これを行なうには`% m` *from* `RET`でそれらのファイルをマークして、それらのファイルを操作するコマンドで同じ正規表現を使用します。もっと便利にするために、ファイルを操作する`%`コマンドは、最後の`%`コマンドで指定された正規表現をデフォルトで使用します。

27.11 Dired でのファイル比較

`=` (`dired-diff`) コマンドは、`diff`プログラムを使用して、カレントファイル (ポイント位置のファイル) と他のファイル (ミニバッファーを使用して指定) を比較します。ミニバッファーで指定されたファイルは、`diff`の 1 番目の引数となり、ポイント位置のファイルは 2 番目の引数になります。`diff`プログラムの出力は、Diff モードを使用するバッファーで表示されます (Section 15.9 [Comparing Files], page 163 を参照してください)。

リージョンがアクティブの場合、ミニバッファーを使って読み取られるファイル名のデフォルトは、マーク (Dired のマークではなく、Emacs の通常のマークのこと。Section 8.1 [Setting Mark], page 52 を参照してください) の位置のファイルになります。リージョンがアクティブでない場合、ポイント位置のファイルがバックアップファイル (Section 15.3.2 [Backup], page 152 を参照してください) のときは、それがデフォルトになります。

`ediff-files`を使用してファイルを比較することもできます。Section "Major Entry Points" in *Ediff User's Manual* を参照してください。

27.12 Dired でのサブディレクトリー

Dired バッファーは通常、1 つのディレクトリーだけを表示しますが、オプションでサブディレクトリーも同様に含めることができます。

1 つの Dired バッファーに複数のディレクトリーを含める一番簡単な方法は、`ls`にオプション '`-lR`' を指定して実行する方方です (Dired を実行するとき数引数を指定すると、ミニバッファーでこれらのオプションを指定できます)。これはすべてのサブディレクトリーの、すべてのレベルにたいして、再帰的にディレクトリーを一覧します。

特定のサブディレクトリーだけを見たいときもあります。これは `i` (`dired-maybe-insert-subdir`) により行なうことができます:

`i` サブディレクトリーの内容を、バッファーの最後に挿入します。

ディレクトリーを記述する行でこのコマンドを使用した場合、同じ Dired バッファーにそのディレクトリーの内容を挿入して、そこに移動します。サブディレクトリーの内容は、'`ls -lR`'の出力と同じように、Dired バッファーのトップレベルのディレクトリーの後に挿入されます。

サブディレクトリーの内容がすでにバッファーに表示されている場合、`i`コマンドはそこに移動するだけです。

どちらの場合も miは移動する前にマークをセットするので、C-u C-SPCで Dired バッファーの元の位置に戻ることができます (Section 8.1 [Setting Mark], page 52 を参照してください)。‘^’を使って、同じ Dired バッファーの親ディレクトリーに戻ることもできます (Section 27.5 [Dired Visiting], page 384 を参照してください)。

l (dired-do-redisplay) コマンドを使用すると、サブディレクトリーの内容を更新し、サブディレクトリーのヘッダー行で C-u kを使用すると、サブディレクトリーのリストを削除します (Section 27.15 [Dired Updating], page 396 を参照してください)。挿入したサブディレクトリーを、隠したり表示したりすることもできます (Section 27.14 [Hiding Subdirectories], page 395 を参照してください)。

27.13 サブディレクトリー間の移動

Dired バッファーがサブディレクトリーを一覧している場合、ディレクトリー全体を移動するためのページ移動コマンド C-x [および C-x]を使用することができます。

以下のコマンドは、ディレクトリー単位で移動したり、Dired バッファーのディレクトリーツリーを上下に移動します。これらのコマンドは、ディレクトリーヘッダー行に移動します。ディレクトリーヘッダー行とは、ディレクトリーの名前を示す行で、ディレクトリー内容の最初に表示されます。

C-M-n　ディレクトリーのレベルに関係なく、次のサブディレクトリーのヘッダー行に移動します (dired-next-subdir)。

C-M-p　ディレクトリーのレベルに関係なく、前のサブディレクトリーのヘッダー行に移動します (dired-prev-subdir)。

C-M-u　親ディレクトリーのヘッダー行に移動します (dired-tree-up)。

C-M-d　次のサブディレクトリーのヘッダー行に、ディレクトリーツリーを下に移動します (dired-tree-down)。

<　前のディレクトリーファイル行に、上へ移動します (dired-prev-dirline)。これは親ディレクトリーのファイルとして、ディレクトリーが記述されている行のことです。

>　次のディレクトリーファイル行に、下へ移動します (dired-next-dirline)。

M-G　ディレクトリーの入力を求めて、そのディレクトリーファイル行に移動します (dired-goto-subdir)。

27.14 サブディレクトリーを隠す

サブディレクトリーを隠すとは、ヘッダー行を除きそれを非表示にすることを意味します。

$　ポイント位置のサブディレクトリーを隠す、または表示します (dired-hide-subdir)。これは切り替えコマンドです。数引数は繰り返し回数として機能します。

M-$　Dired バッファーのすべてのサブディレクトリーを、ヘッダー行だけを残して隠します (dired-hide-all)。1 つでも隠されたサブディレクトリーがある場合、すべてのサブディレクトリーを再表示します。非常に深いディレクトリーツリーの概観を得たり、離れた場所のサブディレクトリーに素早く移動するために、このコマンドを使うことができます。

通常の Dired コマンドは、隠されたサブディレクトリーの中のファイルは考慮しません。たとえばマークされたファイルを操作するコマンドは、たとえファイルがマークされていても、それが隠されたディレクトリーの中にある場合は、それらを無視します。したがって、サブディレクトリーのファ

イルにつけられた Dired マークを外さなくても、そのサブディレクトリーを一時的に隠せば、操作対象から除外することができます。

サブディレクトリーのリストを挿入する方法については Section 27.12 [Subdirectories in Dired], page 394 を、それを削除する方法については Section 27.15 [Dired Updating], page 396 を参照してください。

27.15 Dired バッファーの更新

このセクションでは、外部 (Dired ではない) のディレクトリーやファイルに対する変更を反映して Dired バッファーを更新したり、Dired バッファーの一部を削除するコマンドを説明します。

`g` Dired バッファーの内容全体を更新します (`revert-buffer`)。

`l` 指定したファイルを更新します (`dired-do-redisplay`)。`l`にたいするファイルの指定は、ファイル操作にたいするファイル指定と同じ方法で行ないます。

`k` 指定したファイル行 (ファイルではなく、その行だけ) を削除します (`dired-do-kill-lines`)。

`s` 並び替えをアルファベット順、日時順に切り替えます (`dired-sort-toggle-or-edit`)。

`C-u s` *switches* `RET`
*switches*を `dired-listing-switches`に使用して、Dired バッファーを更新します。

`g` (`revert-buffer`) とタイプすると、リストされたファイルやディレクトリーにたいする変更にもとづき、Dired バッファーの内容を更新します。消されたファイルを除き、すでにあるマークはすべて保持されます。隠されたサブディレクトリーも更新されますが、隠されたまま残ります。

いくつかのファイルだけ更新するには、`l` (`dired-do-redisplay`) とタイプします。Dired のファイル操作コマンドと同様に、このコマンドは次の nファイル (または前の $-n$ファイル)、または (もしあれば) マークされたファイル、またはカレントファイルを操作します。ファイルの更新とは、ファイルの現在の状態を読み取り、バッファーの対応する行がその状態を示すように、その行を更新することを意味します。

サブディレクトリーのヘッダー行で `l`を使用した場合、対応するサブディレクトリーの内容を更新します。

`C-x d`や他の Dired コマンドを使用して、すでに Dired バッファーで表示されているディレクトリーを visit する場合、Dired はそのバッファーに切り替えますが更新はしません。そのバッファーが最新でない場合、Dired は `g`をタイプして更新するよう警告を表示します。再 visit したとき、自動的に Dired バッファーをリバートするよう Emacs に指示するには、変数 `dired-auto-revert-buffer`を非 `nil`値にセットしてください。

バッファーからファイル行 — 実際のファイルではなく ^^ef^^bd^^b0^^ef^^bd^^b0^^ef^^bd^^b0 を削除するには、`k` (`dired-do-kill-lines`) とタイプします。ファイル操作コマンドと同じように、このコマンドは次の nファイル、またはもしあればマークされたファイルを操作します。しかし、間違って `k`をタイプしたとき面倒なので、カレントファイルは操作しません。

すでに Dired バッファーにサブディレクトリーとして挿入したディレクトリーファイル行で、`k`(Section 27.12 [Subdirectories in Dired], page 394 を参照) とタイプすると、これはサブディレクトリーのリストも同様に削除します。サブディレクトリーのヘッダー行で `C-u k`とタイプしても、Dired バッファーからサブディレクトリーの行を削除できます。

この方法で kill した個別の行は gで元通り表示されますが、サブディレクトリーについては、iを使ってサブディレクトリーを再挿入しなければなりません。

Dired バッファーのファイルは通常、ファイル名のアルファベット順で一覧されます。かわりに Dired は日時順でソートすることもできます。Dired コマンド s (dired-sort-toggle-or-edit) は 2、つのソートモードを切り替えます。Dired バッファーのモードラインには、現在のソート方法が by name や by date のように示されます。

C-u s *switches* RETで、dired-listing-switchesに新しい値を指定できます。

27.16 Dired と find

ファイルを選択するために findユーティリティーを使うことにより、Dired バッファーに表示する一連のファイルを、より柔軟に選択できます。

ワイルドカードパターンに名前がマッチするファイルを検索するには、M-x find-name-dired を使います。これは引数 *directory*と *pattern*を読み取り、*directory*およびそのサブディレクトリーで、名前が *pattern*にマッチするすべてのファイルを選択します。

これにより選択されたファイルは Dired バッファーに表示され、通常の Dired コマンドが利用可能です。

ファイルの名前ではなく内容をテストしたいときは、M-x find-grep-diredを使用します。ものコマンドはミニバッファーで 2 つの引数 *directory*と *regexp*を読み取ります。これは *directory*またはそれのサブディレクトリーで、内容が *regexp*にマッチするすべてのファイルを選択します。このコマンドは、プログラム findおよび grepを実行することにより機能します。Section 24.4 [Grep Searching], page 313 の M-x grep-findも参照してください。正規表現を記述するときは、Emacs にたいしてではなく、grepにたいして記述することを忘れないでください (与えられた regexp に内容がマッチするファイルを表示する別の方法として、% g *regexp*があります。Section 27.6 [Marks vs Flags], page 385 を参照してください)。

この系列でもっとも一般的なコマンドは M-x find-diredで、これは findがテストできる任意の条件を指定することができます。これはミニバッファーで 2 つの引数 *directory*と *find-args*を受け取ります。このコマンドは、findがどのような条件でテストを行なうかを指定する *find-args*を引数として、*directory*で findを実行します。このコマンドを使うためには、findの使い方を理解する必要があります。

これらのコマンドで生成されるリストの書式は変数 find-ls-optionにより制御されます。これは一対のオプションです。1 番目にファイル一覧を生成するために findをどのように呼び出すか、2 番目にその出力を Dired が解析するかを指定します。

コマンド M-x locateは、locateプログラムと似たインターフェースを提供します。M-x locate-with-filterも同様ですが、与えられた正規表現に名前がマッチするファイルだけを保持します。

これらのバッファーは、通常の Dired バッファーと完全に同じようには機能しません。ファイル操作は機能しますが、常にバッファーを自動的に更新するわけではありません。gでバッファーをリバートすると、挿入されたサブディレクトリーは削除され、フラグとマークはすべてリセットされます。

27.17 Dired バッファーの編集

Wdired は、Dired バッファーを直接変更することによりファイル操作を行なうことができる、特別なモードです (“Wdired” の “W” は、“writable: 書き込み可能” が由来です)。Wdired モードに入るには、Dired バッファーで C-x C-q (dired-toggle-read-only) とタイプします。かわりに、メニューアイテムの ‘Immediate / Edit File Names’を使うこともできます。

Wdired モードでは、Dired バッファーに表示されているファイルの名前を編集することにより、ファイルをリネームできます。矩形操作や query-replaceも含めて、通常の Emacs 編集コマンドがすべて利用できます。編集を終えたら C-c C-c (wdired-finish-edit) とタイプします。これにより変更が適用され、通常の Dired モードに戻ります。

単なるファイルのリネーム以外に、ファイルの新しい名前をタイプすることにより (絶対ファイル名でも相対ファイル名でも可)、ファイルを他のディレクトリーに移動することもできます。ファイルに削除のマークをつけるには、ファイル名全体を削除します。シンボリックリンクのリンク先を変更するには、リンク名の隣に表示されているリンク先の名前を編集します。

新たなサブディレクトリーを作成するためにファイル名を編集する場合、Wdired はそれらの新たなディレクトリーを自動的に作成するでしょう。この振る舞いを抑止したい場合は、wdired-create-parent-directoriesを nilにセットしてください。

ファイルサイズや修正時刻など、バッファーの残りのテキストは読み込み専用にマークされているので、編集できません。しかし wdired-allow-to-change-permissionsを tにセットした場合、ファイルのパーミッションを編集できます。たとえば '-rw-r--r--'を '-rw-rw-rw-'に変更することにより、そのファイルを誰でも書き込めるようにできます。これらの変更は、C-c C-cをタイプしたときに効果をもちます。

27.18 Dired でのイメージとサムネイルの閲覧

Image-Dired はイメージファイルをブラウズする機能です。これはイメージファイルにたいして、Emacs 内または外部ビューアーを用いたサムネイルまたはフルサイズでの閲覧を提供します。これは Emacs バッファー内でイメージファイルを visit する Image モードとは異なります (Section 15.19 [Image Mode], page 172 を参照)。

Image-Dired に入るには、Dired バッファーで見たいイメージファイルを、通常のように mを使ってマークします。それから C-t d (image-dired-display-thumbs) とタイプします。これはマークしたファイルに対応する Image-Dired のバッファーを作成して切り替えます。

M-x image-diredとタイプすることにより、直接 Image-Dired に入ることもできます。これはディレクトリーの入力を求めるので、イメージファイルがあるディレクトリーを指定します。これによりそのディレクトリーにある、すべてのイメージファイルのサムネイルが作成され、それらは thumbnail バッファーに表示されます。サムネイルはバックグラウンドで生成されて利用可能になるとロードされます。

サムネイルバッファーにポイントがある際には、RET(image-dired-display-this) とタイプしてそのイメージを別ウィンドウに表示できます。サムネイルバッファー内の移動には、Emacs の標準の移動用キーバインディングや矢印キーを使用します。ポイントを進めて次のイメージを簡単にブラウジングするには SPC (image-dired-display-next) を使用します。ポイントを前のサムネイルに移動して表示するためには、DEL (image-dired-display-previous) を使用してください。

C-RET (image-dired-thumbnail-display-external) とタイプすると、イメージを外部ビューアーで表示します。これは最初に image-dired-external-viewerを設定しなければなりません。

Image-Dired を通じて、イメージを削除することもできます。d (image-dired-flag-thumb-original-file) とタイプすると、Dired バッファーでそのイメージに削除のフラグをつけます。他にも thumbnail バッファーで C-d (image-dired-delete-char) とタイプすれば、イメージに削除フラグをつけなくてもサムネイルイメージを削除することもできます。

"インライン" で操作を行う (Dired バッファーに直接入る) ために Image-Dired を使うこともできます。C-t C-tとタイプすれば、Dired で選択されているイメージの名前の前にサムネイルが表示

されます (image-dired-dired-toggle-marked-thumbs)。C-t iとタイプするとポイントの下にあるイメージを Emacs で表示、C-t xなら外部ビューアーで表示します。

さらに上級の機能として、イメージタグ *(image tags)* があります。これはイメージをカテゴリー分けするために使用されるメタデータです。このタグは、image-dired-tags-db-fileにより設定されるプレーンテキストファイルに格納されます。

イメージファイルにタグ付けするには、それらを Dired バッファーでマークして (thumbnail バッファーで mをタイプして、Dired のファイルをマークすることもできます)、C-t t (image-dired-tag-files) とタイプします。このコマンドはミニバッファーでタグ名読み取ります。特定のタグをもつファイルをマークするには、C-t f (image-dired-mark-tagged-files) とタイプします。特定のタグをもつイメージファイルをマークした後は、C-t dを使ってそれらを閲覧することができます。

thumbnail バッファーから直接ファイルにタグ付けするには t tとタイプし、タグを削除するには t rとタイプします。各ファイルにたいして、“コメント” と呼ばれる特別なタグがあります (コメントは他のタグと正確に同じ意味でのタグではありません。コメントは若干異なる扱いをされるからです)。コメントタグは、イメージについてのコメントや説明を入力するのに使用されます。thumbnail バッファーでコメントするには、cとタイプします。これはコメントの入力を求めます。Dired からコメントを追加するには C-t c (image-dired-dired-comment-files)、C-t e (image-dired-dired-edit-comment-and-tags) とタイプするとコメントとタグを編集するためのバッファーを表示します。

image-dired-thumb-visible-marksが非 nil (デフォルト) なら、Dired でマークしたファイルは Image-Dired でもマークされます。

Image-Dired は、シンプルなイメージ操作も提供します。thumbnail バッファーで Lとタイプすると、オリジナルのイメージを反時計回りに 90 度ローテートし、Rで時計回りにローテートします。このローテーションは損失がなく、jpegtranと呼ばれる外部ユーティリティーを使用し、これは最初にインストールする必要があります。

27.19 その他の Dired の機能

Dired はデフォルトでは利用可能なディスクスペースを最初の行に表示します。これは変数 dired-free-spaceの値が firstの場合です。かわりに separateにセットすれば、(カレントディレクトリー内のファイルが占めるスペースを含めて) 別の行に表示します。nilにセットした場合には、空きスペースの表示を何も行いません。

コマンド+ (dired-create-directory) はディレクトリー名を読み取り、そのディレクトリーを作成します。そのディレクトリーがすでに存在する場合は、エラーをシグナルします。

このコマンド (dired-create-empty-file) はファイル名を読み取りファイルを作成します。そのファイルがすでに存在すればエラーをシグナルします。

コマンド M-s a C-s (dired-do-isearch) は、マークされたファイルにたいして、複数ファイルにたいするインクリメンタル検索を開始します。ファイル終端で検索が失敗した場合、C-sとタイプすると、マークされた次のファイルで検索を繰り返します。マークされた最後のファイルの終端に達すると、マークされた最初のファイルに戻って検索します。コマンド M-s a M-C-s (dired-do-isearch-regexp) は、同じことを正規表現検索で行ないます。検索の繰り返しに関する情報は、Section 12.1.2 [Repeat Isearch], page 106 を参照してください。

コマンド w (dired-copy-filename-as-kill) は、マークされたファイル (または次の *n* ファイル) の名前を、あたかも C-wで kill したかのように、kill リングに配します。ファイル名はスペースで区切られます。

プレフィクス引数に 0 を指定した場合、マークされた各ファイルの絶対ファイル名を使用します。プレフィクス引数に C-uだけを指定した場合、Dired バッファーのデフォルトディレクトリーからの相対ファイル名を使用します (サブディレクトリーから行なった場合はスラッシュが含まれます)。特別なケースとして、ポイントがディレクトリーのヘッダー行にある場合、wで、そのディレクトリーの絶対パスを得ることができます。この場合、プレフィクス引数とマークされたファイルは無視されます。

このコマンドの主な目的は、他の Emacs コマンドの引数として、ファイル名を yank することです。これは kill リングに何が追加されたかも表示するので、現在マークされているファイルを、エコーエリアに表示するためにも使用できます。

ファイルリストの中に HTML ファイルがある場合は、そのファイルをブラウザで閲覧できると便利かもしれません。W (browse-url-of-dired-file) コマンドは、そのファイルの閲覧に標準に設定されたブラウザを使用します。

コマンド ((dired-hide-details-mode) は、カレントの Dired バッファーでの、ファイル所有者やファイルパーミッションなどの、詳細表示を切り替えます。デフォルトでは、シンボリックリンクのリンク先や、ヘッダー行とファイルやディレクトリーの一覧以外のすべての行も隠します。これを変更するには、オプション dired-hide-details-hide-symlink-targetsと dired-hide-details-hide-information-linesをカスタマイズしてください。

visit しているディレクトリーが、バージョンコントロール (Section 25.1 [Version Control], page 332) の配下にある場合、選択されたファイルにたいして、通常の VC diff および VC log コマンドが処理を行ないます。

コマンド M-x dired-compare-directoriesは、カレントの Dired バッファーと、他のディレクトリーを比較するために使用されます。2 つのディレクトリーで異なる、すべてのファイルがマークされます。カレントの Dired バッファーも含めて、それらのファイルをリストする、すべての Dired バッファーでマークが付されます。

(入力を求められたとき RETをタイプすると使用される) デフォルトの比較方法では、ファイル名だけを比較します — つまり別のディレクトリーに存在しないファイルは、異なります。Lisp 式を入力して、より厳密な比較を指定することができます。この場合、変数 size1と size2はファイルサイズ、mtime1と mtime2は最終修正時刻を秒で表した浮動小数点数、fa1と fa2はファイルの属性リスト (関数 file-attributesで返される形式) です。この比較は、同じ名前のファイルごとに評価され、式の値が非 nilの場合、それらのファイルは異なります。

たとえばシーケンス M-x dired-compare-directories RET (> mtime1 mtime2) RETは、別のディレクトリーよりこのディレクトリーのファイルが新しい場合はマークし、このディレクトリーより別のディレクトリーのファイルが古い場合にマークします。相手がいないファイルも、両方のディレクトリーでマークが付されます。

X ウィンドウシステムでは、Emacs はドラッグアンドドロップのプロトコルをサポートします。別のプログラムからファイルオブジェクトをドラッグして Dired バッファーにドロップできます。これによってそのディレクトリーにファイルの移動、コピー、あるいはリンクが作成されます。どのアクションが採用されるかは、正確には元のプログラムによって決定されます。ユーザーオプション dired-mouse-drag-filesを有効にすることによって、Dired バッファーの外部へのファイルのドラッグもサポートされるので、他のプログラムへのファイルのドラッグにマウスを使うことができます。linkにセットすると他のプログラム (通常はファイルマネージャーなど) がそのファイルへのシンボリックリンクを作成、moveにセットすると他のプログラムはそのファイルを新たな場所に移動するでしょう。それ以外の非 nilにセットした場合には、他のプログラムはそのファイルをオープンするか、ファイルのコピーを作成します。ドラッグアンドドロップ操作の間にキーボードの修飾キーを押下することによって、そのファイルにたいして他のプログラムがどのアクションを行なうか制御することもできます。

28 カレンダーとダイアリー

Emacs は、過去のイベントや将来のプランのためのダイアリーをもつ、卓上カレンダー機能を提供します。これにはアポイントメントを管理したり、特定のプロジェクトにどれだけ時間を費やしているか追跡する機能もあります。

カレンダーを起動するには、`M-x calendar`とタイプします。これは 3ヶ月分のカレンダーを表示します。当月が中央に表示され、今日の日付にポイントが配されます。`C-u M-x calendar`のように数引数を指定すると、3ヶ月カレンダーの中央に表示する、月と年の入力を求めます。カレンダーは自身のバッファーを使用し、メジャーモードは Calendar モードです。

カレンダーの中で `mouse-3`をクリックすると、特定の日付にたいする操作メニューが表示されます。`mouse-2`をクリックすると、特定の日付とは関係のない、一般的に使用されるカレンダー機能のメニューが表示されます。カレンダーを終了するには、`q`とタイプします。

このチャプターでは、基本的なカレンダー機能を説明します。より進んだトピックについては、Section “Advanced Calendar/Diary Usage” in *Specialized Emacs Features* を参照してください。

28.1 カレンダーでの移動

Calendar モードは日、週、月、年のような論理的な単位で、カレンダーを移動するコマンドを提供します。最初に表示されている 3ヶ月の外に移動すると、カレンダーの表示は選択された日付が表示されるように、自動的にスクロールします。ある日付に移動することにより、その日が何の休日かを表示したり、ダイアリーのエントリーを表示したり、他のカレンダーに変換できます。単にカレンダーをスクロールするのは、長い期間を移動するのにも便利です。

28.1.1 標準的な時間間隔での移動

カレンダーバッファーでの移動コマンドは、テキスト内での移動コマンドと似ています。日、週、月、年を前方および後方に移動できます。

`C-f`　　ポイントを、1 日前方に移動します (`calendar-forward-day`)。

`C-b`　　ポイントを、1 日後方に移動します (`calendar-backward-day`)。

`C-n`　　ポイントを、1 週前方に移動します (`calendar-forward-week`)。

`C-p`　　ポイントを、1 週後方に移動します (`calendar-backward-week`)。

`M-}`　　ポイントを、1 月前方に移動します (`calendar-forward-month`)。

`M-{`　　ポイントを、1 月後方に移動します (`calendar-backward-month`)。

`C-x ]`　　ポイントを、1 年前方に移動します (`calendar-forward-year`)。

`C-x [`　　ポイントを、1 年後方に移動します (`calendar-backward-year`)。

day および week のコマンドは、文字単位または行単位で移動するための Emacs コマンド、自然に類似しています。通常 `C-n`は、後続の行の同じ列に移動しますが、Calendar モードでは翌週の同じ日に移動します。`C-p`は、前の週の同じ日に移動します。

他のモードで通常振る舞うように、矢印キーは `C-f`、`C-b`、`C-n`、`C-p`と等価です。

月単位および年単位の移動のためのコマンドは、週単位の移動コマンドに似ていますが、より長い期間を移動します。month コマンドの `M-}`および `M-{`は、月全体を前方または後方に移動します。year コマンドの `C-x ]`および `C-x [`は、年全体を前方または後方に移動します。

これらのコマンドを簡単に覚える方法は、月および年が、テキストのパラグラフとページに類似するものだと考えることです。しかし、カレンダーの移動コマンド自体は、テキスト間の移動コマンドと非常に類似している訳ではありません。Emacs のパラグラフコマンドは通常、パラグラフの先頭または最後に移動しますが、月コマンドと年コマンドは、月または年の同じ日を保ちながら、月単位または年単位で移動します。

これらのコマンドは、繰り返し回数として数引数を受け取ります。Calendar モードでは便宜上、メタ修飾なしの数字キーとマイナス記号で、数引数を指定できます。たとえば 100 C-fは、現在の位置から 100 日前方へポイントを移動します。

28.1.2 週、月、年の開始と終了

週 (または月や年) は、単に日が集まったものではなく、特定に日付から開始されると考えます。そのため Calendar モードは週、月、年の、最初と最後に移動するコマンドを提供します。

C-a 週の最初にポイントを移動します (calendar-beginning-of-week)。

C-e 週の最後にポイントを移動します (calendar-end-of-week)。

M-a 月の最初にポイントを移動します (calendar-beginning-of-month)。

M-e 月の最後にポイントを移動します (calendar-end-of-month)。

M-< 年の最初にポイントを移動します (calendar-beginning-of-year)。

M-> 年の最後にポイントを移動します (calendar-end-of-year)。

これらのコマンドは繰り返し回数として数引数を受け取り、繰り返し回数で、何週、何ヶ月、何年前方または後方へ移動するかを示します。

デフォルトでは、週は日曜日から開始されます。週が月曜日から開始されるようにするには、変数 calendar-week-start-dayを 1 にセットします。どの日付へッダーを週末としてハイライトするか変更するには、変数 calendar-weekend-daysをセットしてください。

28.1.3 日付の指定

Calendar モードは、さまざまな方法で、指定された特定の日付へ移動するコマンドを提供します。

g d 指定した日付にポイントを移動します (calendar-goto-date)。

g D 年の指定された日にポイントを移動します (calendar-goto-day-of-year)。

g w 年の指定された週にポイントを移動します (calendar-iso-goto-week)。

o 指定された月をカレンダーの中央にします (calendar-other-month)。

. 今日の日付にポイントを移動します (calendar-goto-today)。

g d (calendar-goto-date) は年、月、月の何日目かの入力を求め、その日付に移動します。カレンダーは西暦の最初からすべての日付を含むので、年は全部タイプしなければなりません。つまり、‘10’ではなく ‘2010’とタイプしてください。

g D (calendar-goto-day-of-year) は、年と日数の入力を求め、その日付に移動します。負の日数は年の最後から後方に数えます。g w (calendar-iso-goto-week) は、年と週数の入力を求め、その週に移動します。

o (calendar-other-month) は、月と年の入力を求め、3ヶ月カレンダーの中央にその月を配します。

. (calendar-goto-today) で今日の日付に戻ることができます。

28.2 カレンダーでのスクロール

カレンダーの表示は、ポイントが表示部分の外に移動すると、自動的にスクロールします。手動でスクロールすることもできます。カレンダーのウィンドウに、月が書き込まれた細長い紙が含まれているのを想像してみてください。カレンダーのスクロールとは、この紙を水平方向に動かして、そのウィンドウに新しい月を表示することを意味します。

`>` カレンダーを 1 月前方にスクロールします (`calendar-scroll-left`)。

`<` カレンダーを 1 月後方にスクロールします (`calendar-scroll-right`)。

`C-v`
`PageDown`
`next` カレンダーを 3 月前方にスクロールします (`calendar-scroll-left-three-months`)。

`M-v`
`PageUp`
`prior` カレンダーを 3 月後方にスクロールします (`calendar-scroll-right-three-months`)。

もっとも基本的なカレンダーのスクロールコマンドは、1 度に 1 月スクロールします。これはコマンドの前後の表示で、2 月がオーバーラップして表示されることを意味します。`>`は内容を、1 度に 1 月前方にスクロールします。`<`は内容を、1 度に 1 月後方にスクロールします。

コマンド `C-v`および `M-v`は、カレンダーを画面全体 — つまり 3 月スクロールします。この動作は、これらのコマンドが通常意味する動作と類似しています。`C-v`は後の日付を表示し、`M-v`は前の日付を表示します。これらのコマンドは、繰り返し回数として数引数を受け取ります。特に `C-u`は次のコマンドを 4 倍するので、`C-u C-v`は 1 年前方にカレンダーをスクロールし、`C-u M-v`は 1 年後方にカレンダーをスクロールします。

他のモードのときと同様に、ファンクションキーの `PageDown` (または `next`) と `PageUp` (または `prior`) は、`C-v`や `M-v`と等価です。

28.3 日付のカウント

`M-=` カレントリージョン内の日数を表示します (`calendar-count-days-region`)。

ある範囲の日数を数えるには、`C-SPC`で日付をマークし、他の日付にポイントを移動して、`M-=` (`calendar-count-days-region`) とタイプします。表示される日数は包括的 *(inclusive)* です。つまりマークとポイントで指定された日も、日数に含まれます。

28.4 その他のカレンダーコマンド

`p d` day-in-year(1 年で何日目か) を表示します (`calendar-print-day-of-year`)。

`C-c C-l` カレンダーウィンドウを再生成します (`calendar-redraw`)。

`SPC` 次のウィンドウを上にスクロールします (`scroll-other-window`)。

`DEL`
`S-SPC` 次のウィンドウを下にスクロールします (`scroll-other-window-down`)。

`q` カレンダーを exit します (`calendar-exit`)。

年の初めからの経過日数と、年の最後までの残り日数を表示するには、`p d` command (`calendar-print-day-of-year`) とタイプします。これは、それら両方の日数をエコーエリアに表示します。経過日数には選択された日付も含まれます。残り日数に、その日は含まれません。

カレンダーウィンドウのテキストがおかしくなった場合は、再描画するために `C-c C-l` (`calendar-redraw`) とタイプします (これは Calendar モード以外のモードの編集コマンドを使ったときだけ発生します)。

Calendar モードでは、`SPC` (`scroll-other-window`) と `DEL` (`scroll-other-window-down`) を使って、(もしあれば) 他のウィンドウを上または下にスクロールします。これは休日のリストやダイアリーのエントリーを他のウィンドウに表示しているときなどに便利です。

カレンダーから exit するには、`q` (`calendar-exit`) とタイプします。これはカレンダーに関連するすべてのバッファーを隠し (bury)、他のバッファーを選択します (フレームがカレンダー専用のウィンドウを含む場合、カレンダーを exit することによりそのフレームは、削除またはアイコン化されます。これは変数 `calendar-remove-frame-by-deleting`の値に依存します)。

28.5 カレンダーファイルの記述

カレンダーとダイアリーのエントリーを、HTML や LaTeX に書き出すことができます。

Calendar HTML コマンドは、カレンダー、休日、ダイアリーのエントリーを含む、HTML コードのファイルを生成します。各ファイルは 1ヶ月に対応し、*yyyy-mm*`.html`という形式の名前をもちます。ここで *yyyy*は 4 桁の年、*mm*は 2 桁の月です。変数 `cal-html-directory`は、HTML ファイルのためのデフォルトの出力先ディレクトリーを指定します。休日が表示されないようにするには、`cal-html-holidays`をカスタマイズしてください。

<と>で囲まれたダイアリーのエントリー (たとえば、this is a diary entry with some red text) は、HTML タグと解釈されます。HTML ファイルを含むディレクトリーの中にあるスタイルシート `cal.css`を通じて、表示される HTML ページの全体的な外観を変更できます (関連するスタイルセッティングについては、変数 `cal-html-css-default`の値を参照してください)。

`H m` 1ヶ月のカレンダーを生成します (`cal-html-cursor-month`)。

`H y` 1 年の各月のカレンダーファイルを生成し、同様にインデックスページも生成します (`cal-html-cursor-year`)。デフォルトでは、このコマンドはファイルをサブディレクトリー *yyyy*に書き込みます。これを変更した場合、各年の間のハイパーリンクは機能しなくなるでしょう。

変数 `cal-html-print-day-number-flag`が非 `nil`の場合、マンスリーカレンダーには day-of-the-year(年の何日目か) の数字が表示されます。変数 `cal-html-year-index-cols`は、年のインデックスページの列数を指定します。

Calendar LaTeX コマンドは、カレンダーとしてプリントされる LaTeX コードのバッファーを生成します。どのコマンドを使うかにより、プリントされるカレンダーは、ポイントのある日、週、月、年をカバーします。

`t m` 1ヶ月のカレンダーを生成します (`cal-tex-cursor-month`)。

`t M` 横向き印刷 (sideways-printing) で、1ヶ月のカレンダーを生成します (`cal-tex-cursor-month-landscape`)。

`t d` 1 日のカレンダーを生成します (`cal-tex-cursor-day`)。

`t w 1` 時間を併記した、1 週間の 1 ページカレンダーを生成します (`cal-tex-cursor-week`)。

`t w 2` 時間を併記した、1 週間の 2 ページカレンダーを生成します (`cal-tex-cursor-week2`)。

`t w 3` 時間なしで、1 週間の ISO スタイルのカレンダーを生成します (`cal-tex-cursor-week-iso`)。

`t w 4` 時間を併記した、月曜で始まる 1 週間のカレンダーを生成します (`cal-tex-cursor-week-monday`)。

`t w W` 時間なしで、1 週間の 2 ページカレンダーを生成します (`cal-tex-cursor-week2-summary`)。

`t f w` 2 週間を見開きする、Filofax スタイルのカレンダーを生成します (`cal-tex-cursor-filofax-2week`)。

`t f W` 1 週間を見開きする、Filofax スタイルのカレンダーを生成します (`cal-tex-cursor-filofax-week`)。

`t y` 1 年のカレンダーを生成します (`cal-tex-cursor-year`)。

`t Y` 横向き印刷 (sideways-printing) で、1 年のカレンダーを生成します (`cal-tex-cursor-year-landscape`)。

`t f y` 1 年の Filofax スタイルのカレンダーを生成します (`cal-tex-cursor-filofax-year`)。

これらのコマンドの中には、カレンダーを (ランドスケープモードで) 横向きに印刷するものがあり、その場合は縦長ではなく横長になります。Filofax の用紙サイズ (3.75in x 6.75in) を使うものもあります。これらのコマンドはすべてプレフィクス引数を指定でき、これは何日、何週間、何ヶ月、何年分を印刷するかを指定します (常に選択された日付から開始されます)。

変数 `cal-tex-holidays`が非 `nil`(デフォルト) の場合、プリントされたカレンダーには `calendar-holidays`の休日が表示されます。変数 `cal-tex-diary`が非 `nil`(デフォルトは `nil`) の場合、ダイアリーのエントリーも含まれます (マンスリー、Filofax、iso の週カレンダーのみ)。変数 `cal-tex-rules`が非 `nil`(デフォルトは `nil`) の場合、カレンダーは充分な空間のある、罫線ページのスタイルで表示されます。そのカレンダーがどんな機能をサポートするかは、個別の cal-tex 関数のドキュメントを参照してください。

必要なら、変数 `cal-tex-preamble-extra`を使用して、生成されたドキュメントのプリアンブル (preamble) に、追加の LaTeX コマンドを挿入できます。

28.6 休日

Emacs のカレンダーは、多くのメジャーおよびマイナーな休日を知っており、それらを表示することができます。デフォルトリストに、あなた自身の休日を追加できます。

`mouse-3 Holidays`

`h` 選択された日付の休日を表示します (`calendar-cursor-holidays`)。

`x` カレンダーウィンドウの休日をマークします (`calendar-mark-holidays`)。

`u` カレンダーウィンドウのマークを外します (`calendar-unmark`)。

`a` 表示された 3ヶ月の休日を別のウィンドウにリストします (`calendar-list-holidays`)。

`M-x holidays`

今日を含む 3ヶ月の休日を別のウィンドウにリストします。

`M-x list-holidays`
年の指定した範囲の休日を別のウィンドウにリストします。

ある日付が休日かどうか見るには、カレンダーウィンドウのその日付にポイントを置いて、hコマンドを使用します。その日付を mouse-3でクリックして表示されるメニューで、Holidaysを選択しても見ることができます。どちらの方法も、その日付にたいする休日を、エコーエリアか、エコーエリアに収まらない場合は別のウィンドウに表示します。

カレンダーに表示された、すべての日付にたいする休日の分布を閲覧するには、xコマンドを使用します。これは休日を異なるフェイスで表示します。Section "Calendar Customizing" in *Specialized Emacs Features* を参照してください。このコマンドは、現在表示されている月と、スクロールすることにより見ることができる他の月の両方に適用されます。マークをオフにして現在のマークを消すには、uをタイプします。これはダイアリーマークも消します (Section 28.10 [Diary], page 411 を参照してください)。変数 calendar-mark-holidays-flagが非 nilの場合、カレンダーの作成または更新により、自動的に休日がマークされます。

さらに詳細な情報を得るには、aコマンドを使用します。これはカレントの 3ヶ月の範囲のすべての休日を含むリストを、別のバッファーに表示します。カレンダーウィンドウで SPCおよび DELを使用することにより、そのリストを上または下にスクロールできます。

コマンド M-x holidaysは、今月および前月と来月の休日のリストを表示します。これはカレンダーウィンドウがないときでも機能します。変数 calendar-view-holidays-initially-flagが非 nilの場合、カレンダーの作成により、休日がこの方法で表示されます。違う月を中心にした 3ヶ月の休日リストが欲しい場合は、月と年の入力を求める C-u M-x holidaysを使用してください。

Emace が知っている休日には、United States、major Bah^^c3^^a1'^^c3^^ad、Chinese、Christian、Islamic、Jewish の休日、および夏至と冬至 (solstices)、春分と秋分 (equinoxes) が含まれます。

M-x holiday-listは、ある年の範囲にたいする休日のリストを表示します。この関数は年の開始と終了の入力を求め、すべての休日、または複数のカテゴリーの中の 1 つのカテゴリーに属する休日を選択することができます。このコマンドは、カレンダーウィンドウがないときでも使用することができます。

休日に使用される日付は、歴史的事実ではなく、現在の慣習 *(current practice)* にもとづきます。たとえば Veteran's Day(退役軍人の日) は 1919 年に始まった休日ですが、それより前の年でも表示されます。

28.7 日の出と日の入りの時刻

特別なカレンダーコマンドにより、任意の日付にたいする日の出 (sunrise) と日の入り (sunset) の時刻を、1、2 分の範囲で調べることができます。

`mouse-3 Sunrise/sunset`
`S` 選択された日付の、日の出と日の入りの時刻を表示します (calendar-sunrise-sunset)。

`M-x sunrise-sunset`
今日の日の出と日の入りの時刻を表示します。

`C-u M-x sunrise-sunset`
指定した日付の日の出と日の入りの時刻を表示します。

`M-x calendar-sunrise-sunset-month`
選択された月の日の出と日の入りの時刻を表示します。

カレンダーからは、見たい日付にポイントを移動して Sをタイプすると、エコーエリアに日の出と日の入りの時刻を、地方時間で表示します。日付を mouse-3でクリックし、表示されたメニューの‘Sunrise/sunset’を選択しても表示することができます。今日、または指定した日付にたいしてこの情報を表示するために、カレンダーの外からでも、コマンド M-x sunrise-sunsetが利用可能です。今日以外の日付を指定するには、年、月、日の入力を求める C-u M-x sunrise-sunsetを使用してください。

C-u C-u M-x sunrise-sunsetにより、任意の地域の任意の日付にたいして、日の出と日の入りの時刻を表示できます。これは経度 (longitude)、緯度 (latitude)、調整済みグリニッジ平均時 (Coordinated Universal Time) との分差、日付の入力を求め、その地域の、その日付の日の出と日の入りの時刻を表示します。

日の出と日の入りの時刻は地球上の位置に依存するので、これらのコマンドを使用する前に、Emacs に経度 (latitude)、緯度 (longitude)、地域名 (location name) を指定する必要があります。以下は何をセットするかの例です:

```
(setq calendar-latitude 40.1)
(setq calendar-longitude -88.2)
(setq calendar-location-name "Urbana, IL")
```

calendar-latitudeと calendar-longitudeには、小数点以下 1 位までの数値を使用します。

タイムゾーン (time zone) も、地方時間での日の入りと日の出の時刻に影響を与えます。Emacs は通常、タイムゾーンの情報をオペレーティングシステムから得ますが、その情報が間違っている場合 (またはオペレーティングシステムがその情報を提供しない場合)、これらを自分でセットしなければなりません。以下は例です:

```
(setq calendar-time-zone -360)
(setq calendar-standard-time-zone-name "CST")
(setq calendar-daylight-time-zone-name "CDT")
```

calendar-time-zoneの値は地方標準時 (local standard time) と、調整済みグリニッジ平均時 (Coordinated Universal Time)、またはグリニッジ時 (Greenwich time: GMT) との分差です。calendar-standard-time-zone-nameと calendar-daylight-time-zone-nameの値は、あなたのタイムゾーンで使用される略語です。Emacs は、サマータイム *(daylight saving time)* で調整済みの、日の出と日の入りの時刻を表示します。サマータイムが決定される方法については、Section 28.11 [Daylight Saving], page 418 を参照してください。

(‘"+0100"’のような) 数値的タイムゾーンを表示したければ、これを (‘"CET"’のような) シンボルでなく numericにセットしてください。

ユーザーとしては.emacsで、普段いる場所をカレンダーの位置変数にセットすると便利だと思うかもしれません。システム管理者としては、default.elで、すべてのユーザーにたいして、これらの変数をセットしたいと思うかもしれません。Section 33.4 [Init File], page 528 を参照してください。

28.8 月の位相

以下のカレンダーコマンドは、月の位相 (phases of the moon)、つまり新月 (new moon)、上弦 (first quarter)、満月 (full moon)、下弦 (last quarter) の日付と時刻を表示します。この機能は、月の位相に依存する問題をデバッグするのに便利です。

M 表示されている 3ヶ月にたいして、月のすべての 4 位相の日付と時刻を表示します (calendar-lunar-phases)。

`M-x lunar-phases`
今日を含む3ヶ月にたいして、月のすべての 4 位相の日付と時刻を表示します。

カレンダーからは`M`を使用して、カレントの3ヶ月の範囲での月の位相を、別のバッファーに表示します。数分の誤差で、日付と時刻がリストされます。

カレンダーの外からは、コマンド`M-x lunar-phases`で今月と前後 1ヶ月の、月の位相のリストが表示されます。違う月についての情報は、月と年の入力を求める`C-u M-x lunar-phases`を使用してください。

月の位相にたいして与えられる日付と時刻は、(それが適切なときはサマータイム調整されて) 地方時間で与えられます。前のセクションの議論を参照してください。Section 28.7 [Sunrise/Sunset], page 406 を参照してください。

28.9 他のカレンダーとの間の変換

Emacs のカレンダーは、常にグレゴリオ暦 (Gregorian calendar) で表示されます。これは、新暦 (New Style calendar) と呼ばれることもあり、今日では世界のほとんどで使用されています。しかし、このカレンダーは 16 世紀より前には存在しておらず、、18 世紀より前は広く使われていませんでした。ユリウス暦 (Julian calendar) を置き換えて、世界的に受け入れられたのも 20 世紀初頭です。Emacs カレンダーは、西暦 1 年 1 月から任意の月を表示できますが、グレゴリオ歴がまだ存在しない日付にたいしても、カレンダーは常にグレゴリオ暦で表示されます。

Emacs は他のカレンダーを表示できませんが、日付を他のカレンダーに変換することができます。

28.9.1 サポートされるカレンダーシステム

ISO 商用暦 (ISO commercial calendar) は、ビジネスで使われることがあります。

ユリウス暦 (Julian calendar) は、ジュリアス・シーザー (Julius Caesar) から名前がつけられたカレンダーで、中世ヨーロッパで使用され、19 世紀まで多くの国で使用されてきました。

天文学者は、ユリウス歴の BC4713 年 1 月 1 日正午からの、単純な通算日を使用します。この経過日はユリウス日 *(Julian day: JD)*、または天文日 *(Astronomical day)* と呼ばれます。

ヘブライ暦 (Hebrew calendar) は伝統的にユダヤ教 (Jewish religion) で使用されます。Emacs のカレンダープログラムは Jewish の休日を決定するためにヘブライ暦を使用します。ヘブライ暦では日没が 1 日の開始と終了です。

イスラム暦 (Islamic calendar) は、主に Islamic 国で広く使用されます。Emacs は Islamic の休日の日付を決定するのに、これを使用します。Islamic の世界では、カレンダーについての世界的な取り決めが存在しません。Emacs は広く受け入れられているバージョンを使用しますが、Islamic の休日の正確な日付は、計算によってではなく、宗教的な権威による公告に依存することがしばしばあります。結果として、行事の正確な日付は Emacs が計算する日付と若干違う場合があります。イスラム暦では、日没が 1 日の開始と終了です。

フランス革命暦 (French Revolutionary calendar) は、より非宗教的で自然にもとづいた観点により 1 年のサイクルを表すために、1789 年の革命後にジャコバン派 (Jacobins) により作成されました。これはメートル法のような、合理的な目安により 1 週が 10 日に設定されます。フランス政府は 1805 年の終わりに、このカレンダーを公式に放棄しました。

中央アメリカのマヤには、互いに個別で重複した 3 つのカレンダーシステム *long count*、*tzolkin*、*haab* があります。Emacs はこれら 3 つのカレンダーすべてを知っています。専門家はマヤのカレンダーと私たちのカレンダーとの正確な関連を議論しています。Emacs は、Goodman-Martinez-Thompson の相関関係を使用して計算を行ないます。

コプト教徒 (Copts: エジプト教会のクリスチャン) は、エジプト古代の太陽暦にもとづいたカレンダーを使用します。彼らのカレンダーは 30 日からなる 12 の月と、その後に余分の 5 日が含まれます。4 年に 1 度、余分な 5 日に閏日を加えて 6 日とします。エチオピア暦は構造的に同じですが、異なる年数と月の名前をもちます。

ペルシャ人は、Omar Khayyam のデザインにもとづく太陽暦を使用します。かれらのカレンダーは 12 の月を含み、最初の 6ヶ月は 31 日、次の 5ヶ月は 30 日、最後の月は、通常の年は 29 日で、閏年は 30 日です。閏年は 4 年または 5 年後との複雑なパターンで発生します。ここで実装されているカレンダーは、Birashk により指示されている数学的なペルシャ歴で、2820 年周期にもとづいています。天文学的なペルシャ暦との違いは、それが天文学的なイベントにもとづいている点です。これを記述している時点で、2 つの歴に最初に矛盾が発生すると予測されるのは、2025 年の 3 月 20 日です。現時点では、そのときのイランの公式カレンダーが何になるかは、明確ではありません。

旧暦 (Chinese calendar) は、太陽年に太陰月を組み込んだ複雑なシステムです。年は 60 周期で、各年は通常の年は 12ヶ月、閏年は 13ヶ月です。各月は通常の月は 29 日または 30 日です。年、通常月、日は 10 の *celestial stems* と、12 の *terrestrial branches* の組み合わせにより名前がつけられ、これが合計で 60 個の名前となり、60 回周期で繰り返されます。

Bah^^c3^^a1'^^c3^^ad のカレンダーシステムは、19 日を持つ 19ヶ月の太陽周期にもとづきます。のこり 4 日の閏日 (intercalary days) は、18ヶ月目と 19ヶ月目の間に配されます。

28.9.2 他のカレンダーへの変換

以下のコマンドは、選択された日付 (ポイント位置の日付) をさまざまなカレンダーシステムで記述します:

`mouse-3 Other calendars`

`p o` 選択された日付を、他のさまざまなカレンダーで表示します (`calendar-print-other-dates`)。

`p c` 選択された日付を、ISO 商用暦 (ISO commercial calendar) で表示します (`calendar-iso-print-date`)。

`p j` 選択された日付を、ユリウス日 (Julian date) で表示します (`calendar-julian-print-date`)。

`p a` 選択された日付を、ユリウス日にもとづく天文日 (astronomical day) で表示します (`calendar-astro-print-day-number`)。

`p h` 選択された日付を、ヘブライ暦の日付 (Hebrew date) で表示します (`calendar-hebrew-print-date`)。

`p i` 選択された日付を、イスラム暦の日付 (Islamic date) で表示します (`calendar-islamic-print-date`)。

`p f` 選択された日付を、フランス革命暦の日付 (French Revolutionary date) で表示します (`calendar-french-print-date`)。

`p b` 選択された日付を、Bah^^c3^^a1'^^c3^^ad date で表示します (`calendar-bahai-print-date`)。

`p C` 選択された日付を、旧暦の日付 (Chinese date) で表示します (`calendar-chinese-print-date`)。

`p k` 選択された日付を、Coptic date で表示します (`calendar-coptic-print-date`)。

`p e` 選択された日付を、Ethiopic date で表示します (`calendar-ethiopic-print-date`)。

`p p` 選択された日付を、ペルシャ暦の日付 (Persian date) で表示します (`calendar-persian-print-date`)。

`p m` 選択された日付を、マヤ暦の日付 (Mayan date) で表示します (`calendar-mayan-print-date`)。

変換したい日付にポイントを移動して、上記のテーブルから `p`で始まる適切なコマンドをタイプします。プレフィクスの `p`は、“print” のニーモニックです。これは、Emacs が」変換した日付を、エコーエリアに “プリント” するからです。`p o`は日付を、Emacs が知るすべての形式で表示します。`mouse-3`でクリックして表示されるメニューから、`Other calendars`を選択することもできます。これは、日付を Emacs が知るすべての形式でメニュー形式で表示します (メニュー形式で表示されるさまざまな形式の日付を選択しても、何もおこりません。この場合、メニューは表示のためだけに使用されます)。

28.9.3 他のカレンダーからの変換

サポートされている他のカレンダーで日付を指定して、その日付に移動することができます。このセクションでは、マヤ暦以外のカレンダーを使用してこれを行なうコマンドを説明します。マヤ暦については、以降のセクションを参照してください。

`g c` ISO 商用暦 (ISO commercial calendar) で指定された日付に移動します (`calendar-iso-goto-date`)。

`g w` ISO 商用暦で指定された週に移動します (`calendar-iso-goto-week`)。

`g j` ユリウス暦 (Julian calendar) で指定された日付に移動します (`calendar-julian-goto-date`)。

`g a` ユリウス暦にもとづく天文日で指定された日付に移動します (`calendar-astro-goto-day-number`)。

`g b` Bah^^c3^^a1’^^c3^^ad calendar で指定された日付に移動します (`calendar-bahai-goto-date`)。

`g h` ヘブライ暦 (Hebrew calendar) で指定された日付に移動します (`calendar-hebrew-goto-date`)。

`g i` イスラム暦 (Islamic calendar) で指定された日付に移動します (`calendar-islamic-goto-date`)。

`g f` フランス革命暦 (French Revolutionary calendar) で指定された日付に移動します (`calendar-french-goto-date`)。

`g C` 旧暦 (Chinese calendar) で指定された日付に移動します (`calendar-chinese-goto-date`)。

`g p` ペルシャ暦 (Persian calendar) で指定された日付に移動します (`calendar-persian-goto-date`)。

`g k` コプト暦 (Coptic calendar) で指定された日付に移動します (`calendar-coptic-goto-date`)。

`g e` エチオピア暦 (Ethiopic calendar) で指定された日付に移動します (`calendar-ethiopic-goto-date`)。

これらのコマンドは、他のカレンダーの日付の入力を求め、その日付と等価なグレゴリオ暦 (Ethiopic calendar) の日付に移動して、他のカレンダーの日付をエコーエリアに表示します。Emacs は月の名前を入力するとき、常に強い補完 (Section 5.4.3 [Completion Exit], page 33 を参照してください) を使うので、ヘブライ、イスラム、フランスの名前のスペルを心配する必要はありません。

ヘブライ暦に関する一般的な問題として、*yahrzeit* と呼ばれる死亡日にたいする記念日の計算があります。Emacs のカレンダーには、そのような計算の機能が含まれています。カレンダーにいるときは、コマンド `M-x calendar-hebrew-list-yahrzeits`により、まず年の範囲の入力を求め、ポイント位置の日付により、それらの年の yahrzeit の日付のリストを表示します。カレンダーにいないときは、このコマンドは最初に死亡日、次に年の範囲の入力を求め、yahrzeit の日付の一覧を表示します。

28.10 ダイアリー

Emacs のダイアリーはカレンダーと共に、日常のアポイントメントやその他のイベントを管理します。ダイアリー機能を使うには、最初にイベントとその日付を含むダイアリーファイル (diary file) を作成しなければなりません。Emacs は今日、近い将来、指定された日付のイベントを自動的にピックアップして表示します。

ダイアリーの作成を手作業で行なおうと考えているかもしれませんが、Emacs はダイアリーのエントリーを閲覧、追加、変更するいくつかのコマンドを提供します。

28.10.1 ダイアリーファイル

ダイアリーファイル *(diary file)* とは、特定の日付に関連付けられたイベントを記録するファイルです。ダイアリーファイルの名前は、変数 `diary-file`により指定されます。デフォルトは`~/.emacs.d/diary`ですが、古いバージョンの Emacs との互換性のため、`~/diary`が存在する場合は、それを使用します。

ダイアリーファイルの各エントリーは、1 つ以上の行からなる 1 つのイベントを記述します。エントリーは、常に左端の日付指定で開始されます。エントリーの残りの部分は、イベントを説明するテキストです。エントリーが複数行の場合、2 行目以降はそれが前のエントリーの継続行であることを示すために、空白文字で開始しなければなりません。有効な日付で開始されておらず、前のエントリーの継続行でもない行は無視されます。以下は例です:

```
12/22/2015  Twentieth wedding anniversary!
10/22       Ruth's birthday.
* 21, *:    Payday
Tuesday--weekly meeting with grad students at 10am
         Supowit, Shen, Bitner, and Kapoor to attend.
1/13/89     Friday the thirteenth!!
thu 4pm     squash game with Lloyd.
mar 16      Dad's birthday
April 15, 2016 Income tax due.
* 15        time cards due.
```

この例では、ほとんどのエントリーにたいして、余分なスペースでイベント説明を位置揃えしています。このようなフォーマットは、純粋に好みの問題です。

ダイアリーエントリーの最初の行が、日付または曜日名 (後にブランクも句読点もない) だけの形式を使うこともできます。たとえば:

```
02/11/2012
```

```
Bill B. visits Princeton today
2pm Cognitive Studies Committee meeting
2:30-5:30 Liz at Lawrenceville
4:00pm Dentist appt
7:30pm Dinner at George's
8:00-10:00pm concert
```

このエントリーは、シンプルなダイアリー表示を使ったときは、異なる表示になります (Section “Diary Display” in *Specialized Emacs Features* を参照してください)。シンプルなダイアリー表示は最初の日付行を省略して、継続行だけが表示されます。このスタイルのエントリーは、1 日分だけのエントリーを表示するときはうまく表示されますが、複数日のエントリーの場合は混乱するかもしれません。

28.10.2 ダイアリーの表示

1 度ダイアリーファイルを作成すると、カレンダーを使ってそれを閲覧できます。Calendar モードの外で、今日のイベントを閲覧することもできます。以下は Calendar バッファーで参照されるキーバインドです。

`mouse-3` `Diary`
:

`d`
: 選択された日付の、すべてのダイアリーエントリーを表示します (`diary-view-entries`)。

`s`
: ダイアリーファイル全体を表示します (`diary-show-all-entries`)。

`m`
: 表示されている日付で、ダイアリーエントリーをもつものをすべてマークします (`diary-mark-entries`)。

`u`
: カレンダーウィンドウのマークを外します (`calendar-unmark`)。

`M-x diary-print-entries`
: 表示されているダイアリーのハードコピーを印刷します。

`M-x diary`
: 今日の日付の、すべてのダイアリーエントリーを表示します。

`M-x diary-mail-entries`
: 近づきつつあるダイアリーエントリーのリマインダーとして、あなた自身にメールします。

`d`でダイアリーエントリーを表示すると、カレンダーで選択された日付のダイアリーエントリーが別のバッファーに表示されます。新しいバッファーのモードラインには、ダイアリーエントリーの日付が表示されます。休日はバッファーとモードラインの両方で表示され、それは選択した表示方法に依存します (Section “Diary Display” in *Specialized Emacs Features* を参照してください)。`d`に数引数を指定した場合、選択した日付から指定した日数までのすべてのダイアリーエントリーを表示します。したがって、`2 d`は選択された日と、その次の日のすべてのエントリーを表示します。

ある日付にたいするダイアリーエントリーを表示する別の方法は、日付を`mouse-3`でクリックして、表示されるメニューから`Diary entries`を選択する方法です。変数`calendar-view-diary-initially-flag`が非`nil`の場合、カレンダーの作成により、その日のダイアリーエントリーがリストされます (その日が表示されている場合)。

より広い視点でダイアリーが記載されている日を知るには、`m`コマンドを使用します。これはダイアリーエントリーをもつ日付を、異なるフェイスでマークします。Section “Calendar Customizing” in *Specialized Emacs Features* を参照してください。

このコマンドは現在表示されている月と、スクロールすることにより表示される月の両方に適用されます。マークを消すには`u`をタイプします。これにより休日のマークもオフになります (Section 28.6

[Holidays], page 405 を参照してください)。変数 `calendar-mark-diary-entries-flag`が非`nil`の場合、カレンダーの作成または更新により、ダイアリーの日付が自動的にマークされます。

カレンダーで、特定のダイアリーエントリーのマークを抑止するには、エントリーの開始、日付の前に、`diary-nonmarking-symbol`で指定された文字列 (デフォルトは '`&`') を挿入します。これはエントリーをダイアリーバッファーで表示する場合は、影響ありません。カレンダーウィンドウで日付をマークするときだけ影響があります。マークされないエントリーは、マークの数が多くなりすぎるような、一般的なエントリーにたいして有用かもしれません。

限られたエントリーだけでなく、ダイアリーファイル全体を参照するには、`s`コマンドを使用します。

コマンド `M-x diary`は、カレンダーの表示とは独立に、その日のダイアリーエントリーと、オプションでその後の何日かのダイアリーエントリーも同様に表示します。何日分を含めるかは変数`diary-number-of-entries`で指定してください。Section "Diary Customizing" in *Specialized Emacs Features* を参照してください。

`.emacs`ファイルに `(diary)`を記述した場合、Emacs を開始したときに、その日のダイアリーエントリーのウィンドウを自動的に表示します。

ダイアリーのイベント通知を、メールで受けとるのを好む人もいます。そのようなメールをあなた自身に送るには、コマンド `M-x diary-mail-entries`を使用します。プレフィクス引数は、(今日から開始して) 何日分をチェックするかを指定します。プレフィクス引数を指定しない場合、変数`diary-mail-days`により、何日分をチェックするかが指定されます。

28.10.3 日付のフォーマット

以下は、異なる方法でフォーマットされた日付を説明するために、ダイアリーエントリーの例をいくつか示したものです。この例では、日付はすべてアメリカ形式 (月、日、年) ですが、Calendar モードは、オプションでヨーロッパ形式 (日、月、年) と、ISO 形式 (年、月、日) をサポートします。

```
4/20/12  Switch-over to new tabulation system
apr. 25  Start tabulating annual results
4/30  Results for April are due
*/25  Monthly cycle finishes
Friday  Don't leave without backing up files
```

最初のエントリーは、2012 年 4 月 20 日に 1 度だけ表示されます。2 番目と 3 番目のエントリーは毎年、指定した日に表示され、4 番目のエントリーは月にワイルドカード (アスタリスク) を使っているので、毎月 25 日に表示されます。最後のエントリーは毎週、金曜日に表示されます。

'*month*`/`*day*'や'*month*`/`*day*`/`*year*'のように、日付を表すのに数字だけを使うこともできます。この場合、その後に続く文字は数字以外の文字でなければなりません。日付の *month*と *day*は、1 桁または 2 桁の数字です。オプションの *year*も数字で、省略形として最後の 2 桁を使うこともできます。つまり、'`11/12/2012`'または'`11/12/12`'を使うことができます。

日付は、'*monthname day*'または'*monthname day*`,` *year*'という形式をもつこともできます。月の名前のスペルには完全形、または省略形 (最後のピリオドはあってもなくてもよい) を指定できます。月または曜日の優先される省略形は、変数 `calendar-abbrev-length`、`calendar-month-abbrev-array`、`calendar-day-abbrev-array`を使ってセットできます。デフォルトは、名前の最初の 3 文字を省略形として使用します。大文字小文字に違いはありません。

日付はジェネリック *(generic)* — つまり指定されていない部分があっても構いません。この場合、エントリーは日付に指定された部分にマッチする、すべての日付に適用されます。日付が年を含まない場合、これはジェネリックで、任意の年に適用されます。かわりに *month*、*day*、*year*に '`*`'を使

用することもできます。これは任意の月、任意の日、任意の年にマッチします。したがって、日付は‘3/*/*’のダイアリーエントリーは任意の年の 3 月の任意の日にマッチします。‘march *’も同じです。

ヨーロッパ形式 (月の前に日を記述する)、または ISO 形式 (年、月、日の順で記述する) で日付を記述したい場合は、カレンダーで M-x calendar-set-date-styleとタイプするか、変数calendar-date-styleをカスタマイズします。これはダイアリーの日付がどのように解釈されるか、日付の表示、コマンドが与えられた引数に要求する順序に影響を与えます。

週のある曜日に適用されるジェネリックな日付として、曜日名を使うことができます。曜日名は完全なスペルを記述するか、上述した省略形を使用できます。大文字小文字に違いはありません。

28.10.4 ダイアリーに追加するコマンド

カレンダーでは、ダイアリーのエントリーを作成するコマンドがいくつかあります。以下は基本的なコマンドの一覧です。より複雑なコマンドは次のセクションで説明します ((Section 28.10.5 [Special Diary Entries], page 414 を参照してください))。エントリーにはグレゴリオ暦以外も使用できます。Section “Non-Gregorian Diary” in *Specialized Emacs Features* を参照してください。

i d 選択された日付のダイアリーエントリーを追加します (diary-insert-entry)。

i w 選択された曜日のダイアリーエントリーを、各週に追加します (diary-insert-weekly-entry)。

i m 選択された日のダイアリーエントリーを、各月に追加します (diary-insert-monthly-entry)。

i y 選択された日のダイアリーエントリーを、各年に追加します (diary-insert-yearly-entry)。

カレンダーウィンドウで日付を選択して、i dコマンドをタイプすることにより、特定の日付にたいするダイアリーエントリーを作成することができます。このコマンドは、ダイアリーファイルの最後の部分を別のウィンドウに表示して、その日付を追加します。その後、ダイアリーエントリーの残りの部分をタイプできます。

毎週、特定の曜日に適用されるダイアリーエントリーを作成したいときは、その曜日 (同じ曜日ならどの日でも構いません) を選択して、i wとタイプします。これは曜日をジェネリックな日付として挿入します。そのあとダイアリーエントリーの残りの部分をタイプできます。月ごとのダイアリーエントリーも同じやり方で作成できます。まず月のある日付を選択して、i mコマンドを使い、その後エントリーの残りの部分をタイプします。同様に i yコマンドで、年ごとのダイアリーエントリーを挿入できます。

上記のすべてのコマンドは、デフォルトでマークされるダイアリーエントリーを作成します。マークされないダイアリーエントリーを作成するには、コマンドにプレフィクス引数を与えます。たとえば、C-u i wは、マークされない週次のダイアリーエントリーを作成します。

ダイアリーファイルを変更したときは、Emacs を終了する前にそのファイルを保存してください。上記の挿入コマンドを使用した後でダイアリーファイルを保存することにより、それが適切な場合は、カレンダーウィンドウのダイアリーのマークを自動的に更新します。随時に更新させるには、コマンド calendar-redrawを使うことができます。

28.10.5 特別なダイアリーエントリー

カレンダーの日付にもとづくエントリーに加え、ダイアリーファイルは記念日のような定期的なイベントにたいする、*sexp* エントリー *(sexp entries: S* 式エントリー*)* を含むことができます。これらのエントリーは、Emacs がダイアリーファイルをスキャンすることにより評価される、Lisp 式 (sexp)

にもとづきます。日付のかわりに、sexp エントリーは '%%'と、その後ろに続く Lisp 式を含んでおり、Lisp 式はカッコで始まりカッコで終わらなければなりません。Lisp 式は、エントリーが適用される日付を決定します。

Calendar モードは、一般的に使用される特定の sexp エントリーを挿入するためのコマンドを提供します:

`i a` 選択された日付にたいして、記念日ダイアリーエントリー (anniversary diary entry) を追加します。

`i b` カレントリージョンにたいして、ブロックダイアリーエントリー (block diary entry) を追加します。

`i c` その日付に開始される、周期的ダイアリーエントリー (cyclic diary entry) を追加します。

特定の日付の記念日に適用されるダイアリーエントリーを作成したい場合は、ポイントをその日付に移動して `i a`コマンドを使用します。これはダイアリーファイルの最後の部分を別のウィンドウに表示して、記念日の記述を追加します。その後ダイアリーエントリーの残りの部分をタイプできます。エントリーは以下のようになります:

```
%%(diary-anniversary 10 31 1988) Arthur's birthday
```

このエントリーは、1988 年以降の任意の年の 10 月 31 日に適用されます。'10 31 1988'は日付を指定します (ヨーロッパ形式または ISO 形式を使用している場合、入力順は月、日、年とは異なります)。この式が開始年を要求する理由は、ダイアリーの上級機能が、経過年数を計算できるようにするためです。

ブロックダイアリーエントリーは、特定の連続する日付範囲に適用されます。以下は 2012 年 6 月 24 日から 2012 年 7 月 10 日までの、すべての日付に適用されるブロックダイアリーエントリーです:

```
%%(diary-block 6 24 2012 7 10 2012) Vacation
```

'6 24 2012'は開始日付を示し、'7 10 2012'は終了日付を示します (繰り返しになりますが、ヨーロッパ形式または ISO カレンダー形式を使用している場合、月、日、年の順は異なります)。

ブロックエントリーを入力するには、開始と終了の範囲となる 2 つの日付にポイントとマークを配し、`i b`とタイプします。このコマンドは、ダイアリーファイルの最後の部分を別のウィンドウに表示して、ブロックの記述を挿入します。その後で、ダイアリーエントリーをタイプすることができます。

周期的 *(cyclic)* ダイアリーエントリーは、ある固定された日数の期間繰り替えされるエントリーです。これを作成するには、開始日を選択して `i c`コマンドを使用します。コマンドは期間の長さの入力を求め、それから以下のようなエントリーを挿入します:

```
%%(diary-cyclic 50 3 1 2012) Renew medication
```

このエントリーは、2012 年 3 月 1 日以降の 50 日間適用されます。'3 1 2012'は開始日の指定です (ヨーロッパ形式または ISO カレンダー形式を使用している場合、月、日、年の入力順は異なります)。

これら 3 つのコマンドはすべて、ダイアリーエントリーをマークします。マークしないダイアリーエントリーを挿入するには、プレフィクス引数を与えます。たとえば `C-u i a`は、マークされない記念日ダイアリーエントリーを作成します。

カレンダーで sexp ダイアリーエントリーを作成すると、カレンダーウィンドウで表示されているすべての日付にたいして個別にチェックしなければならないので、時間がかかるかもしれません。そのため、可能なら sexp ダイアリーエントリーを、('`&`'を使って) マークされないようにするのがよいでしょう。

その他の複雑な sexp エントリーとして、浮動 *(floating)* ダイアリーエントリーがあります。これは日、週、年のオフセットで指定される、定期的に発生するイベントを指定するためのものです。こ

れは cronにより解釈される crontab エントリーに類似しています。以下はマークされない、浮動ダイアリーエントリーで、11 月の第 4 木曜日に適用されます。

```
&%%(diary-float 11 4 4) American Thanksgiving
```

11 は 11 月 (11 番目の月) を指定し、4 は木曜日 (週の第 4 日。日曜日は 0)、2 番目の 4 は第 4 週 (1 は第 1 週、2 は第 2 週、−2 は最終週から 2 番目の週) を指定します。月は 1 つの月、または月のリストを使用できます。したがって上記の 11 を‘'(1 2 3)’に変更すると、このエントリーは 1 月、2 月、3 月の第 4 木曜日に適用されることになります。月が tの場合、そのエントリーは年の各月に適用されます。

```
%%(diary-offset '(diary-float t 3 4) 2) Monthly committee meeting
```

これは毎月第 3 木曜日の後の土曜日にエントリーを適用します。2 は S 式'(diary-float t 3 4)が tが評価された後の日数を指定します。これはたとえばあなたの組織で第 3 木曜日の月例会議の 2 日後に委員会会合が開かれる場合や、日付に違いが生じるような別のタイムゾーンでのオンライン会議に出席するような場合に有用です。

標準的な sexp ダイアリーエントリーは、フェイス名またはカレンダーをマークするときに使用する 1 文字の文字列を指定する、オプションのパラメーターを受け取ります。一般的には、sexp ダイアリーエントリーは、それらが適用されるときを決定するために、任意の計算を処理することができます。Section “Sexp Diary Entries” in *Specialized Emacs Features* を参照してください。

28.10.6 アポイントメント

アポイントメント (appointment: 約束、予約) にたいするダイアリーエントリーがある場合、そのダイアリーエントリーが認識可能な日時で開始されていれば、Emacs は保留されたアポイントメントがあることを警告することができます。Emacs は、変数 appt-display-formatで選択されたフォーマットでメッセージを表示して、アポイントメントにたいする注意を喚起します。appt-audibleの値が非 nilの場合、警告には音によるリマインダーも含まれます。加えて、appt-display-mode-lineが非 nilの場合、Emacs はアポイントメントまで何分あるかを、モードラインに表示します。

appt-display-formatの値が windowの場合、変数 appt-display-durationがリマインダーウィンドウを表示する長さを制御します。変数 appt-disp-window-functionおよび appt-delete-window-functionは、ウィンドウを生成または破棄する関数の名前を与えます。

アポイントメントの通知を有効にするには、M-x appt-activateとタイプします。正の引数は通知を有効に、負の引数は通知を無効に、引数を指定しない場合は通知のオンとオフを切り替えます。通知を有効にすることにより、ダイアリーファイルから見つかった、認識可能な日時をもつすべてのダイアリーエントリーにより、今日のアポイントメントリストをセットアップし、それらのアポイントメントの直前に注意を促します。

たとえば、ダイアリーファイルに以下のような行が含まれているとします:

```
Monday
  9:30am Coffee break
 12:00pm Lunch
```

月曜日には、9:20am 頃にコーヒーブレーク、11:50am 頃にランチの注意が促されます。変数 appt-message-warning-timeには、前もって何分前 (デフォルトは 12) に警告するかを指定します。これはデフォルトの警告タイムです。appt-warning-time-regexpに部分マッチングを追加することにより、各アポイントメントに異なる警告タイムを指定できます (詳細は、この変数のドキュメントを参照してください)。

時刻は am/pm スタイル (‘12:00am’は真夜中で、‘12:00pm’は正午)、またはヨーロッパ/軍隊の 24 時間制で記述できます。どちらを使うか一貫性がある必要はありません。ダイアリーファイルで、

この 2 つのスタイルを混交させることができます。時刻が認識されるためは、それがダイアリーエントリーの最初に記述されていなければなりません。

Emacs は、真夜中直後にダイアリーファイルから自動的にアポイントメントリストを更新します。アポイントメント通知を再度有効にすることにより、随時に更新させることができます。`appt-display-diary`を `nil`にセットしていなければ、これらのアクションはその日のダイアリーバッファーにも表示されます。アポイントメントリストは、ダイアリーファイル (またはそれをインクルードするファイル。Section “Fancy Diary Display” in *Specialized Emacs Features* を参照してください) に保存されます。Org モードを使用していて、アポイントメントを Org アジェンダファイルに保持したい場合は、`org-agenda-to-appt`コマンドを使用して、アポイントメントをアジェンダファイルのリストに追加できます。このコマンドについての詳細は、Section “Weekly/daily agenda” in *The Org Manual* を参照してください。

アラーム時計のように、アポイントメント通知機能を使うこともできます。コマンド `M-x appt-add` は、ダイアリーファイルに影響を与えずに、アポイントメントリストにエントリーを追加します。アポイントメントリストからエントリーを削除するには、`M-x appt-delete`を使います。

28.10.7 ダイアリーエントリーのインポートとエクスポート

Emacs のダイアリーファイルと、他のさまざまなフォーマットの間で、ダイアリーエントリーを変換できます。

Outlook が生成したアポイントメントメッセージから、ダイアリーエントリーをインポートできます。そのようなメッセージを Rmail や Gnus で閲覧しているときは、エントリーをインポートするために `M-x diary-from-outlook`を実行します。変数 `diary-outlook-formats`をカスタマイズすることにより、このコマンドに追加のアポイントメッセージ形式を認識させることができます。他のメールクライアントは、`diary-from-outlook-function`を適切な値にセットできます。

icalendar パッケージにより、Emacs ダイアリーファイルと、*RFC 2445 — Internet Calendaring and Scheduling Core Object Specification (iCalendar)* で定義される iCalendar との間で、データを変換することができます (初期の vCalendar 形式も同様)。

コマンド `icalendar-import-buffer`は、カレントバッファーから iCalendar データを抽出して、それをダイアリーファイルに追加します。この関数は、iCalendar データを自動的に抽出するためにも適しています。たとえば Rmail メールクライアントは、以下を使用できます:

```
(add-hook 'rmail-show-message-hook 'icalendar-import-buffer)
```

コマンド `icalendar-import-file`は、iCalendar ファイルをインポートして、その結果を Emacs ダイアリーファイルに追加します。たとえば:

```
(icalendar-import-file "/here/is/calendar.ics"
                       "/there/goes/ical-diary")
```

もし違うファイルが存在する場合は、インポートファイルの内容を追加するために、`#include`ディレクティブを使うことができます。Section “Fancy Diary Display” in *Specialized Emacs Features* を参照してください。

`icalendar-export-file`を使って、Emacs ダイアリーファイル全体を iCalendar 形式にエクスポートできます。ダイアリーファイルの一部をエクスポートするには、関連する領域をマークして、`icalendar-export-region`を呼び出します。どちらの場合も、Emacs は結果をターゲットファイルに追加します。

28.11 サマータイム

Emacs は、標準時とサマータイムの違いを理解します。日の出、日の入り、夏至、冬至、春分、秋分、月の位相では、時刻を補正しています。サマータイムのルールは、場所によりさまざまで、歴史的にも異なります。処理を正しく行なうには、Emacs が使用するルールを知る必要があります。

どこにいるかにより適用されるルールを、追跡するオペレーティングシステムもあります。そのようなシステムでは、Emacs はそのシステムから必要な情報を取得します。これらの情報の一部またはすべてが欠落している場合、Emacs は現在マサチューセッツのケンブリッジで使用されているルールで、そのギャップを埋めます。その結果としてのルールが望むものでない場合、変数 `calendar-daylight-savings-starts`および `calendar-daylight-savings-ends`をセットすることにより、Emacs に使用するルールを指定できます。

これらの値は、変数 `year`を参照する Lisp 式で、サマータイムが開始または終了される、`(`*`month day year`*`)`という形式のリストによる、グレゴリオ暦の日付として評価されます。その地域がサマータイムを使用しない場合、値は `nil`であるべきです。

Emacs は、サマータイムの開始を決定するためにこれらの式を使い、休日リスト、太陽および月に関する時刻の補正を行ないます。

マサチューセッツのケンブリッジにたいする値は、以下のとおりです:

```
(calendar-nth-named-day 2 0 3 year)
(calendar-nth-named-day 1 0 11 year)
```

つまり、`year`で指定される年の、3 番目の月 (3 月) の、第 2 週の 0 番目の曜日 (日曜日) と、、その年の 11 番目の月 (11 月) の、第 1 週の日曜日です。サマータイムが 10 月 1 日から開始されるように変更する場合、`calendar-daylight-savings-starts`を以下のように変更します:

```
(list 10 1 year)
```

その地域でサマータイムがない、またはすべての時刻を標準時にしたい場合は、`calendar-daylight-savings-starts`と `calendar-daylight-savings-ends`を、`nil`にセットします。

変数 `calendar-daylight-time-offset`は、サマータイムと標準時の差を、分で指定します。マサチューセッツのケンブリッジでは 60 です。

最後に、2 つの変数 `calendar-daylight-savings-starts-time`と `calendar-daylight-savings-ends-time`は、サマータイムの開始と終了の遷移時に、地方時の真夜中から何分ずれるかを指定します。マサチューセッツのケンブリッジでは、変数の値は両方とも 120 です。

28.12 時間間隔の加算

timeclock パッケージは、時間間隔を加算していくので、(たとえば) 特定のプロジェクトにどれだけ時間を費やしているか、追跡することができます (より上級の代替手段は Org モードの時間計測機能の使用である。Section “Clocking Work Time” in *The Org Manual* を参照)。

プロジェクトの作業を開始したとき `M-x timeclock-in`コマンドを使用し、作業を終えたら `M-x timeclock-out`コマンドを使用します。これを行なうたびに、プロジェクトの記録に時間間隔を追加します。違うプロジェクトの作業に変更する場合は、`M-x timeclock-change`を使用します。

いくつかの時間間隔をからデータを収集したら、`M-x timeclock-workday-remaining`を使って、その作業にたいして今日の残り時間を見ることができ (1 日の作業時間の平均は通常 8 時間とみなします)、`M-x timeclock-when-to-leave`で作業を終了する時間を計算します。

Emacs にたいして、モードラインに作業日の残り時間を表示させたいときは、変数 `timeclock-mode-line-display`を `t`にセットするか、コマンド `M-x timeclock-mode-line-display`を呼び出します。

Emacs のカレントセッションを終了することは、そのプロジェクトの作業を終えることを意味するか不明なので、デフォルトでは Emacs は確認を求めます。しかし変数 `timeclock-ask-before-exiting`の値を `nil`にカスタマイズすることにより、確認を無視できます。その場合、明示的に `M-x timeclock-out`か `M-x timeclock-change`を呼び出した場合だけ、カレントの時間間隔が終了したと Emacs に告げます。

timeclock 関数は、`~/.emacs.d/timelog`と呼ばれるファイルに、データを集積することにより機能します。変数 `timeclock-file`をカスタマイズすることにより、このファイルに違う名前を指定できます。手作業で timeclock を編集したり、timeclock のカスタマイズ可能な変数の値を変更したときは、コマンド `M-x timeclock-reread-log`を実行するべきです。

29 メールの送信

Emacs から電子メールのメッセージを送信するには、C-x mとタイプします。これはメッセージのテキストとヘッダーを編集できる、*unsent mail*という名前のバッファーに切り替えます。編集が終了したら C-c C-sまたは C-c C-cとタイプして、それを送信します。

C-x m　　メールの作成を開始します (compose-mail)。

C-x 4 m　　同様ですが、他のウィンドウで行ないます (compose-mail-other-window)。

C-x 5 m　　同様ですが、新しいフレームで行ないます (compose-mail-other-frame)。

C-c C-s　　メールバッファーでは、メッセージを送信します (message-send)。

C-c C-c　　メールバッファーでは、メッセージを送信して、そのバッファーを隠し (bury) ます (message-send-and-exit)。

メールバッファーは通常の Emacs バッファーなので、メールの作成中に他のバッファーに切り替えることができます。現在のメールを終了する前に他のメールを送信したい場合は、再度 C-x mとタイプして新しいメールバッファーを開きます、このバッファーは異なる数字が後ろについた名前をもちます (Section 16.3 [Misc Buffer], page 179 を参照)。(これはメール作成にデフォルトの Message モードを使用する場合のみ機能する。Section 29.4 [Mail Commands], page 423 を参照のこと。) 編集中の未送信メッセージの作成を継続したいと判っている場合には、C-u C-x m のようにコマンドにプレフィクス引数を与えれば、Emacs は使用していた最後のメールバッファーに切り替えて、終了する前にメッセージの編集ができるように計らいます。

コマンド C-x 4 m (compose-mail-other-window) は、C-x mと同じことを行ないますが、これはメールバッファーを別のウィンドウに表示します。コマンド C-x 5 m (compose-mail-other-frame) は、新しいフレームでこれを行ないます。

C-c C-cまたは C-c C-sとタイプしてメールを送信するとき、Emacs はどのようにしてメールを送信するべきか — SMTP を通じて直接送信するか、またはほかの方法を使うか — を尋ねます。詳細は、Section 29.4.1 [Mail Sending], page 423 を参照してください。

29.1 メールバッファーのフォーマット

以下はメールバッファーの内容の例です (訳注: 出典はコナン・ザ・グレートらしいです):

```
To: subotai@example.org
CC: mongol.soldier@example.net, rms@gnu.org
Subject: Re: What is best in life?
From: conan@example.org
--text follows this line--
To crush your enemies, see them driven before you, and to
hear the lamentation of their women.
```

メールバッファーのトップは、一連のヘッダーフィールド *(header fields)* です。これは電子メールの受取人、subject(題目) などに関する情報を指定するのに使用されます。上記のバッファー例には‘To’、‘CC’、‘Subject’、‘From’のヘッダーフィールドが含まれています。いくつかのヘッダーフィールドは、それが適切なときは、事前に自動的に初期化されます。

‘--text follows this line--’という行は、ヘッダーフィールドとメッセージの *body*(本文、または *text*) を分割します。この行の上にあるすべては、ヘッダーの一部として扱われます。下にあるすべては、body として扱われます。区切り行自体は、実際に送信されるメッセージには含まれません。

通常の編集コマンドで、ヘッダーフィールドの挿入と編集ができます。ヘッダーフィールドを編集するための特定のコマンドについては、Section 29.4.2 [Header Editing], page 424 を参照してください。‘Date’や‘Message-Id’のような特定のヘッダーは、通常はメールバッファーでは省略され、メッセージを送信するとき自動的に作成されます。

29.2 メールヘッダーフィールド

メールバッファーのヘッダーフィールドは、行の最初がフィールド名で始まります。フィールド名はコロンで終端されます。フィールド名で大文字小文字は区別されません。コロンとオプションの空白文字の後に、フィールドの内容を記述します。

好きな名前のヘッダーフィールドを使用できますが、多くの人は通常、一般に認められた意味をもつは、標準的なフィールド名だけを使用します。

ヘッダーフィールド‘From’は、電子メールを送信した人 (あとえばあなた) を識別します。これは有効なメールアドレスである必要があります。なぜなら、通常はそのアドレスに返信されるからです。このヘッダーフィールドのデフォルトの内容は、変数 `user-full-name`(あなたのフルネームを指定します) と `user-mail-address`(あなたの電子メールアドレス) から計算されます。いくつかのオペレーティングシステムでは、Emacs はこの 2 つの変数を環境変数から初期化します (Section C.4.1 [General Variables], page 580 を参照してください)。この情報が利用不可能か間違っている場合、変数を自分でカスタマイズする必要があります (Section 33.1 [Easy Customization], page 499 を参照してください)。

以下は‘From’以外の、一般的に使用されるフィールドの表です:

‘To’ メッセージを送信するメールアドレスです。複数のアドレスを記述するには、それらを区切るのにカンマを使用します。

‘Subject’ メッセージの subject です。

‘CC’ メッセージを送信する追加のメールアドレスです。これは‘To’と似ていますが、受取人はそのメッセージが自分宛だと思うべきではありません。

‘BCC’ 実際に送信するメッセージのヘッダーには現れない、追加のメッセージ送信先のメールアドレスです。‘BCC’は *blind carbon copies* が由来です。

‘FCC’ 送信されたメッセージのコピーが追加されるべき、ファイルの名前です。ファイルが Babyl 形式 (Emacs23 以前の Rmail で使用されていました) の場合、Emacs は Babyl 形式で書き込み、それ以外は mbox 形式で書き込みます。Rmail バッファーがそのファイルを visit している場合、Emacs はそれに合わせて更新します。複数のファイルを指定するには、複数の‘FCC’フィールドを使用して、各フィールドに 1 つのファイル名を記述します。

‘Reply-To’
‘From’のかわりとなる、返信が送信されるべきアドレスです。これは何らかの理由により、‘From’のアドレスが返信を受け取れないときに使用します。

‘Mail-Reply-To’
このフィールドは‘Reply-To’より優先されます。メーリングリストの中には、‘Reply-To’を独自の目的 (問題の多い、何らかの方法) のために使用しているものがあるため、これが使用されます。

‘Mail-Followup-To’
: follow-up メッセージのための、デフォルトの受取人のために使用する 1 つ以上のアドレスです。これは通常、登録しているメーリングリストからメッセージを受信するとき、余分なコピーを送らせないようにしたいときに使用します。

‘In-Reply-To’
: 返信するメッセージのための識別子です。ほとんどのメールリーダーは、関連するメッセージをまとめてグループ化するためにこの情報を使用します。このヘッダーは通常、Emacs に組み込まれた任意のメールプログラムでメッセージに返信するとき、自動的に充填されます。

‘References’
: 以前の関連するメッセージのための識別子です。‘In-reply-To’と同じように、これは通常、自動的に充填されます。

‘To’、‘CC’、‘BCC’のフィールドは何回記述してもよく、また各フィールドにはカンマで括って複数のアドレスを含めることができます。この方法により、メッセージを送信する複数の場所を指定できます。これらのフィールドには継続行も使用できます。フィールドの開始行に続く、空白文字で始まる 1 行以上の行は、そのフィールドの一部とみなされます。以下は継続行を使用した ‘To’フィールドの例です:

```
To: foo@example.net, this@example.net,
  bob@example.com
```

変数 mail-default-headersに文字列をセットすることにより、特定のデフォルトヘッダーを挿入するよう、Emacs に指示できます。これにより C-x mでメッセージヘッダーにその文字列が挿入されます。たとえば以下は、各メッセージにヘッダー ‘Reply-To’と ‘FCC’を追加する例です:

```
(setq mail-default-headers
      "Reply-To: foo@example.com\nFCC: ~/Mail/sent")
```

デフォルトのヘッダーフィールドが、特定のメッセージにたいして適切でない場合、メッセージを送信する前にそれらを編集する必要があります。

29.3 メールエイリアス

メールエイリアス *(mail aliases)* を定義することができます。これは 1 つ以上のメールアドレスを意味する短い呼び名です。デフォルトでは、メールエイリアスはファイル~/.mailrcで定義されます。変数 mail-personal-alias-fileをセットすることにより、異なるファイル名を指定できます。

~/.mailrcでエイリアスを定義するには、以下のように記述します:

```
alias nick fulladdresses
```

これは *nick*が *fulladdresses*に展開されるという意味で、*fulladdresses*は単一のアドレス、またはスペースで区切られた複数のアドレスです。たとえば maingnuが、gnu@gnu.orgとあなた自身のローカルのアドレスを意味するよう定義するには、以下の行を記述します:

```
alias maingnu gnu@gnu.org local-gnu
```

アドレスにスペースが含まれる場合は、以下のようにダブルクォートでアドレス全体をクォートします:

```
alias jsmith "John Q. Smith <none@example.com>"
```

その人の名前の部分などのような、アドレスの特定の部分をダブルクォートで括る必要はないことに注意してください。Emacs は必要に応じてそれらを挿入します。たとえば上記のアドレスは ‘"John Q. Smith" <none@example.com>’のように挿入されます。

Emacs は、`~/.mailrc`での include コマンドも識別します。これらは以下のようなものです:

```
source filename
```

ファイル`~/.mailrc`は、Emacs だけのものではありません。他の多くのメールを閲覧するプログラムが、メールアドレスのためにこれを使用し、他にもさまざまなコマンドが含まれます。しかし Emacs は、エイリアスの定義とインクルードコマンド以外のすべてを無視します。

メールエイリアスは abbrev のように — つまり、エイリアスの後で単語区切り文字をタイプするとすぐに — 展開されます。この展開はヘッダーフィールド‘`To`’、‘`From`’、‘`CC`’、‘`BCC`’、‘`Reply-To`’(およびそれらの‘`Resent-`’の変種) だけで展開されます。‘`Subject`’のような、他のヘッダーフィールドでは展開されません。

コマンド `M-x mail-abbrev-insert-alias`を使って、エイリアスされたアドレスを直接挿入することもできます。これは補完つきでエイリアス名を読み取り、ポイント位置にそれの定義を挿入します。

29.4 メールコマンド

`*mail*`バッファーのデフォルトのメジャーモードは、Message モードと呼ばれます。これは Text モードのように振る舞いますが、メッセージをより快適に編集するために、`C-c`プレフィクスを伴う、追加のコマンドをいくつか提供します。

このセクションでは、Message モードで利用可能な、もっとも一般的に使用されるコマンドを説明します。

29.4.1 メールの送信

`C-c C-c` そのメッセージを送信して、メールバッファーを隠し (bury) ます (`message-send-and-exit`)。

`C-c C-s` そのメッセージを送信して、メールバッファーを選択されたまま残します (`message-send`)。

メッセージを送信するために通常使用されるコマンドは、`C-c C-c` (`message-send-and-exit`) です。これはメッセージを送信して、メールバッファーを隠す (bury) — つまりバッファーを再選択するときもっとも低い優先順位になるようにします。かわりにメールバッファーを kill したい場合は、変数 `message-kill-buffer-on-exit`を `t`に変更してください。

コマンド `C-c C-s` (`message-send`) はメッセージを送信して、そのバッファーを選択されたまま残します。(たとえば新しい送信先のために) メッセージを変更して、再度送信したい場合に、このコマンドを使用します。

メッセージの送信により、フック `message-send-hook`が実行されます。メールバッファーがファイルを visit しているバッファーの場合をのぞき、送信によりメールバッファーは変更なし (unmodified) とマークされます (ファイルを visit している場合は、そのファイルを保存したときだけ変更なしとマークされます。このため、同じメッセージを 2 回送信しても警告はされません)。

変数 `message-send-mail-function`はメッセージの配送方法を制御します (`send-mail-function`は Mail モードで使用される)。`send-mail-function`の値は以下の関数のいずれかを指定します:

`sendmail-query-once`
: 配送方法 (このリストの他の項目のうちの 1 つ) を尋ね、このメッセージにその方法を使用します。`smtpmail-send-it`を通じてすでにメールの配送方法を変数にセットしていない場合 (以下参照)、これがデフォルトです。

`smtpmail-send-it`
: インターネットサービスプロバイダーの対外SMTPメールサーバーのような、外部のメールホストを通じてメールを送信します。そのSMTPサーバーに接続する方法をEmacsに指示していない場合、コマンドはこの情報の入力を求め、それは変数`smtpmail-smtp-server`と、ファイル`~/.authinfo`に保存されます。Section “Emacs SMTP Library” in *Sending mail via SMTP* を参照してください。

`sendmail-send-it`
: システムのデフォルトの`sendmail`プログラム、またはそれと等価なプログラムを使用してメールを送信します。これは、そのシステムがSMTPで直接メールを配送できるように、セットアップされている必要があります。

`mailclient-send-it`
: メールバッファーを、システムで指定されたメールクライアントに渡します。詳細はファイル`mailclient.el`の、コメントセクションを参照してください。

`feedmail-send-it`
: これは`sendmail-send-it`と同様ですが、後で送信するためにメッセージをキューできます。詳細は、ファイル`feedmail.el`の、コメントセクションを参照してください。

非ASCII文字を含むメッセージを送信する場合、それらはコーディングシステムでエンコードされる必要があります。コーディングシステムは通常、選択された言語環境 (Section 19.2 [Language Environments], page 218 を参照してください) により、自動的に指定されます。変数`sendmail-coding-system`をセットすることにより、対外に送信するメールのコーディングシステムを明示的に指定することができます (Section 19.6 [Recognize Coding], page 225 を参照してください)。そのようにして決定されたコーディングシステムで扱えない文字が、特定のメッセージに含まれる場合、Emacsは利用可能なコーディングシステムのリストを表示して、使用するコーディングシステムの選択を求めます。Section 19.8 [Output Coding], page 227 を参照してください。

29.4.2 メールヘッダーの編集

Messageモードは、特定のヘッダーフィールドに移動したり、ヘッダーのアドレスを補完する、以下の特別なコマンドを提供します。

`C-c C-f C-t`
: ‘`To`’ヘッダーに移動します (`message-goto-to`)。

`C-c C-f C-s`
: ‘`Subject`’ヘッダーに移動します (`message-goto-subject`)。

`C-c C-f C-c`
: ‘`CC`’ヘッダーに移動します (`message-goto-cc`)。

`C-c C-f C-b`
: ‘`BCC`’ヘッダーに移動します (`message-goto-bcc`)。

`C-c C-f C-r`
: ‘`Reply-To`’ヘッダーに移動します (`message-goto-reply-to`)。

`C-c C-f C-f`
: ‘`Mail-Followup-To`’ヘッダーフィールドに移動します (`message-goto-followup-to`)。

C-c C-f C-w
: ファイル名の補完つきで、新しい ‘FCC’ヘッダーフィールドを追加します (message-goto-fcc)。

C-c C-b
: メッセージ本文の先頭に移動します (message-goto-body)。

TAB
: メールアドレスを補完します (message-tab)。

特定のヘッダーフィールドにポイントを移動するコマンドは、すべて C-c C-fというプレフィクスを指定します (‘C-f’は “field” が由来です)。指定したフィールドが存在しない場合、コマンドは新たにそれを作成します (例外は mail-fccで、これは毎回新たなフィールドを作成します)。

コマンド C-c C-b (message-goto-body) は、ポイントをヘッダー区切り行の下 — つまり本文の先頭にポイントを移動します。

‘To:’、‘CC:’、‘BCC:’のようなアドレスを含むヘッダーフィールドを編集しているときは、TAB (message-tab) をタイプすることにより、アドレスを補完できます。これは 2 つの方法にもとづいて、そのアドレスにタイプするフルネームを挿入しようと試みます。まず複数のディレクトリーサーバープロトコルを認識する EUDC ライブラリーを試み (Section “EUDC” in *The Emacs Unified Directory Client* を参照してください)、それに失敗したときはメールエイリアスによりアドレスの展開を試みます (Section 29.3 [Mail Aliases], page 422 を参照してください)。メール本文のような、アドレスを要求しないヘッダーフィールドにポイントがあるとき、TABはタブ文字を挿入するだけです。

29.4.3 メールの引用

C-c C-y
: 選択されたメッセージを、引用のためにメールリーダーから yank します (message-yank-original)。

C-c C-q
: 他のメッセージから引用された各パラグラフをフィルします (message-fill-yanked-message)。

C-c C-y (message-yank-original) を使用して、返信するメッセージから “引用 (cite)” することができます。これは、そのメッセージのテキストをメールバッファーに挿入します。このコマンドは、Rmail のように、Emacs から実行されるメールリーダーから呼び出されたときだけ機能します。

デフォルトでは、Emacs は引用されたテキストの各行の前に、文字列 ‘>’を挿入します。このプレフィクス文字列は、変数 message-yank-prefixで指定されます。プレフィクス引数を指定して message-yank-originalを呼び出した場合、引用のためのプレフィクスは挿入されません。

C-c C-yを使用した後で、C-c C-q (message-fill-yanked-message) とタイプして、引用されたメッセージのパラグラフをフィルできます。C-c C-qの 1 つの使い方としては、そのようなパラグラフのすべてを個別にフィルすることです。クォートされたメッセージの 1 つのパラグラフをフィルするには、M-qを使います。フィルが、あなたが使用している引用プレフィクスを自動的に処理しない場合は、フィルプレフィクスを明示的にセットしてみてください。Section 22.6 [Filling], page 256 を参照してください。

フック mail-citation-hookを通じて、メールの引用をカスタマイズできます。たとえば、より柔軟な引用を提供する Supercite パッケージを使うことができます (Section “Introduction” in *Supercite* を参照してください)。

29.4.4 メール、その他

メールバッファーで C-c C-a (mml-attach-file) とタイプすることにより、送信するメッセージに、ファイルを添付 *(attach)* できます。添付は、MIME(Multipurpose Internet Mail Extensions) 標準を使って行なわれます。

`mml-attach-file`コマンドはファイル名と、添付ファイルの *content type(*内容のタイプ*)*、*description(*説明*)*、*disposition(*性質*)* の入力を求めます。通常 content type は自動的に検知されます。単に `RET`とタイプすると、そのデフォルトが適用されます。description は 1 行のテキストで、そのメールの受取人には添付ファイルの隣に表示されます。これは空にすることもできます。disposition は‘`inline`’、または‘`attachment`’のどちらかです。‘`inline`’の場合、メッセージ本文に添付ファイルへのリンクが表示され、‘`attachment`’の場合は、本文とは別にリンクが表示されます。

`mml-attach-file`コマンドは、Message モードに特有なコマンドです。Mail モードではかわりに、`mail-add-attachment`を使用します。これはファイル名の入力だけを求め、content type と disposition は自動的に決定されます。添付ファイルの説明を含めたい場合は、それをメッセージの本文にタイプしてください。

添付ファイルの実際の内容は、メールバッファーには挿入されません。かわりに、以下のような代替テキストがメールバッファーに挿入されます:

```
<#part type="text/plain" filename="~/foo.txt" disposition=inline>
<#/part>
```

`C-c C-c`または `C-c C-s`でメッセージを送信するとき、それと一緒に添付ファイルも送信されます。

メッセージを作成しているとき、`M-x ispell-message`とタイプして、メッセージテキストのスペル訂正を行なうことができます。受信メッセージから送信用の下書きに yank した場合、このコマンドは yank されたテキストをスキップして、あなた自身が挿入したテキストだけをチェックします (このコマンドはインデント、またはあなたの入力と引用された行を区別する `mail-yank-prefix`を調べます)。Section 13.4 [Spelling], page 134 を参照してください。

Message モードをオンに切り替えると (`C-x m`は自動的にこれを行ないます)、ノーマルフックの `text-mode-hook`と `message-mode-hook`が自動的に実行されます。新しい送信メッセージの初期化では、ノーマルフック `message-setup-hook`が実行されます。メールバッファーの外観を変更したい場合は、このフックを使うことができます。Section 33.2.2 [Hooks], page 509 を参照してください。

これらのフックの主な違いは、それらが呼び出されるタイミングだけです。`C-x m`, `message-mode-hook`とタイプしたときは、メールバッファーが作成された直後に `message-mode-hook`が実行されます。その後 `message-setup`関数がそのバッファーのデフォルトの内容を挿入します。これらのデフォルトの内容が挿入された後に、`message-setup-hook`が実行されます。

既存のメッセージにたいして、`C-x m`で作成を継続する場合、そのメールバッファーに切り替えた直後に、`message-mode-hook`が実行されます。バッファーが変更されていない場合、またはそれを削除して新たに作成を開始する場合には、デフォルトの内容が挿入された後に、`message-setup-hook`が実行されます。

29.5 メール署名

各メッセージの最後に標準的なテキストの断片 — メール署名 *(mail signature)* — を追加できます。この署名には、あなたの電話番号や住所などの情報を含めることができます。変数 `message-signature`は、Emacs がメール署名を扱う方法を決定します。

`message-signature`のデフォルト値は `t`です。これはメール署名をファイル`~/.signature`から探すことを意味します。ファイルが存在する場合、そのファイルの内容がメールバッファーの最後に自動的に挿入されます。変数 `message-signature-file`を通じて署名ファイルを変更できます。

`message-signature`を文字列に変更すると、それは署名のテキストを直接指定することになります。

`message-signature`を`nil`に変更した場合、Emacsはメール署名を自動的に挿入しません。メールバッファーで`C-c C-w` (`message-insert-signature`) とタイプすることにより、メール署名を挿入できます。この場合も、Emacsは署名ファイルから署名を探します。

メールを作成するのにMessageモードではなくMailモードを使用する場合、どのようにして署名を送るか決定する変数は、`mail-signature`と`mail-signature-file`になります。

慣例により、メール署名は行の内容が '`-- `'であることによりマークされるべきです。署名にこのプレフィクスがない場合、このプレフィクスが追加されます。署名は4行を超えないようにするべきです。

29.6 アミューズメント

`M-x spook`は、送信するメールメッセージに、ランダムに選択されたキーワードの行を追加します。これらのキーワードは、あなたが何らかの破壊活動を企てていると思わせるような単語のリストから選択されます。

この機能の背後にあるアイデアは、NSA[1]や他の情報機関が、かれらが関心をもつキーワードを含むすべてのインターネットメールのメッセージを監視しているという疑惑です (そのような政府機関は、“やっていない”と言いますが、もちろんかれらはそう言うでしょう)。このアイデアは、もし多くの人々がメッセージに不審な単語を含めれば、政府機関は不審な入力で手一杯になり、やがて最後はそれらを読むのを止めるだろうというものです。これが本当かどうかはわかりませんが、少なくとも人々を楽しませることができます。

`fortune`プログラムを使用して、送信メールにフォーチューンクッキー (fortune cookie) メッセージを追加できます。これを行なうには`mail-setup-hook`に`fortune-to-signature`を追加してください。

```
(add-hook 'mail-setup-hook 'fortune-to-signature)
```

多分、これを使う前に変数`fortune-file`をセットする必要があるでしょう。

29.7 メール作成方法

このチャプターでは、メールを編集したり送信する通常のEmacsモードである、Messageモードを説明してきました。これは、いくつかの利用可能なモードのうちの1つに過ぎません。Emacs23.2以前では、デフォルトのモードはMailモードで、これは多くの点でMessageモードに似ていますが、MIMEサポートのような機能がありません。その他の利用可能なモードはMH-Eです (Section “MH-E” in *The Emacs Interface to MH* を参照してください)。

これらのメールユーザーエージェント *(mail user agents)* から、メールを編集したり送信するための、好みの方法を選択できます。コマンド`C-x m`、`C-x 4 m`、`C-x 5 m`は指定されたエージェントを使用するので、Emacsでメールを送信する他の様々な部分 (たとえばバグリポーターなど、Section 34.3 [Bugs], page 542 を参照してください) もこれを行ないます。メールユーザーエージェントを指定するには、変数`mail-user-agent`をカスタマイズします。現在のところ、正式な値には`message-user-agent`(Messageモード)、`sendmail-user-agent`(Mailモード)、`gnus-user-agent`、`mh-e-user-agent`が含まれます。更に追加のオプションが利用できるかもしれません。詳細についてはあなたのメールユーザーエージェントのマニュアルを確認してください。`define-mail-user-agent`を使用すれば他のメールユーザーエージェントを定義することもできます。

他のメール作成方法を選択した場合、メールバッファーとMessageモードに関するこのチャプターの情報は適用できません。他の方法は違うバッファーで異なるフォーマットのテキストを使用し、コマンドも異なります。

[1] The US National Security Agency.

同様に、メールを読むための好みの方法を指定するには、変数 `read-mail-command`をカスタマイズします。デフォルトは `rmail`です (Chapter 30 [Rmail], page 429 を参照してください)。

30 Rmail でメールを読む

Rmail は、メールを閲覧したり処理するための、Emacs のサブシステムです。Rmail は、Rmail ファイルを呼ばれるファイルに、メールメッセージを保存します。Rmail ファイルの中のメッセージの閲覧は、Rmail モードという特別なメジャーモードで行なわれます。このモードはメールを管理するために実行するコマンドのために、多くの文字を再定義します。

Emacs には、より複雑かつ柔軟な Gnus と呼ばれるメールを読むためのサブシステムが同梱されています。Gnus は巨大なパッケージなので、独自のマニュアル *The Gnus Newsreader* で説明されているので、それを参照してください。

30.1 Rmail の基本的な概念

もっとも簡単な方法で Rmail を使用するには、メールが保存される`~/RMAIL`という Rmail ファイルを使用します。これはプライマリー *Rmail* ファイル *(primary Rmail file)* と呼ばれます。コマンド `M-x rmail`はプライマリー Rmail ファイルを読み込み、inbox(受信箱) から新しいメールをマージして、未読の最初のメッセージを表示して、それを閲覧出きるようにします。変数 `rmail-file-name` はプライマリー Rmail ファイルの名前を指定します。

Rmail は、Rmail ファイルのメッセージを、1 度に 1 つだけ表示します。表示されているメッセージは、カレントメッセージ *(current message:* 現在のメッセージ*)* と呼ばれます。Rmail モードの特別なコマンドは、カレントメッセージの削除、他のファイルへのコピー、返信、他のメッセージへの移動を行なうことができます。複数の Rmail ファイル (Chapter 15 [Files], page 145 を参照) を作成して、それらの間でメッセージの移動をするのに、Rmail を使用することができます (Section 30.7 [Rmail Output], page 434 を参照)。

Rmail ファイルでは通常、メッセージは受信した順になっています。それらをソートする他の方法を指定できます (Section 30.12 [Rmail Sorting], page 442 を参照してください)。メッセージは連続する整数で識別され、それはメッセージナンバー *(message numbers)* と呼ばれます。カレントメッセージのナンバーは Rmail のモードラインに表示され、その後にはファイル内のメッセージの総数が続きます。`j`でメッセージナンバーを指定して、そのメッセージに移動できます (Section 30.3 [Rmail Motion], page 430 を参照してください)。

通常の Emacs の慣例にしたがい、Rmail での変更は、そのファイルを保存したときだけ永続化されます。`s` (`rmail-expunge-and-save`) で、ファイルを保存することができます、これは最初に削除されたメッセージをファイルから完全に削除します (Section 30.4 [Rmail Deletion], page 431 を参照してください)。完全な削除を行なわずにファイルを保存するには、`C-x C-s`を使用します。Rmail は、inbox ファイルから新しいメールをマージした後に、自動的に Rmail ファイルを保存します。

Rmail を exit するには、`q` (`rmail-quit`) を使用します。これは Rmail にたいして完全な削除と保存を行い、Rmail バッファーと、(表示されていれば) サマリーバッファーを隠します (Section 30.11 [Rmail Summary], page 439 を参照してください)。しかし正式に exit する必要はありません。Rmail から他のバッファーを編集するために切り替えて、2 度と Rmail に戻らなければ、それは exit したことになります。(他の変更したファイルと同様に) 最終的に Rmail ファイルを確実に保存することだけが重要です。これを行なうには`C-x s`が適しています (Section 15.3.1 [Save Commands], page 150 を参照してください)。Rmail コマンドの `b rmail-bury`は、Rmail ファイルにたいする完全な削除と保存を行なわずに、Rmail バッファーとサマリーを隠します。

30.2 メッセージのスクロール

Rmail が画面に収まらないメッセージを表示しているときは、残りを読むためにスクロールしなければなりません。通常のスクロールコマンド `C-v`、`M-v`、`M-<` (Section 11.1 [Scrolling], page 77 を参

照) などでこれを行なうことができますが、Rmail でのスクロールは頻繁に行なわれるので、簡単に行なえるようにする価値があります。

SPC 前方にスクロールします (scroll-up-command)。

DEL
S-SPC 後方にスクロールします (scroll-down-command)。

. メッセージの最初にスクロールします (rmail-beginning-of-message)。

/ メッセージの最後にスクロールします (rmail-end-of-message)。

メッセージを読むときにもっとも一般的に行なうのは、画面単位でメッセージをスクロールすることなので、Rmail は SPCと DEL(または S-SPC) で、C-v (scroll-up-command) と M-v (scroll-down-command) と同じことを行なうようにしています。

コマンド. (rmail-beginning-of-message) は、選択されたメッセージの最初に、後方へスクロールします。これは M-<とまったく同じではありません。このコマンドはマークをセットしません。他にも、カレントメッセージのバッファー境界を変更していた場合 (たとえば編集により。Section 30.15 [Rmail Editing], page 445 を参照されたい) は、それをリセットします。同様に、コマンド/ (rmail-end-of-message) は、選択されたメッセージの最後に、前方へスクロールします。

30.3 メッセージ間の移動

メッセージにたいして行なうもっとも基本的なことは、それを読むことです。Rmail でこれを行なうために、そのメッセージをカレントにします。通常の方法はファイルを、受信したメッセージ順に移動していく方法です (その最初のメッセージは 'unseen'(未読) の属性をもちます。Section 30.9 [Rmail Attributes], page 437 を参照してください)。他の新しいメッセージを読むには、前方に移動します。古いメッセージを再読するには後方に移動します。

n 間にある削除されたメッセージをスキップして、次の削除されていないメッセージに移動します (rmail-next-undeleted-message)。

p 前の削除されていないメッセージに移動します (rmail-previous-undeleted-message)。

M-n 削除されたメッセージも含めて、次のメッセージに移動します (rmail-next-message)。

M-p 削除されたメッセージも含めて、前のメッセージに移動します (rmail-previous-message)。

C-c C-n カレントメッセージと同じ subject の、次のメッセージに移動します (rmail-next-same-subject)。

C-c C-p カレントメッセージと同じ subject の、前のメッセージに移動します (rmail-previous-same-subject)。

j 最初のメッセージに移動します。引数 *n*を指定すると、*n*番目のメッセージに移動します (rmail-show-message)。

> 最後のメッセージに移動します (rmail-last-message)。

< 最初のメッセージに移動します (rmail-first-message)。

M-s *regexp* RET
*regexp*へのマッチを含む、次のメッセージに移動します (rmail-search)。

- M-s *regexp* RET
  *regexp*へのマッチを含む、前のメッセージに移動します (これは負の引数による M-sである)。

nと pは、Rmail でメッセージを移動する通常の方法です。これらは、(通常そうしたいように) 削除されたメッセージをスキップして、メッセージを順番に移動していきます。これらのコマンドの定義には、rmail-next-undeleted-messageおよび rmail-previous-undeleted-messageという名前がつけられています。削除されたメッセージをスキップしたくない場合— たとえばメッセージの削除を取り消すために — は、変種の M-nと M-p(rmail-next-messageと rmail-previous-message) を使います。これらのコマンドへの数引数は、繰り返し回数を指定します。

Rmail では数引数の指定は、単に数字をタイプして行なうことができます。最初に C-uをタイプする必要はありません。単に-とタイプして、負の引数を指定することもできます。

M-s (rmail-search) は、Rmail 版の検索コマンドです。通常のインクリメンタル検索 C-sは機能しますが、これはカレントメッセージだけを検索します。M-sの目的は、他のメッセージにたいする検索です。これは非インクリメンタルに正規表現 (Section 12.6 [Regexps], page 115 を参照してください) を読み取り、後続のメッセージの先頭から検索を開始して、見つかったらそのメッセージを選択します。*regexp*が空の場合、M-sは前回使用した regexp を再使用します。

ファイルの中の他のメッセージにたいして後方に検索するには、M-sに負の引数を与えます。Rmail では- M-sでこれを行なうことができます。これは前のメッセージの最後から検索を開始します。

ラベルにもとづく検索も可能です。Section 30.8 [Rmail Labels], page 436 を参照してください。

C-c C-n (rmail-next-same-subject) コマンドは、カレントメッセージと同じ subject をもつ、次のメッセージに移動します。プレフィクス引数は繰り返し回数として使用されます。負の引数を指定すると、C-c C-p (rmail-previous-same-subject) のように、後方に移動します。subject を比較するとき、subject への返信に通常付加されるようなプレフィクスは無視します。これらのコマンドは、同じ subject に関するすべてのメッセージ、いわゆる thread を読むときに有用です。

メッセージの絶対番号を指定してメッセージに移動するには、メッセージ番号を引数として、j (rmail-show-message) を使用します。引数を与えない場合、jは最初のメッセージに移動します。< (rmail-first-message) も最初のメッセージを選択します。> (rmail-last-message) は最後のメッセージを選択します。

30.4 メッセージの削除

メッセージを残す必要がなくなったとき、それを削除 *(delete)* できます。これはそのメッセージを無視するフラグをつけ、いくつかの Rmail コマンドは、そのメッセージが存在しないかのように振る舞います。しかし、そのメッセージはまだ Rmail ファイルの中にあり、メッセージ番号ももっています。

Rmail ファイルにたいして完全な削除 *(expunging)* を行なうことにより、削除されたメッセージを実際に消去します。残ったメッセージには新たに連番が振られます。

d
: カレントメッセージを削除して、次の削除されていないメッセージに移動します (rmail-delete-forward)。

C-d
: カレントメッセージを削除して、前の削除されていないメッセージに移動します (rmail-delete-backward)。

u
: カレントメッセージの削除を取り消すか、前の削除されたメッセージに後方へ移動して、そのメッセージの削除を取り消します (rmail-undelete-previous-message)。

x
: Rmail ファイルにたいして完全な削除を行ないます (rmail-expunge)。

Rmail には、メッセージを削除するためのコマンドが 2 つあります。両方ともカレントメッセージを削除して、他のメッセージを選択します。`d` (`rmail-delete-forward`) は、すでに削除されたメッセージをスキップして次のメッセージに移動し、`C-d` (`rmail-delete-backward`) は、前の削除されていないメッセージに移動します。指定方向に、移動先となる削除されていないメッセージが存在しない場合は、単にそのメッセージを削除するだけで、カレントメッセージはそのメッセージのままです。数引数は繰り返し回数を指定します。これにより 1 つのコマンドで複数のメッセージを削除できます。負の引数は `d`と `C-d`の意味を逆転します。

Rmail がメッセージを削除するときは、フック `rmail-delete-message-hook`が実行されます。フック関数が呼び出されるとき、そのメッセージは削除とマークされますが、そのメッセージが Rmail バッファーのカレントメッセージのままです。

すべての削除されたメッセージを最終的に Rmail ファイルから消すには、`x` (`rmail-expunge`) とタイプします。これを行なうまでは、削除されたメッセージの削除を取り消す *(undelete)* ことができます。削除の取り消しコマンド `u` (`rmail-undelete-previous-message`) は、ほとんどのケースにおいて `d`コマンドの効果を取り消すようにデザインされています。カレントメッセージが削除されている場合は、カレントメッセージの削除を取り消します。そうでない場合は、削除されたメッセージが見つかるまで後方に移動して、そのメッセージの削除を取り消します。数引数は繰り返し回数を指定します。これにより 1 つのコマンドで複数のメッセージ削除を取り消すことができます。。

通常、`d`を `u`で取り消すことができます。なぜなら `u`は後方に移動して、`d`で削除されたメッセージの削除を取り消すからです。しかしこれは、削除するメッセージの前にすでに削除されたメッセージがある場合、`d`はこれらのメッセージをスキップするのでうまく機能しません。その後で `u`コマンドを実行すると、スキップされた最後のメッセージの削除を取り消すからです。この問題を避ける明解な方法はありません。しかし `u`コマンドを繰り返すことにより、削除を取り消したいメッセージに戻ることができます。`M-p`コマンドで特定の削除されたメッセージを選択してから、`u`をタイプして削除を取り消すこともできます。

削除されたメッセージは ‘`deleted`’の属性をもち、結果として、カレントメッセージが削除されている場合はモードラインに ‘`deleted`’が表示されます。実際のところ、メッセージの削除と削除の取り消しは、この属性の追加または削除に過ぎません。Section 30.9 [Rmail Attributes], page 437 を参照してください。

30.5 Rmail ファイルと inbox

ローカルでメールを受信したとき、オペレーティングシステムは受信メールを、私たちが *inbox* と呼ぶファイルに配します。Rmail を開始したとき、`movemail`と呼ばれる C プログラムを実行して、inbox から新しいメッセージを、Rmail セッションの Rmail ファイルにコピーします。この Rmail ファイルには、以前の Rmail セッションの他のメッセージも含まれています。Rmail で実際に読むメールは、このファイルの中にあります。この操作は新しいメールの取得 *(getting new mail)* と呼ばれます。`g`とタイプすることにより、いつでも新しいメールを取得できます。

変数 `rmail-primary-inbox-list`は、プライマリー Rmail ファイルにたいする inbox ファイルのリストを含みます。この変数を明示的にセットしない場合、Rmail は環境変数 `MAIL`を使用するか、最後の手段として `rmail-spool-directory`にもとづく、デフォルトの inbox を使用します。デフォルトの inbox はオペレーティングシステムに依存し、それは`/var/mail/`*`username`*、`/var/spool/mail/`*`username`*、`/usr/spool/mail/`*`username`*などです。

コマンド `set-rmail-inbox-list`で、カレントセッションでの任意の Rmail ファイルにたいする inbox ファイルを指定できます。Section 30.6 [Rmail Files], page 433 を参照してください。

inbox とは別に Rmail ファイルをもつべき理由が 2 つあります。

1. inbox ファイルのフォーマットは、オペレーティングシステムと、それを使用する他のメールソフトによりさまざまです。Rmail の一部だけがそれらの候補を理解していればよく、それらすべてを Rmail 自身のフォーマットに変換する方法だけを理解すればよいからです。
2. メールを紛失せずに inbox にアクセスするのは厄介です。なぜならそれはメール配信とインターロック (連動) する必要があるからです。さらにオペレーティングシステムごとに、異なるインターロック技術が使用されています。inbox から別の Rmail ファイルに 1 度メールを移動する方法により、Rmail の残りのすべてがインターロックの必要性を無視できます。なぜなら Rmail は Rmail ファイルだけを操作すればよいからです。

Rmail は、Rmail ファイルの内部フォーマットとして、Unix および GNU システムに取り入れられた、標準的な '`mbox`'フォーマットを使用します (実際のところ、mbox フォーマットとは若干の違いがあります。その違いは重要ではありませんが、変数 `rmail-mbox-format`をセットすることにより、あなたのシステムが使用するフォーマットを Rmail に指定できます。詳細は、変数のドキュメントを参照してください)。

新しいメールを受信したとき、Rmail は最初にその新しいメールを inbox ファイルから Rmail ファイルにコピーします。それから Rmail ファイルを保存して、その後で inbox ファイルからそれをクリアーします。この方法では、システムのクラッシュにより、inbox と Rmail ファイルの間でメールの重複は発生するかもしれませんが、メールを失うことはあり得ません。`rmail-preserve-inbox` が非 `nil`の場合、Rmail は新しいメールを受信したときに inbox ファイルをクリアーしません。旅行の際など、携帯用のコンピューターで POP を通じてメールをチェックするときは、この変数をセットすれば、メールはサーバーに残るので、後であなたがメインに使用するワークステーションのデスクトップに保存することができます。

Rmail が inbox ファイルから間接的に新しいメールをコピーするケースがあります。最初に `movemail`プログラムを実行して inbox から、Rmail ファイルと同じディレクトリーにある、`.newmail-`*`inboxname`*と呼ばれる中間ファイルにメールを移動します。その後、Rmail は、そのファイルから新しいメールをマージして、Rmail ファイルを保存し、中間ファイルの削除はその後にだけ行なわれます。悪いタイミングでクラッシュが発生した場合、中間ファイルは残っているので、Rmail は次に inbox ファイルから新しいメールを取得するとき、それを再使用します。

Rmail が`.newmail-`*`inboxname`*の中のデータを mbox 形式に変換できない場合、ファイルを `RMAILOSE.n`(*n*はファイル名を一意にするために選ばれます) にリネームするので、Rmail はそのデータで再度問題を起こすことはなくなります。メッセージの何が Rmail を混乱させたか調べて、それを削除すべきです (大抵は 8 進コード 037 の control-underscore がメッセージに含まれている場合です)。その後、修正されたファイルから `1 g`を使って新しいメールを取得できます。

30.6 複数の Rmail ファイル

Rmail はデフォルトで、あなたのプライマリー *Rmail* ファイル *(primary Rmail file)* を操作します。これは`~/RMAIL`というファイルで、inbox ファイルからメールを受け取ります。しかし他の Rmail ファイルを所有して。Rmail でそれを編集することができます。これらのファイルは、それら自身の inbox からメールを受け取ったり、明示的な Rmail コマンドでメッセージを移動することができます (Section 30.7 [Rmail Output], page 434 を参照してください)。

`i` *`file`* `RET`
: *file*を Emacs に読み込んで、それにたいして Rmail を実行します (`rmail-input`)。

`g`
: カレント Rmail ファイルの inbox から、新しいメールをマージします (`rmail-get-new-mail`)。

`C-u g` *file* `RET`
: inbox ファイル *file*から新しいメールをマージします。

プライマリー Rmail ファイル以外のファイルで Rmail を実行するために、Rmail で `i` (`rmail-input`) コマンドを使用できます。これは、そのファイルを Rmail モードで visit します。Rmail の外からでも `M-x rmail-input`を使用することができますが、同じことを行なう `C-u M-x rmail`の方が簡単にタイプできます。

通常 `i`で読み込むファイルは、有効な mbox ファイルであるべきです。そうでない場合、Rmail はそのファイルのテキストを mbox 形式に変換しようと試み、そのバッファーで変換されたテキストを visit します。バッファーを保存すると、そのファイルが変換されます。

存在しないファイル名を指定した場合、`i`は新しい Rmail ファイルを作成するために、新しいバッファーを初期化します。

メニューから Rmail ファイルを選択することもできます。メニュー Classify の、アイテム Input Rmail File を選択して、Rmail ファイルを選択します。変数 `rmail-secondary-file-directory` および `rmail-secondary-file-regexp`は、メニューがどのファイルを表示するかを指定します。最初の変数はファイルを探すディレクトリーを指定し、2 番目の変数はそのディレクトリーのどのファイル (正規表現にマッチするファイルすべて) を表示するかを指定します。マッチするファイルがない場合、このメニューアイテムは選択できません。これらの変数は、出力するファイルの選択にも適用されます (Section 30.7 [Rmail Output], page 434 を参照してください)。

使用する inbox ファイルは変数 `rmail-inbox-list`により指定され、これは Rmail モードではバッファーローカルな変数です。特別な例外として、プライマリー Rmail ファイルに inbox を指定していない場合、これは環境変数 `MAIL`、またはシステム標準の inbox を使用します。

`g` (`rmail-get-new-mail`) コマンドは、inbox のメールを、カレント Rmail ファイルにマージします。Rmail ファイルに inbox がない場合、`g`は何もしません。コマンド `M-x rmail`も、新しいメールをプライマリー Rmail ファイルにマージします。

通常の inbox ではないファイルからメールをマージするには、`C-u g`のように `g`キーに数引数を与えます。するとファイル名を読み取り、そのファイルからメールをマージします。引数を使用して `g`を使用しても、inbox ファイルの削除・変更はされません。したがって、これはあるファイルのメッセージを、他のファイルにマージする一般的な方法です。

30.7 外部ファイルへのメッセージのコピー

以下は Rmail ファイルから他のファイルにメッセージをコピーするコマンドです。

`o` *file* `RET`
: カレントメッセージの完全なコピーを、ファイル *file*に追加します (`rmail-output`)。

`C-o` *file* `RET`
: カレントメッセージの表示にしたがい、ファイル *file*に追加します (`rmail-output-as-seen`)。

`w` *file* `RET`
: メッセージの本文だけをファイル *file*に出力します。デフォルトのファイル名は、そのメッセージの ‘`Subject`’ヘッダーからとられます。

コマンド `o`と `C-o`はカレントメッセージを指定したファイルの後尾に追加することによりコピーします。正のプレフィクス引数は繰り返し回数として振る舞います。カレントメッセージから開始して削除済みメッセージを無視しつつ後続のメッセージをその個数分コピーします。

2 つのコマンドの主な違いは、どれだけコピーするかです。C-oが現在表示されているヘッダーだけをコピーするのにたいし、oはヘッダーがすべて表示されていなくても、メッセージヘッダーを完全にコピーします。Section 30.13 [Rmail Display], page 443 を参照してください。加えて、ファイルが Babyl フォーマットのとき、oはメッセージを Babyl フォーマットに変換しますが、C-oは Babyl ファイルを出力できません。

Emacs バッファーで出力ファイルを visit していた場合、出力コマンドはメッセージをそのバッファーに追加します。最終的にそのバッファーをファイルに保存するかは、あなた次第です。

本文にファイル内容がそのまま記載されているようなメッセージを受信することがあるかもしれません。そのような場合、w (rmail-output-body-to-file) コマンドで、本文を (メッセージヘッダーを除いて) ファイルに保存できます。そのようなメッセージは‘Subject’フィールドにファイル名を意図した内容を含んでいる場合があるので、wコマンドは (ファイル名に可搬的に使用できないいくつかの文字を置換した後) デフォルトの出力ファイル名に‘Subject’フィールドを使用します。しかし、ファイル名はミニバッファーを使って読み取られるので、異なる名前を指定できます。

メニューから Rmail ファイルを選択して、メッセージを出力することもできます。メニュー Classify の、メニューアイテム Output Rmail File を選択して、出力したい Rmail ファイルを選択します。これはoコマンドのように、カレントメッセージをそのファイルに出力します。変数 rmail-secondary-file-directoryおよび rmail-secondary-file-regexpは、メニューがどのファイルを表示するかを指定します。最初の変数はファイルを探すディレクトリーを指定し、2 番目の変数はそのディレクトリーのどのファイル (正規表現にマッチするファイルすべて) を表示するかを指定します。マッチするファイルがない場合、このメニューアイテムは選択できません。

oまたはC-oでメッセージをコピーすることにより、メッセージのオリジナルコピーには属性‘filed’が与えられるので、そのメッセージがカレントのときは、モードラインに‘filed’が表示されます。

各メールメッセージにたいして 1 つのコピーを保持したい場合は、変数 rmail-delete-after-outputに tをセットします。その場合、コマンド o、C-oおよび wは、コピー後にオリジナルのメッセージを削除します (望むなら後で削除を取り消すことができる。Section 30.4 [Rmail Deletion], page 431 を参照されたい)。

デフォルトでは oは出力したメッセージの削除ステータスを元メッセージのままに留めます。したがって出力される前に削除済みのメッセージは出力ファイル中に削除済みとして現われます。変数 rmail-output-reset-deleted-flagを非 nil値にセットすることによりこれを回避できます。メッセージのコピーは削除ステータスがリセットされるので出力ファイル中には未削除として現れます。さらにこの変数が非 nilで oに正の引数を指定すると、後続メッセージから出力メッセージを探す際に削除済みメッセージを無視しなくなります。

変数 rmail-output-file-alistは、カレントメッセージの内容にもとづいて、理にかなったデフォルトの出力ファイルを指定できます。値は以下の形式をもつ要素のリストです:

```
(regexp . name-exp)
```

カレントメッセージに *regexp*にたいするマッチが存在する場合、デフォルトの出力ファイルは *name-exp*になります。複数の要素がそのメッセージにマッチする場合、最初にマッチした要素がデフォルトのファイル名を決定します。式 *name-exp*は使用するファイル名を与える文字列定数、またはより一般的に、ファイル名を文字列として取得する任意の Lisp 式を指定できます。rmail-output-file-alist は、oと C-oの両方に適用されます。

Rmail は、(rmail-file-nameで指定される) プライマリー Rmail ファイルから、(変数 rmail-automatic-folder-directivesの値にもとづいて) 他のファイルにメッセージを自動的に保存できます。この変数は、どのメッセージをどこに保存するかを指定する要素 (‘directives’) のリストです。各 directive は出力ファイルからなるリストで、ヘッダー名と正規表現の組が 1 つ以上後に続きます。メッセージのヘッダーが指定された正規表現にマッチする場合、そのメッセージは

与えられたファイルに保存されます。directive が複数のヘッダーエントリーをもつ場合、それらすべてがマッチしなければなりません。Rmail はファイル `rmail-file-name`からメッセージを表示するとき directive をチェックして、(もしあれば) 最初のマッチに適用します。出力ファイルが `nil`の場合、そのメッセージは削除され、保存されません。たとえば特定のアドレスや、特定の subject のメッセージを保存するのに、この機能を使用することができます。

30.8 ラベル

各メッセージは、分類 (classification) のために割り当てられる、さまざまなラベル *(labels)* をもつことができます。各ラベルは名前をもち、名前が異なると違うラベルになります。任意のラベルは、特定のメッセージにたいして、付いているか付いていないかのどちらかです。標準的な意味をもつラベル名がいくつかあり、それが適切なときは、Rmail により自動的にメッセージに付与されます。これらの特別なラベルは、属性 *(attribute)* と呼ばれますそれ以外のすべてのラベルは、ユーザーにより付与されます。

`a` *`label`* `RET`
: カレントメッセージに、ラベル *label*を割り当てます (`rmail-add-label`)。

`k` *`label`* `RET`
: カレントメッセージから、ラベル *label*を外します (`rmail-kill-label`)。

`C-M-n` *`labels`* `RET`
: 複数のラベル *labels*のどれか 1 つをもつ、次のメッセージに移動します (`rmail-next-labeled-message`)。

`C-M-p` *`labels`* `RET`
: 複数のラベル *labels*のどれか 1 つをもつ、前のメッセージに移動します (`rmail-previous-labeled-message`)。

`l` *`labels`* `RET`
`C-M-l` *`labels`* `RET`
: 複数のラベル *labels*のどれかを含む、すべてのメッセージのサマリーを作成します (`rmail-summary-by-labels`)。

コマンド `a` (`rmail-add-label`) および `k` (`rmail-kill-label`) で、カレントメッセージにたいして任意のラベルを割り当てたり、外すことができます。引数 *label*が空の場合、もっとも最近割り当てられた (または外された) ラベルを割り当てる (または外す) ことを意味します。

メッセージを分類するためにラベルを割り当てた後、ラベルを使用する 3 つの方法 — 移動、サマリー、ソート — があります。

`C-M-n` *`labels`* `RET` (`rmail-next-labeled-message`) は、複数のラベル *labels*のうちどれか 1 つをもつ、次のメッセージに移動します。引数 *labels*には、カンマで区切られた 1 つ以上のラベル名を指定します。`C-M-p` (`rmail-previous-labeled-message`) も同様ですが、前のメッセージに後方へ移動します。どちらのコマンドも、数引数は繰り返し回数を指定します。

コマンド `C-M-l` *`labels`* `RET` (`rmail-summary-by-labels`) は、指定された複数のラベルのうち、少なくとも 1 つをもつメッセージだけを含むサマリーを表示します。引数 *labels*はカンマで区切られた 1 つ以上のラベル名です。サマリーについての詳細は、Section 30.11 [Rmail Summary], page 439 を参照してください。

`C-M-n`、`C-M-p`、`C-M-l`にたいして引数 *labels*が空の場合は、それらのコマンドにたいして、もっとも最近に指定された *labels*を使うことを意味します。

ラベルでメッセージをソートする情報については、Section 30.12 [Rmail Sorting], page 442 を参照してください。

30.9 Rmail の属性

‘deleted’や‘filed’のようないくつかのラベルはビルトインの意味をもち、Rmail は適切なときに、それらをメッセージに割り当てます。これらのラベルは属性 *(attributes)* と呼ばれます。以下は Rmail の属性のリストです:

‘unseen’ そのメッセージが 1 度もカレントになっていないことを意味します。inbox からメッセージが到着したとき割り当てられ、そのメッセージがカレントになったときに外されます。Rmail を開始したとき、この属性をもつメッセージを最初に表示します。

‘deleted’ メッセージが削除されたことを意味します。削除コマンドにより割り当てられ、削除を取り消すコマンドで外されます (Section 30.4 [Rmail Deletion], page 431 を参照してください)。

‘filed’ そのメッセージが他のファイルにコピーされたことを意味します。ファイル出力コマンド `o`および `C-o`により割り当てられます (Section 30.7 [Rmail Output], page 434 を参照してください)。

‘answered’
メッセージへの返信をメールしたことを意味します。`r` (`rmail-reply`) コマンドにより割り当てられます。Section 30.10 [Rmail Reply], page 437 を参照してください。

‘forwarded’
メッセージを転送したことを意味します。`f` (`rmail-forward`) コマンドにより割り当てられます。Section 30.10 [Rmail Reply], page 437 を参照してください。

‘edited’ メッセージのテキストを Rmail で編集したことを意味します。Section 30.15 [Rmail Editing], page 445 を参照してください。

‘resent’ メッセージを再送したことを意味します。コマンド `M-x rmail-resend`により割り当てられます。Section 30.10 [Rmail Reply], page 437 を参照してください。

‘retried’ 送信に失敗したメッセージを再試行したことを意味します。コマンド `M-x rmail-retry-failure`により割り当てられます。Section 30.10 [Rmail Reply], page 437 を参照してください。

これ以外のすべてのラベルは、ユーザーだけが割り当てたり外すことができ、それらのラベルは標準的な意味をもちません。

30.10 返信の送信

Rmail には、送信メールを送るための複数のコマンドがあります。Message モードの使い方 (Rmail でも動作する特別な機能を含む) に関する情報は、Chapter 29 [Sending Mail], page 420 を参照してください。このセクションでは、送信メッセージ作成に使用する、mail バッファーに入るエンターするための Rmail の特別なコマンドを説明します。メールを送信するための通常のキー— `C-x m`、`C-x 4 m`、`C-x 5 m` — は、Rmail モードでも通常どおり機能することに注意してください。

`m` メッセージを送信します (`rmail-mail`)。

`c` すでに編集を開始した送信メッセージの編集を続けます (`rmail-continue`)。

`r` カレント Rmail メッセージにたいする返信を送信します (`rmail-reply`)。

`f` カレントメッセージを他のユーザーに転送します (`rmail-forward`)。

`C-u f` カレントメッセージを他のユーザーに再送します (`rmail-resend`)。

`M-m` 送信に失敗して戻ってきたメッセージにたいして、2 回目の送信を試みます (`rmail-retry-failure`)。

Rmail にいるときにメッセージを送信する理由でもっとも一般的なのは、読んでいるメールに返信するときでしょう。これを行なうには、`r` (`rmail-reply`) とタイプします。これは `C-x 4 m`のように、別ウィンドウにメール作成バッファーを表示しますが、ヘッダーフィールド '`Subject`'、'`To`'、'`CC`'、'`In-reply-To`'、'`References`'は、返信するメッセージにもとづいて、事前に初期化されています。'`To`'フィールドには、返信するメッセージを送信した人のアドレスがセットされ、'`CC`'にはそのメッセージを受け取った、他のすべての人のアドレスがセットされます。

変数 `mail-dont-reply-to-names`を使用して、自動的に返信に含まれる受信者から、特定の受信者を除外することができます。この変数の値には正規表現を指定します。正規表現にマッチする受信者は、'`CC`'フィールドから除外されます。その受信者を除外することにより '`To`'フィールドが空になる場合を除き、'`To`'フィールドからも除外されます。この変数が `nil`の場合、最初に返信を作成するときに、あなた自身のアドレスにマッチするデフォルト値に初期化されます。

元メッセージの送信者だけにリプライするには、`C-u r`または `1 r`のように、数引数とともに返信コマンドをエンターします。特定の返信にたいして '`CC`'フィールドを完全に省略するには、返信コマンドに数引数を指定します。これは、元のメッセージを送信した人だけに返信することを意味します。

1 度メール作成バッファーが初期化されると、後は通常どおりメールの編集と送信を行なうことができます (Chapter 29 [Sending Mail], page 420 を参照してください)。事前にセットされたヘッダーフィールドが適切でない場合は、それを編集することができます。`C-c C-y`のようなコマンドを使うこともできます。これは返信するメッセージを yank します (Section 29.4 [Mail Commands], page 423 を参照してください)。Rmail バッファーに切り替えて、異なるメッセージを選択してから、また戻って新しいカレントメッセージに yank することもできます。

メッセージが送信先に届かないこともあります。そのような場合メーラーは通常、失敗メッセージ *(failure message)* をあなたに返信します。Rmail コマンドの `M-m` (`rmail-retry-failure`) は、同じメッセージの 2 回目の送信を準備をします。これは前と同じテキストとヘッダーフィールドで、メール作成バッファーをセットアップします。そこですぐに `C-c C-c`をタイプすると、初回とまったく同じメッセージを再送します。テキストやヘッダーを編集してから送信することもできます。変数 `rmail-retry-ignored-headers`は、失敗したメッセージを再試行するとき除外するヘッダーを制御し、フォーマットは `rmail-ignored-headers` (Section 30.13 [Rmail Display], page 443 を参照してください) と同じです。

Rmail からメールを送信する他のよくある理由に、カレントメッセージを他のユーザーに転送 *(forward)* することです。`f` (`rmail-forward`) は、メール作成バッファーのテキストと subject を、カレントメッセージで事前に初期化することにより、これを簡単に行なえるようにします。subject は `[`*`from`*`:` *`subject`*`]`という形式で初期化されます。*from*と *subject*には、元のメッセージの送信者と subject が入ります。あなたが行なう必要があるのは、送信先を記述して、それを送信することだけです。メッセージを転送するとき、受信者が受け取るメッセージの from はあなたになり、メールの内容は元のメッセージと同じになります。

Rmail は転送メッセージにたいして 2 つのフォーマットを提供します。デフォルトは MIME フォーマットを使用します (Section 30.13 [Rmail Display], page 443 を参照してください)。これは元のメッセージを別の部分に含めます。変数 `rmail-enable-mime-composing`を `nil`にセットすることにより、もっと簡単なフォーマットを使うこともできます。この場合、Rmail は元のメッセージを 2 つ

の区切り行で囲むだけです。これは各行の行頭に '- ' を挿入することにより、各行の変更も行ないます。このフォーマットによる転送メッセージを受信した場合、それに普通のテキスト以外の何か — たとえばプログラムのソースコード — が含まれている場合、この変更を取り消せたら便利だと思うかもしれません。これを行なうには、転送されたメッセージを選択して、`M-x unforward-rmail-message` とタイプします。このコマンドは、挿入された文字列 '- ' を削除して、転送されたメッセージのオリジナルを抽出し、カレントメッセージの直後に、別のメッセージとして Rmail ファイルに挿入します。

再送 *(Resending)* は、転送と似た別の方法です。違いは、再送により送信されるメッセージは、あなたが受け取ったときのように、元の送信者が from になり、追加のヘッダーフィールド ('`Resent-From`' と '`Resent-To`') により、それがあなたを通じて送られたことを示すことです。Rmail でメッセージを再送するには、`C-u f`を使用します (`f`は`rmail-forward`を実行し、数引数を指定すると`rmail-resend`を呼び出します)。

`m` (`rmail-mail`) を使用することにより、返信ではない送信用のメールの編集を開始します。これはヘッダーフィールドを空のままにします。`C-x 4 m`との違いは、`r`のように `C-c C-y`で Rmail にアクセスできることです。

`c` (`rmail-continue`) コマンドは、既に編集を開始した送信用メッセージの編集を終えるために、または送信したメッセージを変更するために、メール作成バッファーでの編集を再開します。

変数 `rmail-mail-new-frame`を非 `nil`にセットした場合、メッセージの送信を開始するすべてのコマンドは、それを編集するために新しいフレームを作成します。このフレームは、そのメッセージを送信すると削除されます。

メッセージを送信するすべての Rmail コマンドは、選択されたメール作成方法を使用します (Section 29.7 [Mail Methods], page 427 を参照してください)。

30.11 サマリー

サマリー *(summary)* は、Rmail ファイルのメールを概観するために、メッセージごとに 1 つの行を含むバッファーです。各行にはメッセージ番号、日付、送信者、行数、ラベル、subject が表示されます。サマリーバッファーでポイントを移動することにより、そのサマリー行のメッセージを選択することができます。ほとんどの Rmail コマンドはサマリーバッファーでも有効です。それらのコマンドを使うと、サマリーのカレント行に記述されているメッセージに適用されます。

サマリーバッファーは、1 つの Rmail ファイルだけに適用されます。複数の Rmail ファイルを編集している場合、それぞれが自身のサマリーバッファーをもつことができます。サマリーバッファーの名前は、Rmail バッファーの名前に '`-summary`'を追加して作成されます。通常は 1 度に 1 つだけのサマリーバッファーが表示されます。

30.11.1 サマリーの作成

以下は、カレント Rmail バッファーでサマリーを作成するコマンドです。Rmail バッファーが 1 度サマリーされると、Rmail バッファーでの変更 (メッセージの削除や完全な削除、新しいメールの受信など) により、サマリーも自動的に更新されます。

`h`
`C-M-h`　　すべてのメッセージをサマリーします (`rmail-summary`)。

`l` *`labels`* `RET`
`C-M-l` *`labels`* `RET`
　　1 つ以上の指定したラベルをもつメッセージをサマリーします (`rmail-summary-by-labels`)。

C-M-r *rcpts* RET
: 指定した受信者にマッチするメッセージをサマリーします (rmail-summary-by-recipients)。

C-M-t *topic* RET
: 指定した正規表現 *topic*にマッチする subject をもつメッセージをサマリーします (rmail-summary-by-topic)。

C-M-s *regexp* RET
: 指定した正規表現 *regexp*にマッチするヘッダーをもつメッセージをサマリーします (rmail-summary-by-regexp)。

C-M-f *senders* RET
: 指定した送信者にマッチするメッセージをサマリーします (rmail-summary-by-senders)。

コマンド hまたは C-M-h (rmail-summary) は、カレント Rmail バッファーにたいする、すべてのメッセージのサマリーを、サマリーバッファーに表示します。その後、別のウィンドウにサマリーバッファーを表示して、それを選択します。

C-M-l *labels* RET (rmail-summary-by-labels) は、1 つ以上のラベル *labels*をもつメッセージの、部分的なサマリーを作成します。*labels*には、カンマで区切られたラベル名を指定します。

C-M-r *rcpts* RET (rmail-summary-by-recipients) は、正規表現 *rcpts*にマッチする、1 つ以上の受信者をもつメッセージのサマリーを作成します。これはヘッダー 'To'、'From'、'CC'にたいしてマッチを行ないます (プレフィクス引数を与えた場合は 'CC'ヘッダーを除外する)。

C-M-t *topic* RET (rmail-summary-by-topic) は、正規表現 *topic*にマッチする subject をもつメッセージの、部分的なサマリーを作成します。プレフィクス引数を指定した場合、subject だけでなく、メッセージ全体にたいしてマッチを行ないます。

C-M-s *regexp* RET (rmail-summary-by-regexp) は、正規表現 *regexp*にマッチするヘッダー (日付と subject 行を含む) をもつメッセージの、部分的なサマリーを作成します。

C-M-f *senders* RET (rmail-summary-by-senders) は、正規表現 *senders*にマッチする 'From'フィールドをもつメッセージの、部分的なサマリーを作成します。

1 つの Rmail バッファーにたいして、1 つのサマリーしか存在しないことに注意してください。他の種類のサマリーを作成すると、以前のサマリーは破棄されます。

変数 rmail-summary-window-sizeは、サマリーウィンドウに何行使用するかを指定します。変数 rmail-summary-line-count-flagは、メッセージのサマリー行に、メッセージの総行数を含めるかを制御します。このオプションに nilをセットすると、サマリーの生成が速くなるかもしれません。

30.11.2 サマリーでの編集

Rmail バッファーで行なえることのほとんどは、Rmail サマリーバッファーでも使用できます。実際、1 度サマリーバッファーを作成すれば、Rmail バッファーに戻る必要はありません。

サマリーバッファーで異なる行にポイントを移動するだけで、サマリーバッファーからメッセージを選択して、Rmail バッファーに表示することができます。ポイントを移動する Emacs コマンドが何であるかは問題になりません。コマンドの最後でポイントのある行のメッセージが、Rmail バッファーに表示されます。

ほとんどの Rmail コマンドは、Rmail バッファーと同様に機能します。したがって、サマリーバッファーでは、dがカレントメッセージの削除、uは削除の取り消し、xで完全に削除します (しかし、サ

マリーバッファーでは関連する方向に削除されていないメッセージが存在しない場合、削除コマンドはカレントメッセージに留まるのではなく、最初または最後のメッセージに移動します)。oと C-oは、カレントメッセージをファイルに出力します。他にも、rはそれにたいする返信を開始する、などです。サマリーバッファーで SPCと DELを使用することにより、カレントメッセージをスクロールできます。しかし、サマリーバッファーで SPCまたは DELにより、メッセージの終端または先頭を超えてスクロールすると、削除されていない次または前のメッセージに移動します。rmail-summary-scroll-between-messagesオプションを nilにカスタマイズすれば、次または前のメッセージへのスクロールが無効になります。

M-u (rmail-summary-undelete-many) は、サマリーで削除されたすべてのメッセージの削除を取り消します。プレフィクス引数を指定した場合、以前に削除された、指定した数のメッセージの削除を取り消すことを意味します。

メッセージ間を移動する Rmail コマンドはサマリーバッファーでも機能しますが、動作が少し異なります。これらのコマンドはサマリーに含まれる一連のメッセージ間を移動します。これらのコマンドは、常に Rmail バッファーがスクリーンに表示されるようにします (カーソル移動コマンドは Rmail バッファーの内容を更新しますが、これらのコマンドはウィンドウにすでにそれが表示されているのでなければ、表示しません)。以下はそれらのコマンドのリストです:

n
: “deleted” の行をスキップして次の行に移動し、その行のメッセージを選択します (rmail-summary-next-msg)。

p
: “deleted” の行をスキップして前の行に移動し、その行のメッセージを選択します (rmail-summary-previous-msg)。

M-n
: 次の行に移動して、その行のメッセージを選択します (rmail-summary-next-all)。

M-p
: 前の行に移動して、その行のメッセージを選択します (rmail-summary-previous-all)。

>
: 最後の行に移動して、その行のメッセージを選択します (rmail-summary-last-message)。

<
: 最初の行に移動して、その行のメッセージを選択します (rmail-summary-first-message)。

j
RET
: (Rmail バッファーがスクリーンに確実に表示されるようにして) カレント行のメッセージを選択します (rmail-summary-goto-msg)。引数 *n*を指定した場合、メッセージ番号 *n*のメッセージを選択し、サマリーバッファーのそのメッセージの行に移動します。そのメッセージがサマリーバッファーにリストされていない場合は、エラーをシグナルします。

M-s *pattern* RET
: メッセージから *pattern*を検索します。検索はカレントメッセージから開始されます。マッチが見つかったらそのメッセージを選択して、サマリーバッファーのそのメッセージの行にポイントを移動します (rmail-summary-search)。プレフィクス引数は繰り返し回数として機能します。負の引数は後方に検索を行なうことを意味します (rmail-summary-search-backwardと等価です)。

C-M-n *labels* RET
: 指定した 1 つ以上のラベルのうち、少なくとも 1 つをもつ次のメッセージに移動します (rmail-summary-next-labeled-message)。*labels*はカンマで区切られたラベルのリストです。プレフィクス引数は繰り返し回数として機能します。

C-M-p *labels* RET
: 指定した 1 つ以上のラベルのうち、少なくとも 1 つをもつ前のメッセージに移動します (rmail-summary-previous-labeled-message)。

C-c C-n RET
: カレントメッセージと同じ subject をもつ、次のメッセージに移動します (rmail-summary-next-same-subject)。プレフィクス引数は繰り返し回数として機能します。

C-c C-p RET
: カレントメッセージと同じ subject をもつ、前のメッセージに移動します (rmail-summary-previous-same-subject)。

削除、削除の取り消し、新しいメールの取得はもちろん、異なるメッセージの選択でも、それらの操作を Rmail バッファーで行なったとき、サマリーバッファーは更新されます。変数 rmail-redisplay-summary が非 nil の場合、これらの操作はサマリーバッファーをスクリーンに表示します。

サマリーの使用を終了するときは、Q (rmail-summary-wipe) とタイプして、サマリーバッファーのウィンドウを削除します。サマリーから Rmail を終了することもできます。q (rmail-summary-quit) はサマリーウィンドウを削除して、Rmail ファイルを保存してから Rmail を終了してから、他のバッファーに切り替えます。かわりに b (rmail-summary-bury) とタイプすると、単に Rmail と Rmail サマリーバッファーを隠し (bury) ます。

30.12 Rmail ファイルのソート

C-c C-s C-d
M-x rmail-sort-by-date
: カレント Rmail バッファーのメッセージを、日付順にソートします。

C-c C-s C-s
M-x rmail-sort-by-subject
: カレント Rmail バッファーのメッセージを、subject 順にソートします。

C-c C-s C-a
M-x rmail-sort-by-author
: カレント Rmail バッファーのメッセージを、送信者順にソートします。

C-c C-s C-r
M-x rmail-sort-by-recipient
: カレント Rmail バッファーのメッセージを、受信者の名前順にソートします。

C-c C-s C-c
M-x rmail-sort-by-correspondent
: カレント Rmail バッファーのメッセージを、他の受信者名順にソートします。

C-c C-s C-l
M-x rmail-sort-by-lines
: カレント Rmail バッファーのメッセージを、行数順にソートします。

C-c C-s C-k RET *labels* RET
M-x rmail-sort-by-labels RET *labels* RET
: カレント Rmail バッファーのメッセージを、ラベル順にソートします。引数 *labels*は、カンマで区切られたラベルのリストです。ラベルの順序は、メッセージの順序を指定し

ます。最初のラベルをもつメッセージが最初に、2 番目のラベルをもつメッセージが次に、というようになります。ラベルをもたないメッセージは最後になります。

Rmail のソートコマンドは安定ソート *(stable sort)* を行ないます。2 つのメッセージのどちらを先にするか特に理由がない場合、メッセージの順序は変更されません。これを使用して複数のソート条件を使用できます。たとえば、`rmail-sort-by-date`の後に `rmail-sort-by-author`を使用すれば、メッセージは作者ごとに日付順にソートされます。

プレフィクス引数を指定した場合、これらのコマンドは逆順で比較をします。これはメッセージが新しいものから古いものへ、大きいものから小さいものへ、アルファベットの逆順でソートされることを意味します。

同じキーをサマリーバッファーで使うと、似た関数が実行されます。たとえば `C-c C-s C-l`は、`rmail-summary-sort-by-lines`を実行します。これらのコマンドは、たとえサマリーがメッセージの一部しか表示していなくても、Rmail バッファー全体をソートします。

ソートのアンドゥはできないことに注意してください。そのため、ソートをする前に Rmail バッファーを保存したいと思うかもしれません。

30.13 メッセージの表示

このセクションでは Rmail が、メールヘッダー、MIMEのセクションと添付、URL、暗号化されたメッセージを表示する方法を説明します。

`t` ヘッダーの完全表示を切り替えます (`rmail-toggle-header`)。

各メッセージを最初に表示する前に、Rmail は余分な物を減らすために、重要でないヘッダーを隠して、メッセージのヘッダーを再フォーマットします。`t` (`rmail-toggle-header`) コマンドは、これを切り替えます。つまり再フォーマットされたヘッダーフィールドと、完全な元のヘッダーの間で、表示を切り替えます。正の引数を指定した場合、このコマンドは再フォーマットされたヘッダーを表示します。0 または負の引数を指定した場合、完全なヘッダーを表示します。メッセージを再選択することにより、必要な場合は再フォーマットします。

変数`rmail-ignored-headers`は、隠すべきヘッダーフィールドを指定する正規表現を保持します。これにマッチするヘッダー行は隠されます。変数`rmail-nonignored-headers`は、これをオーバーライドします。この変数の正規表現にマッチするヘッダーフィールドは、たとえそれが`rmail-ignored-headers`にマッチしても、表示されます。変数`rmail-displayed-headers`は、これら 2 つの変数のかわりに使用されます。非`nil`の場合、その値には表示するヘッダーを指定する正規表現を指定します (デフォルトは`nil`です)。

Rmail は特に重要なヘッダーフィールド — デフォルトでは ‘`From`’と ‘`Subject`’フィールドをハイライトします。ハイライトには `rmail-highlight`フェイスが使用されます。変数`rmail-highlighted-headers`は、ハイライトするヘッダーフィールドを指定する正規表現を保持します。これがヘッダーフィールドの先頭にマッチした場合、フィールド全体がハイライトされます。この機能を無効にするには、`rmail-highlighted-headers`に`nil`をセットしてください。

メッセージが MIME(Multipurpose Internet Mail Extensions) 形式で、複数パート (MIMEエンティティー) が含まれている場合、Rmail は各パートにタグライン *(tagline)* を表示します。タグラインはそのパートのインデックス、サイズ、コンテントタイプを要約します。コンテントタイプに依存して、1 つ以上のボタンが含まれる場合があります。これらのボタンは、そのパートをファイルに保存する、などの処理を行ないます。

`RET` ポイント位置の MIMEパートを隠す、または表示します (`rmail-mime-toggle-hidden`)。

`TAB` 次の MIMEタグラインのボタンにポイントを移動します (`rmail-mime-next-item`)。

`S-TAB` 前の MIMEパートにポイントを移動します (`rmail-mime-previous-item`)。

`v` MIME表示と raw メッセージの表示を切り替えます (`rmail-mime`)。

プレーンテキストの MIMEパートは、最初タグラインの直後に表示され、Rmail バッファーの他のタイプの MIMEパートは、Rmail バッファーの一部としてタグラインだけが表示され (メッセージが HTMLパートをもたない場合。以下参照)、実際のコンテンツは隠されています。どちらの場合も、MIMEパートのどこか、またはそれのタグラインで `RET`をタイプすることにより、表示と非表示を切り替えることができます (他の処理を行なうボタンがある場合を除きます)。`RET`とタイプするかマウスでクリックすることにより、タグラインボタンをアクティブにでき、`TAB`でタグラインのボタンにたいして循環的にポイントを移動できます。

`v` (`rmail-mime`) コマンドは、上記で説明したデフォルトの MIME表示と、MIMEでデコードされていない raw データの表示を切り替えます。プレフィクス引数を指定した場合は、ポイント位置にあるものの表示だけを切り替えます。

メッセージに HTMLの MIMEパートがあり、Emacs が HTMLを表示できる場合、Rmail はそれを plain-text パートより優先して表示します [1]。これを抑制してかわりに plain-text パートを表示するには、変数 `rmail-mime-prefer-html`を `nil`にカスタマイズしてください。

Rmail から MIME でデコードされたメッセージの処理を抑止するには、変数 `rmail-enable-mime`を `nil`に変更します。この場合、`v` (`rmail-mime`) は、カレント MIMEメッセージを表示するために、一時的なバッファーを作成します。

カレントメッセージが暗号化されている場合、復号化するためにコマンド `C-c C-d` (`rmail-epa-decrypt`) を使用します。これは EasyPG ライブラリーを使用します (Section “EasyPG” in *EasyPG Assistant User's Manual* を参照)。

Rmail バッファーで Goto Address モードを使用して、URL のハイライトとアクティブ化ができます:

```
(add-hook 'rmail-show-message-hook 'goto-address-mode)
```

このモードを使用すると、その URL を `mouse-2`でクリック (または `mouse-1`で素早くクリック) するか、ポイントをそこに移動して `C-c RET`とタイプすることにより、それらの URL をブラウズできます。Section 31.12.4 [Activating URLs], page 486 を参照してください。

30.14 Rmail とコーディングシステム

Rmail は、Emacs がファイルを visit したりサブプロセスの出力にたいして行なうように、非 ASCII 文字を含むメッセージを自動的にデコードします。Rmail はメッセージで標準の ‘`charset=`*`charset`*’ ヘッダーを使用し、もしそれがあれば、送信者によりメッセージがどのようにエンコードされたか決定します。これは *charset*を、対応する Emacs コーディングシステム (Section 19.5 [Coding Systems], page 223 を参照してください) にマップして、メッセージテキストをデコードするために、そのコーディングシステムを使います。メッセージヘッダーに ‘`charset`’指定がない場合、または *charset*が認識されなかった場合、Rmail は通常の Emacs の経験則とデフォルトに則ったコーディングシステムを選択します (Section 19.6 [Recognize Coding], page 225 を参照してください)。

メッセージが間違ってデコードされることもあります。これは ‘`charset`’指定がないために Emacs が間違ったコーディングシステムを推測したか、そもそも指定が間違っているかです。たと

[1] この機能は、Emacs が `libxml2`サポートつきでビルドされているか、Lynx ブラウザーがインストールされている必要があります。

えば間違って設定されたメーラーが、メッセージが実際には koi8-rでエンコードされているのに、‘charset=iso-8859-1’というヘッダーでメッセージを送るかもしれません。メッセージテキストが文字化けしていたり、文字が 16 進コードや空ボックスで表示されているときは、おそらくこれが発生しています。

正しいコーディングシステムを解決または推測できる場合、正しいコーディングシステムを使ってメッセージを再デコードすることにより、問題を訂正することができます。これを行なうには M-x rmail-redecode-bodyコマンドを呼び出します。これはコーディングシステムの名前を読み取り、指定したコーディングシステムを使って、メッセージを再デコードします。正しいコーディングシステムを指定した場合、デコード結果は読めるようになるでしょう。

Rmail で新しいメールを受信したとき、各メッセージは、それらがあたかも個別のファイルであるかのように、それぞれが記述されたコーディングシステムに自動的に変換されます。これは指定されたコーディングシステムの優先順を使用します。MIME メッセージが文字セットを指定している場合、Rmail はその指定にしたがいます。Rmail ファイルの読み込みと保存にたいして、Emacs は、変数 rmail-file-coding-systemで指定されたコーディングシステムを使用します。デフォルト値は nilで、これは Rmail ファイルが変換されないことを意味します (これらは Emacs の内部文字セットで読み書きされます)。

30.15 メッセージの編集

通常の Emacs のキーバインドのほとんどは Rmail モードで利用可能ですが、C-M-nや C-M-hのように、他の目的のために Rmail により再定義されているものもあります。しかし Rmail バッファーは通常読み取り専用で、ほとんどの文字は Rmail コマンドに再定義されています。メッセージのテキストを編集したい場合、Rmail の eコマンドを使わなければなりません。

e　　カレントメッセージを通常のテキストとして編集します。

e command (rmail-edit-current-message) は、Rmail モードから、Rmail Edit モードという、Text モードと類似した、別のメジャーモードに切り替えます。メジャーモードの変更はモードラインに示されます。

Rmail Edit モードでは、文字は通常どおり文字自身を挿入し、Rmail コマンドは利用できません。メッセージの本文とヘッダーフィールドを編集することができます。メッセージの編集を終えたら、C-c C-c (rmail-cease-edit) で Rmail モードに戻ります。かわりに C-c C-] (rmail-abort-edit) とタイプすれば、編集をキャンセルして Rmail モードに戻ることができます。

Rmail Edit モードに入ることにより、フック text-mode-hook、その後にフック rmail-edit-mode-hookが実行されます (Section 33.2.2 [Hooks], page 509 を参照してください)。通常の Rmail モードにもどると、メッセージを変更した場合には、そのメッセージに属性‘edited’が追加されます (Section 30.9 [Rmail Attributes], page 437 を参照)。

30.16 ダイジェストメッセージ

ダイジェストメッセージ *(digest message)* は、複数の他のメッセージを含み、それを運ぶために存在するメッセージです。ダイジェストは、いくつかのメーリングリストで使用されています。1 日というような一定の期間の間にメーリングリストに到着したすべてのメッセージが、1 つのダイジェストにまとめられて、メーリングリストに登録した人に送られます。1 つのダイジェストを送信するのにかかるコンピューター時間は、たとえ合計サイズが同じでも個別にメッセージを送信するより短くなります。なぜならネットワークでのメール送信において、メッセージ単位のオーバーヘッドがあるからです。

ダイジェストメッセージを受信したとき、それを読むもっとも便利な方法は、それを非ダイジェスト化 *(undigestify)* することです。これはダイジェストを複数のメッセージに戻します。それから個別にメッセージを読んだり削除できます。これを行なうにはダイジェストメッセージを選択して、コマンド `M-x undigestify-rmail-message`をタイプします。これはダイジェストに含まれるメッセージを個別の Rmail メッセージに抽出し、ダイジェストの後に挿入します。ダイジェストメッセージ自身には、削除のフラグがつけられます。

30.17 Rot13 メッセージを読む

読む人を怒らせたり不快にするかもしれないメーリングリストのメッセージは、*rot13* と呼ばれる単純なコードでエンコードされているときがあります。この名前はエンコードの方法がアルファベットを 13 文字分巡回させることに由来します。このコードに機密性はなく、提供もしていません。むし、実際のテキストを見るのを避けたいと思う人のためのものです。たとえばビデオの講評などでは重要なあらすじを隠す (訳注: いわゆるネタバレ回避) ために rot13 を使います。

rot13 を使ったバッファーを閲覧するには、コマンド `M-x rot13-other-window`を使用します。これはカレントバッファーを他のウィンドウで表示します。このウィンドウではテキストを表示するときこのコードを適用します。

関心があるのはリージョンだけであれば、コマンド `M-x rot13-region`を使うほうがよいかもしれません。これはアクティブなリージョンをそのまま暗号化/復号化します。読み取り専用のバッファーでは、エコーエリアに平文テキストの表示を試みます。それがエコーエリアにたいして長すぎるテキストであれば、このコマンドは暗号化/復号化されたテキストを表示する一時的なバッファーをポップアップします。

30.18 `movemail` program

Rmail は、inbox から Rmail ファイルにメールを移動するために、`movemail`プログラムを使用します。最初にロードされたとき、Rmail は `movemail`プログラムを探して、そのバージョンを判断します。`movemail`プログラムには 2 つのバージョンがあります。それは GNU Mailutils バージョン (Section “movemail” in *GNU mailutils* を参照) と、`--with-mailutils`で Emacs を configure したときにインストールされる Emacs 固有バージョンです。これらのコマンドは、同じコマンドラインシンタックスをもち、同じ基本的なサブセットオプションをもちます。しかし Mailutils バージョンは、追加の機能を提供します。

Emacs バージョンの `movemail`は、通常の Unix mailbox 形式の mailbox からメールを取得することができます。警告: これは POP3 プロトコルの使用も可能ですが、暗号化された TLS チャンネルを通じた POP3 をサポートしないので、推奨しません。

Mailutils バージョンは、プレーン Unix mailbox、`maildir`および `MH`のメールボックスなどの、より広範な mailbox 形式を処理することができます。これは POP3 または IMAP4 プロトコルを使用してリモートの mailbox にアクセスでき、TLS 暗号化チャンネル (TLS encrypted channel) を使用してメールを取得できます。これは URL形式での mailbox 引数を受けとることもできます。mailboix URLの詳細な説明は、Section “URL” in *Mailbox URL Formats* で見ることができます。短く言うと、URLは以下のようなものです:

```
proto://[user[:password]@]host-or-file-name[:port]
```

角カッコ (bracket) はオプションの要素を意味します。

proto *mailbox* プロトコル、または使用するフォーマットを指定します。URLの残りの要素の正確な意味は、*proto*の実際の値に依存します (以下参照)。

user リモート mailbox にアクセスするためのユーザー名です。

password リモート mailbox にアクセスするためのユーザーパスワードです。

host-or-file-name
リモート mailbox のリモートサーバーのホスト名、またはローカル mailbox のファイル名です。

port そのプロトコルのデフォルト以外のポート番号をオプションで指定します。

*proto*には以下の 1 つを指定します:

`mbox` 通常の Unix mailbox 形式です。この場合 *user*、*pass*、*port*は使用せず、*host-or-file-name*は mailbox ファイルのファイル名を意味します (例: `mbox:///var/spool/mail/smith`)。

`mh` MH形式のローカル mailbox です。*user*、*pass*、*port*は使用せず、*host-or-file-name* は MHフォルダーのファイル名を意味します (例: `mh:///Mail/inbox`)。

`maildir` maildir形式のローカル mailbox です。*user*、*pass*、*port*は使用せず、*host-or-file-name* は `maildir` mailbox の名前を意味します (例: `maildir:///mail/inbox`)。

`file` mailbox 形式の任意のローカルファイルです。実際の形式は `movemail`により自動的に決定されます。

`pop`
`pops` POP3 プロトコルを通じてアクセスされるリモート mailbox です。*user*は使用するリモートのユーザー名を指定し、*pass*はユーザーパスワードを指定するのに使用され、*host-or-file-name*は接続するリモートメールサーバーのホスト名か IP アドレス、*port*はポート番号です (例: `pop://smith:guessme@remote.server.net:995`)。サーバーがサポートする場合、`movemail`は暗号化された接続— 暗号化接続に要求される '`pops`'の使用を試みます。

`imap`
`imaps` IMAP4 プロトコルを通じてアクセスされるリモート mailbox です。*user*は使用するリモートのユーザー名を指定し、*pass*はユーザーパスワードを指定するのに使用され、*host-or-file-name*は接続するリモートメールサーバーのホスト名か IP アドレス、*port* はポート番号です (例: `imap://smith:guessme@remote.server.net:993`)。サーバーがサポートする場合、`movemail`は暗号化された接続— 暗号化接続に要求される '`imaps`'の使用を試みます。"

かわりに、使用する mailbox のファイル名を指定できます。これはプロトコルに '`file`'を指定するのと等価です:

`/var/spool/mail/`*user* ≡ `file:///var/spool/mail/`*user*

変数 `rmail-movemail-program`は、どのバージョンの `movemail`を使用するかを制御します。文字列の場合、それは `movemail`実行ファイルの絶対ファイル名を指定します。`nil`の場合、Rmail は `rmail-movemail-search-path`、`exec-path`(Section 31.5 [Shell], page 457 を参照してください)、`exec-directory`の順で、これらの変数にリストされたディレクトリーから、`movemail`を検索します。

30.19 リモート mailbox からのメールの取得

inbox ファイルにデータを格納するかわりに、POP3 と呼ばれる手法を使用してユーザーの inbox データにアクセスするサイトがいくつかあります。Mailutils の `movemail`は、デフォルトで TLS 暗号化された POP3 をサポートします。警告**:** たとえ `Emacs movemail`が POP3 をサポートしていて

も、Mailutils バージョンがサポートする暗号化された接続をサポートしないので、これを使用することは推奨しません。どちらのバージョンの `movemail`も、POP3 にたいしてだけ機能し、POP の古いバージョンにたいしては機能しません。

どちらの `movemail`を使用するかにかかわらず、POP3 *URL*(Section 30.18 [Movemail], page 446 を参照) を使用して POP3 inbox を指定できます。POP3 URLは、'`pop://`*username*`@`*hostname*`:`*port*'という形式で、*hostname*と *port*はリモートメールサーバーのホスト名 (または IP アドレス) とポート番号、*username*はそのサーバーでのユーザー名です。これに加えて '`pop://`*username*`:`*password*`@`*hostname*`:`*port*'のような mailbox URLでパスワードを指定することもできます。この場合、*password*は `rmail-remote-password` (以下参照) で指定された値より優先されます。これは複数のリモートメールサーバーで異なるパスワードを指定するとき、特に便利です。

後方互換のため、Rmail はリモートの POP3 mailbox を指定する他の方法もサポートします。'`po:`*username*`:`*hostname*`:`*port*'による inbox 名の指定は、'`pop://`*username*`@`*hostname*`:`*port*'と等価です。*:hostname*の部分を省略した場合は、環境変数 `MAILHOST`で、どのマシンの POP3 サーバーを探すか指定します。

リモート mailboxes にアクセスする他の方法に、IMAP があります。この方法は Mailutils `movemail`だけでサポートされます。inbox リストで IMAP mailbox を指定するには、'`imap://`*username*`[:`*password*`]@`*hostname*`:`*port*'の形式の mailbox URLを使用します。上記で説明したように、*password*の部分はオプションです。'`imap`'の箇所に '`imaps`'を使用したいと思うかもしれません。

リモート mailbox へのアクセスにはパスワードが要求されます。これを取得するために Rmail は以下のアルゴリズムを使います:

1. mailbox URL(上記参照) で *password*が与えられた場合はそれを使います。
2. 変数 `rmail-remote-password-required`が `nil`の場合、Rmail はパスワードが要求されないと想定します。
3. 変数 `rmail-remote-password`が非 `nil`の場合はその値を使います。
4. 上記以外の場合、Rmail はパスワードの入力を求めます。

ユーザー名にドメイン情報が含まれるメールサーバーでは、ユーザー名に '`@`'文字が含まれる可能性があります。inbox 指定子文字列はメールサーバー名の開始シグナルに '`@`'を使用します。これは movemail に混乱を生みます。ユーザー名が '`@`'を含み、`movemail`の`@`を使用している場合には、ユーザー名の '`@`'を URLエンコードの '`%40`'で置き換えることにより問題を解決できます。

追加のコマンドラインフラグを `movemail`に渡す必要がある場合は、使いたいフラグのリストを変数 `rmail-movemail-flags`にセットします。inbox の内容を保持するために。この変数を使ってフラグ '`-p`'を渡さないでください。かわりに `rmail-preserve-inbox`を使用してください。

あなたのサイトにインストールされた `movemail`プログラムは、ケルベロス認証 (Kerberos authentication) をサポートするでしょう。もしサポートされている場合、`rmail-remote-password`および `rmail-remote-password-required`がセットされていないときに、POP3 メールの取得を試みたときは、デフォルトでケルベロス認証を使います。

メッセージを逆順に保存する POP3 サーバーもあります。あなたのサーバーがこれを行なっている場合、到着した順にメールを読みたいときは、`rmail-movemail-flags`に '`-r`'フラグを追加することにより、逆順でメッセージをダウンロードするよう、`movemail`に指示できます。

Mailutils `movemail`は、TLS 暗号化 (TLS encryption) をサポートします。これを使いたい場合は、`rmail-movemail-flags`に '`--tls`'フラグをセットしてください。

30.20 さまざまな形式のローカル mailbox からのメールの取得

受信したメールがローカルマシンの Unix mailbox 以外の形式に保存される場合、これを取得するために Mailutils `movemail`を使う必要があるでしょう。`movemail`のバージョンについての詳細な説明は、Section 30.18 [Movemail], page 446 を参照してください。たとえば`/var/spool/mail/in`にある `maildir`形式の inbox のメールにアクセスするには、Rmail の inbox リストに以下を含める必要があるでしょう:

```
maildir:///var/spool/mail/in
```

31 その他のコマンド

このチャプターには、他のどこにも適さないような、複数のトピックについての概略が含まれています。それらには以下のものが含まれます: Usenet ニュースの購読、ホストとネットワークのセキュリティー、PDF および類似のドキュメントの閲覧、ウェブのブラウジング、シェルコマンドおよびシェルサブプロセスの実行、エディターをサブプロセスとして実行するユーティリティーとして 1 つの共有 Emacs を使用する、印刷、テキストのソート、バイナリーファイルの編集、後で再開するために Emacs セッションを保存する、再帰編集レベル、ハイパーリンクをフォローする、およびさまざまな気晴らしと娯楽、などです。

31.1 Gnus による E メールと Usenet ニュース

Gnus は、主に Usenet ニュースを読んだりポストするためにデザインされた、Emacs パッケージです。これはいくつかの異なるソース — 電子メール、リモートディレクトリー、ダイジェスト、などを読んだり、メッセージを返すためにも使うことができます。以下は Gnus の紹介と、いくつかの基本的な機能の説明です。Gnus についての完全な詳細は、`C-h i`とタイプしてから、Gnus manual を選択してください。

31.1.1 Gnus バッファー

Gnus は、情報を表示したり返信コマンドのために、複数のバッファーを使用します。もっとも一般的に使用される 3 つの Gnus バッファーはグループバッファー *(group buffer)*、サマリーバッファー *(summary buffer)*、アーティクルバッファー *(article buffer)* です。

グループバッファーは、アーティクルソースのリスト (たとえばニュースグループや電子メールの inbox) を含んでおり、それらはグループとして参照されます。これは Gnus を開始したときに最初に表示されるバッファーです。これは通常、あなたが登録したグループと、未読のアーティクルだけを表示します。このバッファーから、読みたいグループを選択できます。

サマリーバッファーは 1 つのグループのアーティクルをリストし、1 行に 1 つのアーティクルを表示します。デフォルトでは、アーティクルの作者、subject、行数が表示されます。サマリーバッファーは、グループバッファーでグループを選択すると作成され、グループを抜けると kill されます。

サマリーバッファーから、閲覧するアーティクルを選択できます。アーティクルはアーティクルバッファーで表示されます。通常の Gnus の使い方では、このバッファーを閲覧はしますが選択はしません— すべての便利な Gnus コマンドはサマリーバッファーから呼び出すことができます。しかし望むなら、アーティクルバッファーを選択して、そこから Gnus コマンドを実行することもできます。

31.1.2 Gnus を起動したとき

あなたのシステムが Usenet ニュースをよむためにセットアップされていれば、Gnus を始めるのは簡単です — `M-x gnus`とタイプするだけです。

起動時に、Gnus はホームディレクトリーにある`.newsrc`という名前のニュース初期化ファイル *(news initialization file)* を読み込みます。これにはあなたの Usenet ニュースグループと購読状況がリストされています (これは Gnus 固有のファイルではありません。他の多くのニュースリーダープログラムにより使用されています)。その後システムのデフォルトのニュースサーバーへの接続を試みます。これは通常、環境変数 `NNTPSERVER`により指定されます。

あなたのシステムがデフォルトのニュースサーバーをもっていない場合、または電子メールを読むために Gnus を使いたい場合は、`M-x gnus`を呼び出す前に、どこでニュースおよび/またはメールを取得するか、Gnus に指示する必要があります。これを行なうには、変数 `gnus-select-method`お

よび/または gnus-secondary-select-methodsをカスタマイズします。詳細は、Gnus のマニュアルを参照してください。

1 度 Gnus を開始すると、グループバッファーを表示します。デフォルトでは少数の *subscribed* グループ *(subscribed groups:* 登録されたグループ*)* だけが表示されます。他の状態 — *unsubscribed*、*killed*、*zombie* — のグループは表示されません。最初に Gnus を開始したとき、登録していないグループは killed グループになります。その後にニュースサーバーに現れたグループは zombie グループになります。

先に進むには、グループバッファーでグループを選択して、そのグループのサマリーバッファーを開かなければなりません。その後サマリーバッファーのアーティクルを選択して、別のウィンドウでアーティクルバッファーを閲覧します。以下のセクションでは、これを行なうための、グループバッファーとサマリーバッファーの使用について説明します。

Gnus を終了するには、グループバッファーで qとタイプします。これは自動的にグループの状態をファイル.newsrcト.newsrc.eldに記録するので、その後の Gnus セッションでも効果があります。

31.1.3 Gnus Group バッファーの使用

以下のコマンドは、Gnus グループバッファーで利用可能です:

SPC　カレント行のグループの、サマリーバッファーに切り替えます (gnus-group-read-group)。

l
A s　e グループバッファーでは、未読のアーティクルを含む登録したグループだけをリストします (gnus-group-list-groups。これはデフォルトの一覧方法である)。

L
A u　すべての subscribed(登録) および unsubscribed(未登録) のグループをリストしますが、killed または zombie のグループは表示しません (gnus-group-list-all-groups)。

A k　kill されたグループをリストします (gnus-group-list-killed)。

A z　ゾンビ状態のグループをリストします (gnus-group-list-zombies)。

u　グループの登録状態を切り替えます (gnus-group-toggle-subscription-at-point)。killed または zombie のグループにたいしてこれを呼び出すと、そのグループを unsubscribed グループにします。

C-k　カレント行のグループを kill します (gnus-group-kill-group)。killed となったグループは.newsrcファイルに記録され、lまたは Lのリストには表示されなくなります。

DEL　未読アーティクルを含む、前のグループにポイントを移動します (gnus-group-prev-unread-group)。

n　次の未読グループにポイントを移動します (gnus-group-next-unread-group)。

p　前の未読グループにポイントを移動します (gnus-group-prev-unread-group)。

q　Gnus のセッティングを更新して Gnus を終了します (gnus-group-exit)。

31.1.4 Gnus Summary バッファーの使用

以下のコマンドは、Gnus サマリーバッファーで利用可能です:

SPC　選択されたアーティクルがない場合、カレント行のアーティクルを選択して、それをアーティクルバッファーに表示します。そうでない場合、選択されたアーティクルバッファー

のウィンドウでスクロールを試みます。バッファーの最後に到達した場合、次の未読アーティクルを選択します (gnus-summary-next-page)。
したがって、繰り返し SPCをタイプすることにより、すべてのアーティクルを読むことができます。

DEL アーティクルのテキストを後方にスクロールします (gnus-summary-prev-page)。

n 次の未読アーティクルを選択します (gnus-summary-next-unread-article)。

p 前の未読アーティクルを選択します (gnus-summary-prev-unread-article)。

s 選択されたアーティクルバッファーで、あたかもそのバッファーに切り替えて C-s(Section 12.1 [Incremental Search], page 105 を参照してください) とタイプしたかのように、インクリメンタル検索を行ないます (gnus-summary-isearch-article)。

M-s M-s *regexp* RET
*regexp*へのマッチを含むアーティクルを、前方に検索します (gnus-summary-search-article-forward)。

M-r *regexp* RET
*regexp*へのマッチを含むアーティクルを、後方に検索します (gnus-summary-search-article-backward)。

q サマリーバッファーを exit して、グループバッファーに戻ります (gnus-summary-exit)。

31.2 ホストのセキュリティ

Emacs は GNU/Linux のようなオペレーティングシステムの内部で実行され、ファイルアクセスのようなセキュリティー制限のチェックはオペレーティングシステムに依存します。Emacs のデフォルトセッティングは、通常の使用のためにデザインされています。通常よりセキュリティーが重要な場合、または重要でない環境では、何らかの調整が必要になるでしょう。たとえばファイルローカル変数 (file-local variables) が危険な場合もあるので、変数 enable-local-variablesに:safeや、(より安全に)nilをセットできます。ファイルがすべて信用でき、それらの変数にたいするデフォルトのチェックが煩わしいときには、enable-local-variablesに:allをセットできます。Section 33.2.4.2 [Safe File Variables], page 515 を参照してください。

大きなアプリケーションの一部として Emacs を使用する場合の、セキュリティー考慮についての情報は、Section “Security Considerations” in *The Emacs Lisp Reference Manual* を参照してください。

31.3 ネットワークのセキュリティ

Emacs がネットワーク接続を確立するときは、常にその確立された接続を NSM(*Network Security Manager*) に渡します。NSMは、あなたのコントロールのもとにセットワークセキュリティーを実施する責任があります。現在のところ、これは TLS(Transport Layer Security) の機能を使用して動作します。

変数 network-security-levelは NSMが実施するセキュリティーレベルを決定します。変数の値が lowならセキュリティーのチェックは行なわれません。これは必須ではなく。基本的にはそのネットワーク接続が信頼できないことを意味しています。しかしネットワークの問題テスト時のように制限された状況ではこのセッティングは便利かもしれません。

この変数が `medium`(デフォルト) の場合、いくつかのチェックが行なわれます。チェックの結果、NSMがそのネットワーク接続を信頼できないと判断した場合は、それを知らせて、そのネットワーク接続にたいして何を行なうか尋ねます。

証明されていない接続 (unverified connection) にたいして、永続的なセキュリティー例外 (security exception) を登録したり、一時的な例外 (temporary exception) とするか、接続を完全に拒絶することができます。

証明書の基本的な正当性チェック加えて、いくつかの TLSアルゴリズムによるチェックが利用可能です。以前は安全だと思われていたいくつかの暗号化技術で脆弱性が判明しているものについては、(デフォルトでは)Emacs は問題に関して警告します。

プロトコルネットワークのチェックは `network-security-protocol-checks`変数を介して制御されます。

これは各連想値の最初の要素がチェックの名前、2 つ目の要素がチェックの使用を要するセキュリティレベルであるような alist です。

関数 `nsm-protocol-check--rc4`の結果内の `(rc4 medium)`のような要素は `(nsm-protocol-check--rc4 host port status settings)` のように呼び出します。この関数は接続を継続するなら非 `nil`、それ以外は非 `nil`をリターンします。

以下はデフォルトの `medium`で行なわれるチェックのリストです。

TLS認証が検証できない (unable to verify a TLS certificate)
: その接続が TLS、SSL、STARTTLS接続の場合、NSMは接続しようとしているサーバーの同一性 (identity) を証明するために使用される認証が、検証できるかどうかチェックします。

  無効な認証が懸念される場合 (Man-in-the-Middle によりネットワーク接続がハイジャックて、あなたのパスワードが盗まれるかもしれません) や、とにかく接続を行なう正当な理由があるかもしれません。たとえばサーバーが自己署名された認証 (self-signed certificate) を使用していたり、認証の期限が切れている場合もあるでしょう。接続の継続を容認するかどうかの決定は、あなたに任されています。

自己署名された認証が変更されている (a self-signed certificate has changed)
: 以前自己署名された認証を許容したが、今回はそれが変更されていて、それはそのサーバーが単に認証を変更しただけかもしれませんが、ネットワーク接続がハイジャックされている可能性もあります。

以前は暗号化されていた接続が、暗号化されていない (previously encrypted connection now unencrypted)
: 接続が暗号化されていないが、以前のセッションでは暗号化されていた場合、あなたとそのサーバーの間に STARTTLSアナウンス (STARTTLS announcements) を剥奪して、接続を非暗号化するプロキシーがあることを意味するかもしれません。これは通常とても疑わしいです。

パスワードを送信するとき暗号化されていないサービスと通信する (talking to an unencrypted service when sending a password)
: IMAPや POP3のサーバーに接続するとき、ネットワーク越しにパスワードを送信するのが一般的なので、接続は通常暗号化されています。同様に、パスワードを要求する SMTP を通じて email を送信する場合は通常、その接続が暗号化されていることを望むでしょう。その接続が暗号化されていない場合、NSMは警告します。

Diffie-Hellman ロープライムビット (Diffie-Hellman low prime bits)
: パグリックキーの交換を行なう際、そのチャンネルが第三者により盗聴できないことを確実にするために、プライムビット (prime bits) の数は十分に高くあるべきです。この数があまりに低い (low) 場合には Emacs は警告を発するでしょう (これは `network-security-protocol-checks`の `diffie-hellman-prime-bits` チェック)。

RC4ストリーム暗号 (RC4 stream cipher)
: RC4ストリーム暗号は低品質とされており、第三者による盗聴を許すかもしれません (これは `network-security-protocol-checks`の `rc4`チェック)。

ホスト証明書や中間証明書の SHA1
: 中間証明書 (intermediate certificate) が SHA1ハッシュアルゴリズムを使用していれば、第三者がその発行インスタンスを偽装して証明書を発行できると考えられています。したがってこれらの接続は中間者攻撃にたいして脆弱です (これらは `network-security-protocol-checks`の `signature-sha1`と `intermediate-sha1`チェック)。

SSL1、SSL2、SSL3
: TLS1.0より古いプロトコルは、様々な攻撃にたいして脆弱とされており、あなたが行なうことがより高いセキュリティーを要する場合は、使用を避けたいと思うかもしれません (これは `network-security-protocol-checks`の `ssl`チェック)。

`network-security-level`が `high`の場合、上記に加えて以下のチェックが行なわれます:

3DES暗号
: 3DESストリーム暗号は最大 112 ビットの効果的なセキュリティーを提供してローエンド向きと考えられています (これは `network-security-protocol-checks`の `3des` チェック)。

有効な認証がパブリックキーを変更した (a validated certificate changes the public key)
: サーバーはキーを変更するときがあり、通常それは心配することはありません。しかし、サードパーティーのサービスにたいして新しい認証を発行する、偽装しやすい証明期間 (pliable Certificate Authorities) へのアクセスをもつエージェントにより、ネットワーク接続がハイジャックされているか心配なときは、これらの変更を追跡したいと思うかもしれません。

最後に、`network-security-level`が `paranoid`の場合は、最初に NSMが新たな認証に遭遇したときに、それが通知されます。これにより Emacs によるすべての接続のすべての認証を検証することができるでしょう。

以下の追加の変数は、NSM操作の詳細を制御するために使用できます。

`nsm-settings-file`
: これは NSMが接続の詳細を保存するファイルです。デフォルトは `~/.emacs.d/network-security.data`です。

`nsm-save-host-names`
: デフォルトでは、非 `STARTTLS`接続ではホスト名は保存されません。接続の識別には、かわりにホストとポートによるハッシュ値 (host/port hash) が使用されます。ユーザーがどのサーバーに接続しているか確認するために、誰かが気軽に設定ファイルを読めないことを意味します。この変数が `t`の場合、NSMは `nsm-settings-file`にもホスト名を保存します。

31.4 ドキュメントの閲覧

DocView モードは、DVI、PostScript(PS)、PDF、OpenDocument、Microsoft Office、EPUB、CBZ、FB2、XPS、OXPS といったドキュメントを閲覧するためのメジャーモードです。このモードはスライス、ズーム、ドキュメント内の検索などの機能を提供します。これは、`gs` (GhostScript)、`pdfdraw`/`mutool draw` (MuPDF) といったコマンド、およびその他の外部ツールを使用してドキュメントを一連のイメージに変換、それらのイメージを表示することにより機能します。

DocView モードで表示可能なドキュメントを visit すると、Emacs は自動的にそのモードを使用します[1]。例外として、PostScript ファイルを visit したとき、Emacs は PostScript ファイルをテキストとして編集するためのメジャーモードの、PS モードに切り替わります。しかし、これは DocView minor モードも有効にするので、`C-c C-c`とタイプして、そのドキュメントを閲覧することができます。DocView モードまたは DocView minor モードでは、`C-c C-c` (`doc-view-toggle-display`) を繰り返すことにより、DocView とその背後にあるファイル内容を切り替えることができます。

いくつかの要件が満たされないとき (たとえばテキスト端末のフレームを操作していたり、その Emacs は PNG をサポートしないときなど) に、通常 DocView モードで処理されるファイルを visit した場合は、そのドキュメントの内容をプレーンテキストとして閲覧したいか問い合わせます。これに同意すると、そのバッファーは text モードとなり、DocView minor モードがアクティブになります。したがって`C-c C-c`とタイプすることにより、fallback モードに切り替わります。もう 1 度`C-c C-c`とタイプすると、DocView モードに戻ります。DocView モードで`C-c C-t` (`doc-view-open-text`) とタイプすることにより、プレーンテキストで内容を表示することもできます。

コマンド`M-x doc-view-mode`で、DocView モードを明示的に有効にすることができます。また、`M-x doc-view-minor-mode`で、DocView minor モードに切り替えることができます。

DocView モードを開始したときは、ウェルカム画面を表示して、そのファイルを 1 ページずつフォーマットしていきます。最初のページがフォーマットされると、そのページを表示します。

DocView バッファーを kill するには、`k` (`doc-view-kill-proc-and-buffer`) とタイプします。バッファーを隠す (bury) には、`q` (`quit-window`) とタイプします。

31.4.1 DocView の操作

DocView モードでは通常の Emacs 移動キー、つまり`C-p`、`C-n`、`C-b`、`C-f`、および矢印キーを使って、ページをスクロールできます。

デフォルトでは、行移動キーの`C-p`と`C-n`は、カレントページの先頭または最後でスクロールを止めます。しかし、変数`doc-view-continuous`を非`nil`値に変更した場合、カレントページの先頭で`C-p`とタイプすると前のページを表示し、カレントページの最後で`C-n`とタイプすると次のページを表示します。

`n`、`PageDown`、`next`、`C-x ]`をタイプすることにより、次のページを表示することもできます (`doc-view-next-page`)。前のページを表示するには、`p`、`PageUp`、`prior`、`C-x [`をタイプします (`doc-view-previous-page`)。

`SPC` (`doc-view-scroll-up-or-next-page`) は、ドキュメントを順に読んでいくのに便利な方法です。これはカレントページをスクロールするか、次のページに移動します。`DEL` (`doc-view-scroll-down-or-previous-page`) は、同様の方法で後方に移動します。

最初のページに移動するには、`M-<` (`doc-view-first-page`) とタイプします。最後のページに移動するには、`M->` (`doc-view-last-page`) とタイプします。ページ番号を指定して移動するには、`M-g M-g`または`M-g g` (`doc-view-goto-page`) とタイプしてください。

[1] そのドキュメントに必要な外部ツールが利用可能でなければならず、Emacs がグラフィカルなフレームで実行されていて、PNG イメージをサポートしなければなりません。これらの条件が満たされなければ、Emacs は他のメジャーモードにフォールバックします。

+ (doc-view-enlarge) と- (doc-view-shrink) で、ドキュメントを拡大したり縮小することができます。これらのコマンドはデフォルトではすでに描画済みのイメージを再スケールします。かわりにイメージを新たなサイズに再描画するには doc-view-scale-internallyを nilにセットしてください。DocView にたいするデフォルトサイズを指定するには、変数 doc-view-resolution をカスタマイズしてください。

DocView は imenu機能 (Section 23.2.3 [Imenu], page 287 を参照) を通じてアクセスできるように、mutoolプログラムが利用可能な場合にはそれを使って outline メニュー用のエントリーを生成します。システムに mutoolが存在する場合であってもこの機能を無効にしたい場合には、変数 doc-view-imenu-enabledを nil値にカスタマイズしてください。doc-view-imenu-formatや doc-view-flattenといった変数を使えば、imenuがアイテムをどのようにフォーマット、表示するかを更にカスタマイズすることができます。

31.4.2 DocView の検索

DocView モードでは、ファイルのテキストにたいして正規表現の検索を行なうことができます (Section 12.6 [Regexps], page 115 を参照してください)。検索のインターフェースは isearchが元になっています (Section 12.1 [Incremental Search], page 105 を参照してください)。

検索を開始するには、C-s (doc-view-search) または C-r (doc-view-search-backward) とタイプします。これはミニバッファーを使用して正規表現を読み取り、そのドキュメントでマッチした数をエコーします。C-sまたは C-rとタイプすることにより、マッチ間を前方または後方に移動できます。DocView モードはページイメージの中でマッチを表示できません。かわりに、カレントページのマッチするすべての行を一覧するツールチップを、(マウス位置に) 表示します。このツールチップを強制的に表示するには C-t (doc-view-show-tooltip) とタイプしてください。

新しい検索を開始するには、たとえば前方検索では C-u C-s、後方検索では C-u C-rのように、検索コマンドにプレフィクス引数を使用します。

31.4.3 DocView のスライス

印刷のために広い余白をもつドキュメントもあります。これらはスクリーンでドキュメントを読むとき邪魔になることがあります。なぜならこれらは画面スペースを消費して、スクロールが不便になるからです。

DocView では、表示するページのスライス *(slice)* を選択することにより、これらの余白を隠すことができます。スライスはページ内の矩形領域です。DocView で 1 度スライスを指定すると、閲覧するすべてのページに適用されます。

数値でスライスを指定するには、c s (doc-view-set-slice) とタイプします。その後、スライスの左上のピクセル位置 (pixel position) と、スライスの幅 (width) と高さ (height) を入力します。

スライスを指定するための、もっと便利でグラフィカルな方法は、c m (doc-view-set-slice-using-mouse) で、スライスの選択にマウスを使う方法です。これは単に、スライスにしたいリージョンの左上隅で左マウスボタンを押して、そのまま右下隅にマウスポインターを移動してマウスボタンを離します。

最適なスライスをセットする一番簡単な方法は、c b (doc-view-set-slice-from-bounding-box) とタイプすることにより、そのドキュメントから自動的に判断される BoundingBox 情報を使う方法です。

選択されたスライスを取り消すには、c r (doc-view-reset-slice) とタイプします。するとDocView は、余白全体を含めたページ全体を表示します。

31.4.4 DocView の変換

効率のために、DocView は gsにより生成されたイメージをキャッシュします。イメージをキャッシュするディレクトリーの名前は、変数 doc-view-cache-directoryにより与えられます。M-x doc-view-clear-cacheとタイプすることにより、キャッシュディレクトリーをクリアーできます。

現在閲覧中のドキュメントを強制的に再変換するには、type rまたは g (revert-buffer) とタイプします。カレントバッファーに関連付けられた変換プロセスを kill するには、K (doc-view-kill-proc) とタイプします。コマンド k (doc-view-kill-proc-and-buffer) は、変換プロセスと DocView バッファーを kill します。

31.5 Emacs からのシェルコマンドの実行

Emacs には、シェルサブプロセスに 1 つのコマンドラインを渡したり、入出力に Emacs バッファーを使用して対話的にシェルを実行するコマンドや、端末エミュレーターウィンドウでシェルを実行するコマンドがあります。

M-! *cmd* RET
: シェルコマンド *cmd*を実行して、出力を表示します (shell-command)。

M-| *cmd* RET
: リージョンの内容を入力としてシェルコマンド *cmd*を実行します。オプションでリージョンを出力で置き換えます (shell-command-on-region)。

M-& *cmd* RET
: シェルコマンド *cmd*を非同期で実行し、出力を表示します (async-shell-command)。

M-x shell
: Emacs バッファーを通じて入出力を行なう、サブシェルを実行します。その後で、コマンドを対話的に与えることができます。

M-x term
: Emacs バッファーを通じて入出力を行なう、サブシェルを実行します。その後でコマンドを対話的に与えることができます。完全な端末エミュレーションが利用できます。

(上記のコマンドの *cmd*引数、または他のコンテキストにおいて) 実行可能プログラムとして相対ファイル名を指定したときは、Emacs は変数 exec-pathにより指定されるディレクトリーのプログラムを検索します。この変数の値は、ディレクトリーのリストでなければなりません。デフォルト値は、Emacs が開始されたときの環境変数 PATHにより初期化されます (Section C.4.1 [General Variables], page 580 を参照してください)。

M-x eshellは、Emacs で完全に実装されたシェルを呼び出します。eshell については、自身のマニュアルにドキュメントされています。Emacs と共に配布される、Eshell Info マニュアルを参照してください。

31.5.1 単一のシェルコマンド

M-! (shell-command) は、ミニバッファーを使って 1 行のテキストを読み取り、それをシェルコマンドとして、そのコマンドのためだけに作成されたサブシェルで実行します。そのコマンドの標準入力は null デバイスです。シェルコマンドが出力を生成する場合、その出力はエコーエリア (出力が短い場合)、または別ウィンドウに表示される‘"*Shell Command Output*"’ (shell-command-buffer-name) という名前の Emacs バッファー (出力が長い場合) に表示されます。変数 resize-mini-windowsと max-mini-window-heightは、出力がエコーエリアにたいして長過ぎると Emacs が判断した場合に、ウィンドウのサイズ変更を制御します (Section 5.3 [Minibuffer Edit], page 30 を参照)。エコーエリアに何が表示されるかは、以下で説明する shell-command-dont-erase-bufferをカスタマイズすることによって影響を受けるかもしれないことに注意してください。

たとえばfoo.gzという名前のファイルを解凍する 1 つの方法は、M-! gunzip foo.gz RETとタイプすることです。このシェルコマンドは通常、ファイル fooを作成して、端末出力を生成しません。

たとえばM-1 M-!のようにshell-commandに数引数を指定した場合、別のバッファーではなく、カレントバッファーに端末出力を挿入します。これはデフォルトではポイントを出力の前、出力の後にマークを配置します (しかし以下の非デフォルト値 shell-command-dont-erase-bufferでこれを変更できます)。たとえばM-1 M-! gunzip < foo.gz RETは、カレントバッファーにファイルfoo.gzの解凍された内容を挿入します。

最後が‘&’でないシェルコマンドを指定した場合、コマンドは同期 *(synchronously)* で実行され、Emacs を継続して使用するには、コマンドが exit するのを待たなければなりません。待つのを中止するには、C-gとタイプします。これはシェルコマンドを終了するために、シグナル SIGINTを送ります (これは通常、シェルで C-cとタイプしたときに生成されるのと同じシグナルです)。その後、Emacs はコマンドが実際に終了するまで待ちます。シェルコマンドが終了しない場合 (そのコマンドがシグナル SIGINTを無視する場合) は、再度 C-gとタイプします。これは　そのコマンドにたいして、無視することが不可能なシグナル SIGKILLを送ります。

‘&’で終わるシェルコマンドは非同期 *(asynchronously)* で実行され、それを実行した後でも、継続して Emacs を使用できます。シェルコマンドを非同期で実行する前に、M-& (async-shell-command) とタイプすることもできます。これは最後の‘&’が必要ない点を除き、最後に‘&’を指定して M-!を呼び出すことと同じです。非同期シェルコマンドからの出力は、デフォルトでは‘"*Async Shell Command*"’ (shell-command-buffer-name-async) という名前のバッファーに送られます。このバッファーがウィンドウに表示されているかに関わらず、Emacs は出力をこのバッファーに挿入します。

同時に複数の非同期シェルコマンドを実行した場合、出力バッファーが競合します。この場合どのようにするか — たとえば既存の出力バッファーをリネームしたり、新しいコマンドに異なるバッファーを使用する — を、オプション async-shell-command-bufferで指定できます。他の可能なオプションについては、この変数のドキュメントを参照してください。

非同期シェルコマンドが出力を生成したときだけ出力バッファーが表示されるようにしたい場合は、async-shell-command-display-bufferを nilにセットしてください。

オプション async-shell-command-widthは非同期シェルコマンドの出力に利用可能な表示列数を定義します。正の整数はコマンド出力にその列数を使用するようシェルに指示します。デフォルト値 nilはシェルが提供する列数と同じ列数の使用を意味します。

上記コマンドにプロンプトにカレントディレクトリーを表示するようにさせるには変数shell-command-prompt-show-cwdを非 nil値にカスタマイズしてください。

M-| (shell-command-on-region) は M-!と同様ですが、入力を与えないかわりに、リージョンの内容をシェルコマンドの標準入力として渡します。数引数を指定した場合、古いリージョンの内容を、シェルコマンドの出力で置き換えます。

たとえばM-|で gpgプログラムを使用して、そのバッファーのキーが何かを見ることができます。そのバッファーが GnuPG キーを含む場合、C-x h M-| gpg RETとタイプして、バッファー全体の内容を gpgに送ることができます。これはキーのリストを shell-command-buffer-nameの値であるような名前のバッファーに出力します。

上記のコマンドは、変数 shell-file-nameで指定されたシェルを使用します。この変数のデフォルト値は、Emacs が開始されたときの環境変数 SHELLにより決定されます。相対ファイル名の場合、Emacs は exec-pathにリストされたディレクトリーを検索します (Section 31.5 [Shell], page 457 を参照してください)。

デフォルトディレクトリーがリモート (Section 15.15 [Remote Files], page 169 を参照) ならデフォルト値は/bin/shです。これは接続ローカルに shell-file-nameを宣言することにより変更可能です (Section 33.2.6 [Connection Variables], page 518 を参照)。

M-!または M-|にたいするコーディングシステムを指定するには、あらかじめコマンド C-x RET c を使用します。Section 19.10 [Communication Coding], page 229 を参照してください。

デフォルトでは、出力バッファーではエラー出力と標準出力が混ざって出力されます。しかし変数 shell-command-default-error-bufferの値を文字列に変更すると、エラー出力はその名前のバッファーに出力されます。

デフォルトではカレントバッファーに出力される場合を除き、シェルコマンド間に出力バッファーは消去されます。オプション shell-command-dont-erase-bufferの値を eraseに変更すると出力バッファーは常に消去されます。それ以外の非 nil値なら出力バッファーの消去は抑制されて、(出力バッファーがカレントバッファーでなければ) シェルコマンドの出力の挿入後にポイントを配置する位置を制御します:

beg-last-out
: ポイントを最後のシェルコマンドの出力の先頭に配置します。

end-last-out
: ポイントを最後のシェルコマンドの出力の最後 (出力バッファー終端) に配置します。

save-point
: シェルコマンドの出力の挿入前の位置にポイントをリストアします。

このオプションが非 nilの場合には出力バッファーの一部だけがエコーエリアに表示されるので、表示される出力は最後のコマンド以外の出力がエコーエリアに表示されるかもしれないことに注意してください。

出力バッファーがカレントバッファーでない場合には、そのバッファーの終端にシェルコマンドの出力が追加されます。

31.5.2 対話的なサブシェル

対話的にサブシェルを実行するには、M-x shellとタイプします。これは*shell*という名前のバッファーを作成 (または再使用) して、そのバッファーにたいして入出力を行なう、シェルのサブプロセスを実行します。つまりサブシェルからの端末出力はポイントの後に挿入されてポイントを進め、サブシェルにたいする端末入力はそのバッファーのテキストになります。サブシェルにたいして入力を与えるには、バッファーの最後に移動して入力をタイプし、RETで終了します。

デフォルトでは、サブシェルが対話的に呼び出された際、カレントウィンドウですでに*shell*バッファーを表示していない場合は、新たなウィンドウに*shell*バッファーが表示されます。この動作は display-buffer-alist(Section 17.6.1 [Window Choice], page 191 を参照) を通じてカスタマイズできます。

サブシェルがコマンドの実行を待つ間、ウィンドウまたはバッファーを切り替えて Emacs で他の編集を行なうことができます。Emacs はそれを処理するときがきたら (たとえばキーボード入力待ちになったとき)、サブシェルからの出力を Shell バッファーに挿入します。

Shell バッファーでは、プロンプトはフェイス comint-highlight-promptで表示され、サブミットされた入力行はフェイス comint-highlight-inputで表示されます。これにより入力行とシェル出力を容易に区別することができます。Section 11.8 [Faces], page 82 を参照してください。

複数のサブシェルを作成するには、(C-u M-x shell) のように) プレフィクス引数を指定して M-x shellを呼び出します。その後、このコマンドはバッファー名を読み取り、そのバッファーでサブシェ

ルを作成 (または再使用) します。M-x rename-uniquelyを使用して*shell*バッファーをリネームしてから、M-x shellで新しい*shell*バッファーを作成することもできます。異なるバッファーのサブシェルは、並行して独立に実行されます。

Emacs はあなたがエンターしたコマンドを調べて 'cd'といったコマンドを探すことによってカレントディレクトリーの追跡を試みます。カレントディレクトリーを変更する方法は多数存在するために、これはエラーの起きやすい解決手段です。したがって Emacs はこの情報をより信頼性の高い流儀で伝達するようデザインされた特別な OSC (Operating System Commands: オペレーティングシステムコマンド) のエスケープシーケンスも探します。たとえば以下のコマンドによって、シェルがそれぞれのプロンプトに適切なエスケープシーケンスをプリントするようアレンジする必要があります:

```
printf "\e]7;file://%s%s\e\\" "$HOSTNAME" "$PWD"
```

M-x shellによりシェルファイル名を指定するには、変数 explicit-shell-file-nameをカスタマイズします。これがnil(デフォルト) の場合、もし存在すれば Emacs は環境変数 ESHELLを使用します。そうでない場合は通常、変数 shell-file-nameを使用します (Section 31.5.1 [Single Shell], page 457 を参照してください)。しかしデフォルトディレクトリーがリモートの場合 (Section 15.15 [Remote Files], page 169 を参照してください)、シェルファイル名の入力を求めます。リモートファイル名を正しくタイプする方法に関するヒントは、Section 5.2 [Minibuffer File], page 28 を参照してください。

Emacs は新しいシェルにたいする入力として、もしそれが存在すればファイル~/.emacs_*shellname*の内容を送ります。ここで *shellname*は、そのシェルがロードされたファイルの名前です。たとえば bash を使う場合、送られるファイルは~/.emacs_bashになります。ファイルが見つからない場合、Emacs は~/.emacs.d/init_*shellname*.shの使用を試みます。

シェルにたいしてコーディングシステムを指定するには、M-x shellの直前にコマンド C-x RET cを使用します。C-x RET pとタイプすることにより、実行されているサブシェルにたいするコーディングシステムを変更することもできます。Section 19.10 [Communication Coding], page 229 を参照してください。

サブシェルでは、Emacs は環境変数 INSIDE_EMACSに '*version*,comint'をセットします。ここで *version*は、Emacs のバージョン (たとえば '28.1') です。プログラムはこの変数をチェックして、Emacs の内部で実行されているかを判断することができます。

31.5.3 Shell モード

Shell バッファーのためのメジャーモードは Shell モードです。このモードの特別なコマンドは C-cプレフィクスにバインドされており、最初に C-cをタイプしなければならない点を除けば、通常のシェルでの編集やジョブ制御文字と共通点があります。以下は Shell モードのコマンドのリストです:

RET サブシェルの入力にカレント行を送ります (comint-send-input)。行頭のシェルプロンプトは省略されます (Section 31.5.4 [Shell Prompts], page 463 を参照してください)。ポイントがバッファーの最後にある場合、これは通常の対話的なシェルにおいてコマンド行をサブミットするのに似ています。しかし、シェルバッファーの他の場所でも、RETを呼び出してカレント行を入力としてサブミットできます。

TAB シェルバッファーのポイントの前のコマンド名、またはファイル名を補完します (completion-at-point)。これは通常の Emacs 補完ルールを使用します (Section 5.4 [Completion], page 31 を参照してください)。ファイル名、環境変数名、シェルコマンドヒストリー、ヒストリー参照が補完候補になります (Section 31.5.5.3 [History References], page 465 を参照してください)。補完を制御するオプションについては、Section 31.5.7 [Shell Options], page 466 を参照してください。

`M-?` ポイントの前のファイル名にたいして、可能性のある補完リストを一時的に表示します (`comint-dynamic-list-filename-completions`)。

`C-d` delete 文字、または EOF文字を送ります (`comint-delchar-or-maybe-eof`)。シェルバッファーの最後でこれをタイプすると、サブシェルに EOFが送られます。バッファーの他の場所でタイプすると、通常どおり文字を削除します。

`C-c C-a` 行の先頭に移動しますが、プロンプトがある場合はプロンプトの後に移動します (`comint-bol-or-process-mark`)。このコマンドを連続して繰り返すと、2 回目はプロセスマーク (process mark) に戻ります。これはサブシェルにまだ送信されていない入力の開始位置のことです (通常これは同じ場所 — その行のプロンプトの最後 — ですが、`C-c SPC`の後では、プロセスマークは前の行にあるかもしれません)。

`C-c SPC` 複数行の入力を累積して、それらを合わせて送ります (`comint-accumulate`)。このコマンドはポイントの前に改行を挿入しますが、その前のテキストをサブシェルの入力として— 少なくともその時点では — 送りません。改行の前後の行は両方、(それらを分割する改行も含めて)`RET`をタイプしたとき一緒に送られます。

`C-c C-u` まだ入力として送られていない、バッファーの最後のすべてのテキストを kill します (`comint-kill-input`)。ポイントがバッファーの最後にない場合、これはポイントの前のテキストだけを kill します。

`C-c C-w` ポイントの前の単語を kill します (`backward-kill-word`)。

`C-c C-c` シェルまたは (もしあれば) カレントのサブジョブに割り込み (interrupt) ます (`comint-interrupt-subjob`)。このコマンドはシェルバッファーの最後の、まだ送られていない任意のシェル入力も kill します。

`C-c C-z` シェルまたは (もしあれば) カレントのサブジョブを停止 (stop) します。(`comint-stop-subjob`)。このコマンドはシェルバッファーの最後の、まだ送られていない任意のシェル入力も kill します。

`C-c C-\` シェルまたは (もしあれば) カレントのサブジョブに、シグナル QUIT を送ります (`comint-quit-subjob`)。このコマンドはシェルバッファーの最後の、まだ送られていない任意のシェル入力も kill します。

`C-c C-o` 前のシェルコマンドからの出力の一群を削除します (`comint-delete-output`)。これはシェルコマンドが大量の出力を吐くときに便利です。プレフィクス引数を指定すると、このコマンドは後でどこかに yank (Section 9.2 [Yanking], page 62 を参照) できるように削除したテキストを `kill-ring` (Section 9.2.1 [Kill Ring], page 62 を参照) に保存します。

`C-c C-s` 前のシェルコマンドからの出力の一群をファイルに書き込みます (`comint-write-output`)。プレフィクス引数を指定した場合は、ファイルに追加で書き込みます。出力の最後のプロンプトは書き込まれません。

`C-c C-r`
`C-M-l` 前の一群の出力の最初がウィンドウの最上部になるようにスクロールし、カーソルもそこに移動します (`comint-show-output`)。

`C-c C-e` バッファーの最後の行がウィンドウの最下部になるようにスクロールします (`comint-show-maximum-output`)。

`C-c C-f` シェルコマンド 1 つ分、前方に移動しますが、カレント行を超えては移動しません (`shell-forward-command`)。変数 `shell-command-regexp`は、コマンドの最後を認識する方法を指定します。

`C-c C-b` シェルコマンド 1 つ分、後方に移動しますが、カレント行を超えて移動はしません (`shell-backward-command`)。

`M-x dirs` シェルに作業ディレクトリーを尋ね、Shell バッファーのデフォルトディレクトリーを更新します。Section 31.5.6 [Directory Tracking], page 465 を参照してください。

`M-x comint-send-invisible RET` *text* `RET`
エコーなしで *text*を読み取った後、それをシェルの入力として送ります。これはシェルコマンドでパスワードを尋ねるプログラムを実行するとき便利です。

デフォルトでは Emacs はパスワードをエコーしないことに注意してください。もし本当にエコーさせたいときは、以下の Lisp 式を評価します (Section 24.9 [Lisp Eval], page 329 を参照してください):

```
(remove-hook 'comint-output-filter-functions
             'comint-watch-for-password-prompt)
```

`M-x comint-continue-subjob`
シェルプロセスを継続します。これは間違ってシェルプロセスをサスペンドしてしまったときなどに便利です。[2]

`M-x comint-strip-ctrl-m`
シェル出力のカレントグループから、すべての control-M 文字を破棄します。このコマンドを使うもっとも便利な方法は、サブシェルからの出力を受け取ったときに自動的に実行されるようにする方法です。これを行なうには、以下の Lisp 式を評価します:

```
(add-hook 'comint-output-filter-functions
          'comint-strip-ctrl-m)
```

`M-x comint-truncate-buffer`
このコマンドはシェルバッファーを、変数 `comint-buffer-maximum-size`により指定される、特定の最大行数に切り詰めます。以下は、サブシェルから出力を受けとるとき、毎回自動的にこれを行なう方法です:

```
(add-hook 'comint-output-filter-functions
          'comint-truncate-buffer)
```

デフォルトでは、Shell モードは (たとえばテキストカラーの変更等に) 一般的な ANSIエスケープコードを使います。init ファイルに以下を記述すれば、Emacs はオプションで OSC (Operating System Codes：オペレーティングシステムコード) のような拡張エスケープコードもいくつかサポートします:

```
(add-hook 'comint-output-filter-functions 'comint-osc-process-output)
```

これを有効にすれば、たとえば`ls --hyperlink`からの出力から Shell モードバッファーにクリック可能ボタンが作成されるでしょう。

Shell モードは、サブプロセスと対話的に通信を行なう一般的な用途向けの Comint モードから派生したモードです。上記のコマンド名からも判るとおり、Shell モードのほとんどの機能は、実際には

[2] シェルプロセスをサスペンドするべきではありません。これはシェルのサブジョブのサスペンドとはまったく違います。サブジョブのサスペンドは通常行なわれますが、サブジョブを継続するためにはシェルを使用しなければなりません。このコマンドはそれを行ないません。

Comint モードが由来です。Shell モードの特別な機能には、ディレクトリー追跡機能、およびいくつかのユーザーコマンドが含まれます。

Comint モードの変種を使う他の Emacs 機能には、GUD (Section 24.6 [Debuggers], page 315 を参照してください) や、`M-x run-lisp` (Section 24.11 [External Lisp], page 330 を参照してください) が含まれます。

サブプロセスとして選択した任意のプログラムを実行するために、Shell モードに特化しない未修正の Comint モードを使用するには、`M-x comint-run`を使用することができます。プログラムに引数を渡すには `C-u M-x comint-run`を使用します。

31.5.4 Shell プロンプト

プロンプトとは、新しいユーザー入力を受け取る準備ができたことを表す、プログラムによるテキスト出力のことです。Comint モード (したがって Shell モードも) は通常、サブプロセスからの出力にもとづいて、バッファーの一部をプロンプトとして自動的に判断します (具体的には、改行で終端されていない任意の出力行を受け取ったときは、プロンプトとみなします)。

Comint モードは、バッファーを 2 つのタイプのフィールド *(fields)* に分けます。1 つは入力フィールド (ユーザーの入力がタイプされるところ) で、もう 1 つは出力フィールド (入力フィールド以外) です。プロンプトは出力フィールドの一部です。ほとんどの Emacs 移動コマンドは、そのコマンドが複数行を移動しない限り、フィールド境界を超えることはありません。たとえば、ポイントがシェルコマンドの入力フィールドにある場合、`C-a`はポイントを入力フィールドの先頭、プロンプトの後に配します。内部的には、フィールドはテキストプロパティ`field`を使って実装されています (Section "Text Properties" in *the Emacs Lisp Reference Manual* を参照してください)。

変数 `comint-use-prompt-regexp`を非 `nil`値に変更した場合、Comint モードは正規表現を使ってプロンプトを認識します (Section 12.6 [Regexps], page 115 を参照してください)。Shell モードでは、その正規表現は変数 `shell-prompt-pattern`により指定されます。`comint-use-prompt-regexp`のデフォルト値は `nil`です。なぜならプロンプト認識のためのこの方法は信頼性が低いためです。しかし特殊な状況では、これを非 `nil`値にセットしたいと思うこともあるでしょう。そのような場合、Emacs は Comint バッファーをフィールドに分割しないので、一般的な移動コマンドは、テキストプロパティを使用せず、それらのコマンドがバッファーで通常行なうように振る舞います。しかし、バッファーを便利に操作するために、パラグラフ移動コマンドを使うことができます (Section 22.3 [Paragraphs], page 253 を参照してください)。Shell モードでは、Emacs はパラグラフ境界に `shell-prompt-pattern`を使用します。

31.5.5 Shell コマンドヒストリー

Shell バッファーは、以前のコマンドを繰り返す 3 つの方法をサポートします。1 つ目はミニバッファーヒストリーにたいして使うのと同じようなキーを使う方法です。つまり、これらはミニバッファーで行なうのと同じように、前のコマンドからテキストを挿入して、ポイントを常にバッファーの最後に保ちます。2 つ目は、バッファーを移動して元の場所から前の入力を取得して、それらを再実行したり、バッファーの最後にコピーする方法です。3 つ目は '`!`'スタイルのヒストリー参照を使う方法です。

31.5.5.1 Shell ヒストリーリング

`M-p`
`C-UP` 以前のシェルコマンドから、次に古いコマンドを取り出します (`comint-previous-input`)。

M-n
C-DOWN 以前のシェルコマンドから、次に新しいコマンドを取り出します (comint-next-input)。

M-r 以前のシェルコマンドにたいして、インクリメンタル regexp 検索を開始します (comint-history-isearch-backward-regexp)。

C-c C-x ヒストリーから次のコマンドを取り出します (comint-get-next-from-history)。

C-c . 以前のシェルコマンドから引数を 1 つ取り出します (comint-input-previous-argument)。

C-c C-l そのバッファーのシェルコマンドのヒストリーを、別のウィンドウに表示します (comint-dynamic-list-input-ring)。

Shell バッファーは、以前に入力したシェルコマンドのヒストリーを提供します。ヒストリーからシェルコマンドを再利用するには、編集コマンド M-p、M-n、M-rを使用します。これらは、ミニバッファーではなく Shell バッファーを操作する点を除き、ミニバッファーヒストリーコマンド (Section 5.5 [Minibuffer History], page 36 を参照してください) と同じように機能します。また Shell バッファー内での M-rは、シェルコマンドヒストリーにたいするインクリメンタル検索を呼び出します。

M-pは、シェルバッファーの最後から以前のシェルコマンドを取り出します。連続して M-pを使用すると、古いシェルコマンドを連続して取り出し、その度にシェル入力の候補として表示されているテキストを置き換えます。M-nも同様に機能しますが、これはバッファーから、より新しいシェルコマンドを連続して探します。C-UPは M-pと同様に機能し、C-DOWNは M-nと同様に機能します。

ヒストリー検索コマンド M-rは、以前のシェルコマンドにたいしてインクリメンタル正規表現検索を開始します。M-rとタイプした後に、検索したい文字列か正規表現のタイプを開始します。するとマッチする最後のシェルコマンドがカレント行に表示されます。インクリメンタル検索コマンドは通常の効果をもちます— たとえば C-sおよび C-rは前方または後方に、次のマッチを検索します (Section 12.1 [Incremental Search], page 105 を参照してください)。探している入力が見つかったら、検索を終了するために RETをタイプします。これにより、入力がコマンドラインに配されます。ヒストリーリストを操作する前にタイプしていた入力の一部は、ヒストリーリングの先頭または最後に達したときに復元されます。

以前に実行した一連のシェルコマンドを、同じ順番で再実行できたら便利なこともあります。これを行なうには、最初に順番に再実行する 1 番目のコマンドを検索します。その後 C-c C-xとタイプします。これは次のコマンド — つまり再実行したコマンドの次のコマンドを取り出します。それから RETでそのコマンドを実行します。C-c C-x RETを繰り返しタイプすることにより、連続する複数のコマンドを再実行することができます。

コマンド C-c . (comint-insert-previous-argument) は、Bash の ESC .のように、以前のコマンドから個別に引数をコピーします。一番簡単な使い方は、以前のシェルコマンドから最後の引数をコピーする方法です。プレフィクス引数 *n*を指定すると、*n*番目の引数をコピーします。繰り返し C-c .とタイプすることにより、さらに古いシェルコマンドからコピーします。この場合、常に同じ *n* の値を使用します (C-c .を繰り返すときは、プレフィクス引数を与えないでください)。

comint-insert-previous-argument-from-endを非 nil値にセットすると、C-c .はかわりに最後から数えて *n*番目の引数をコピーします。これは zshの ESC .をエミュレートしたものです。

これらのコマンドは、以前のシェルコマンドのテキストを、シェルバッファー自身からではなく、特別なヒストリーリストから取得します。したがってシェルバッファーを編集したり、広い範囲を kill しても、これらのコマンドがアクセスするヒストリーに影響はありません。

シェルの中には、コマンドヒストリーをファイルに保存して、以前のシェルセッションからコマンドを参照できるようにするものがあります。Emacs は選択されたシェルにたいして、コマンドヒスト

リーを初期化するために、コマンドヒストリーファイルを読み込みます。ヒストリーファイル名は、bash では~/.bash_history、ksh では~/.sh_history、他のシェルでは~/.historyです。

リモートホストでシェルを実行した場合には、このセッティングは変数 tramp-histfile-overrideに上書きされるかもしれません。この変数は nilにセットすることを推奨します。

31.5.5.2 Shell ヒストリーのコピー

C-c C-p 前のプロンプトにポイントを移動します (comint-previous-prompt)。

C-c C-n 次のプロンプトにポイントを移動します (comint-next-prompt)。

C-c RET ポイント位置の入力コマンドをコピーして、そのコピーをバッファーの最後に挿入します (comint-copy-old-input)。これは以前のコマンドにポイントを移動したときに便利です。コマンドをコピーした後、RETでそのコピーを入力として実行することができます。再実行する前に、そのコピーを編集することもできます。このコマンドを出力行で使用した場合、その行をバッファーの最後にコピーします。

mouse-2 comint-use-prompt-regexpが nil(デフォルト) の場合、クリックした以前の入力コマンドをコピーして、そのコピーをバッファーの最後に挿入します (comint-insert-input)。comint-use-prompt-regexpが非 nil、または以前の入力以外の場所をクリックしたときは、通常のように yank します。

以前の入力に移動して、C-c RETまたは mouse-2でコピーすることは、M-pを充分な回数使用して、ヒストリーリングから以前の入力を取り出したときと同じ結果— つまり同じバッファー内容 — を生成します。しかし C-c RETは、バッファーからテキストをコピーするので、入力をシェルに送信した後に入力テキストを編集していた場合は、ヒストリーリストと異なります。

31.5.5.3 Shell ヒストリーの参照

csh や bash を含むさまざまなシェルは、‘!’や‘^’で始まるヒストリー参照 *(history references)* をサポートします。Shell モードはこれらを認識して、ヒストリーの置き換える処理をします。

ヒストリー参照を挿入して TABとタイプすると、これは入力ヒストリーからマッチするコマンドを検索して、必要なら置換を行い、結果をバッファー内のヒストリー参照の場所に配します。たとえば、一番最近の‘mv’で始まるコマンドを取り出すには、! m v TABとタイプします。必要ならコマンドを編集して、RETでシェルにたいしてコマンドを再実行できます。

Shell モードはヒストリー参照をシェルに送るとき、オプションでヒストリー参照を展開できます。これを行なうには、変数 comint-input-autoexpandを inputにセットします。SPCを comint-magic-spaceにバインドすれば、SPCでヒストリー参照を展開できます。Section 33.3.5 [Rebinding], page 521 を参照してください。

Shell モードは、ヒストリー参照がプロンプトの後にあればヒストリー参照を認識します。Shell モードがプロンプトを認識する方法については、Section 31.5.4 [Shell Prompts], page 463 を参照してください。

31.5.6 ディレクトリーの追跡

Shell モードは、Shell バッファーのデフォルトディレクトリー (Section 15.1 [File Names], page 145 を参照してください) をシェルの作業ディレクトリーと同一に保つために、サブシェルに与えられる‘cd’、‘pushd’、‘popd’のコマンドを追跡します。これは、送信する入力行を調べることにより認識されます。

これらのコマンドにたいしてエイリアスを使用する場合、変数 shell-pushd-regexp、shell-popd-regexp、shell-cd-regexpに適切な正規表現 (Section 12.6 [Regexps], page 115 を参照してください) をセットすることにより、それらも認識するよう Emacs に指示できます。たとえば、shell-pushd-regexpがシェルコマンドラインの先頭にマッチした場合、その行は pushdコマンドとして記録されます。これらのコマンドは、シェルコマンドラインの先頭だけで認識されます。

Emacs が作業ディレクトリーの変更に際して混乱した場合は、M-x dirsを試してください。このコマンドはシェルに作業ディレクトリーを尋ねて、それに対応してデフォルトディレクリーを更新します。これは、一般的なコマンド構文のほとんどをサポートするシェルでは機能しますが、特殊なシェルでは機能しないかもしれません。

シェルの作業ディレクトリーを追跡する他の方法を実装した、バッファーローカルなマイナーモードの、Dirtrack モードを使うこともできます。この方法を使うには、シェルプロンプトに常に作業ディレクトリーが含まれていなければならず、プロンプトのどの部分が作業ディレクトリーを含むか認識するための正規表現を与えなければなりません。詳細は、変数 dirtrack-listのドキュメントを参照してください。Dirtrack モードを使用するには、Shell バッファーで M-x dirtrack-modeとタイプするか、shell-mode-hookに dirtrack-modeを追加します (Section 33.2.2 [Hooks], page 509 を参照してください)。

31.5.7 Shell モードのオプション

変数 comint-scroll-to-bottom-on-inputが非 nilの場合、挿入および yank コマンドは、挿入する前に選択されたウィンドウを、バッファーの最後までスクロールします。デフォルトは nilです。

comint-scroll-show-maximum-outputが非 nilの場合、ポイントが最後にあるとき到着した出力は、可能な限り有用なテキストを表示するために、テキストの最後の行がウィンドウの一番下になるようなスクロールを試みます (これはほとんどの端末のスクロール動作を真似た動作です)。デフォルトは tです。

comint-move-point-for-outputをセットすることにより、出力が到着したときにバッファーの最後にポイントをジャンプさせることができます— その前にポイントがバッファーのどこにあろうと関係ありません。値が thisの場合、選択されたウィンドウ内でポイントがジャンプします。値が allの場合、Comint バッファーを表示するすべてのウィンドウでポイントがジャンプします。値が otherの場合、カレントバッファーを表示する、選択されていないすべてのウィンドウでポイントがジャンプします。デフォルト値は nilで、これはポイントが最後にジャンプしないことを意味します。

comint-prompt-read-onlyをセットした場合、Comint バッファーのプロンプトは読み取り専用になります。

変数 comint-input-ignoredupsは、連続する同じ入力を入力ヒストリーに保存するかを制御します。非 nil値は、入力が前の入力と同じ場合は省略することを意味します。デフォルトは nilで、これは入力が前の入力と同じでも保存することを意味します。

ファイル名の補完をカスタマイズする 3 つの変数があります。変数 comint-completion-addsuffixは、完全に補完されたファイル名またはディレクトリー名の最後にスペースまたはスラッシュを挿入するかを制御します (非 nilは、スペースまたはスラッシュを挿入することを意味します)。comint-completion-recexactが非 nilの場合、通常の Emacs 補完アルゴリズムが 1 文字も追加できないようなときは、TABで一番短い利用可能な補完を選択するよう指示します。comint-completion-autolistが非 nilの場合、補完が完全でないときは、利用可能なすべての候補をリストするよう指示します。

コマンド補完は通常、実行可能ファイルだけを考慮します。shell-completion-execonlyを nilにセットした場合は、実行可能ファイル以外も同様に考慮します。

変数 shell-completion-fignoreは、Shell モードで無視するファイル名の拡張子のリストを指定します。デフォルトは nilですが、‘~’、‘#’、‘%’で終わるファイル名を無視するために ("~" "#" "%")をセットするユーザーもいます。他の Comint モードに関連するモードは、かわりに変数 comint-completion-fignoreを使用します。

シェルコマンド補完の実装の詳細は、shell-dynamic-complete-command関数の lisp ドキュメントで見ることもできます。

‘pushd’の動作を設定することができます。shell-pushd-tohomeは、引数を与えない場合に‘pushd’が‘cd’のように振る舞うかを制御します。shell-pushd-dextractは、数引数を与えたときローテートするのではなく pop するかを制御します。shell-pushd-duniqueは、ディレクトリースタックにディレクトリーがない場合だけ追加するかを制御します。もちろん選択する値は背後のシェルに適合する必要があります。

Comint モードは、環境変数 TERMの値を安全なデフォルト値にセットしますが、この値はいくつかの有用な機能を無効にします。たとえばカラーがサポートされているかの判断に TERMを使用するため、アプリケーションではカラーが無効になっています。したがってシステムの terminfo データベースで定義されている、より高度な機能をもつ端末を選択できるように、Emacs はオプション comint-terminfo-terminalを提供します。system-uses-terminfoが非 nilの場合には、Emacs はこのオプションの値を TERMとして使用します。

comint-terminfo-terminalと system-uses-terminfoはいずれも、リモートシステムの期待にこれらのオプションを合致させられるように接続ローカル変数として定義できます (Section 33.2.6 [Connection Variables], page 518 を参照)。

31.5.8 Emacs の端末エミュレーター

テキスト端末エミュレーターでサブシェルを実行するには、M-x termを使用します。これは *terminal*という名前のバッファーを作成 (または再利用) して、キーボードを入力とするサブシェルを実行し、出力はそのバッファーになります。

端末エミュレーターは、2 つの入力モードをもつ Term モードを使用します。*line* モード *(line mode)* では、Term は基本的に Shell モードのように振る舞います (Section 31.5.3 [Shell Mode], page 460 を参照してください)。*char* モード *(char mode)* では、文字は端末入力として直接サブシェルに送られます。唯一の例外は端末エスケープ文字で、デフォルトは C-cです (Section 31.5.9 [Term Mode], page 468 を参照してください)。入力をエコーするのはサブシェルの役目です。サブシェルからの端末出力は、バッファーのポイントの後に送られます。

(Emacs のような) いくつかのプログラムでは、端末スクリーンで詳細に外観を制御する必要があります。これらのプログラムは特別な制御コードを送ることによりこれを行ないます。Term モードは、xtermを含むほとんどの現代的な端末で使用できる、ANSI 標準の VT100 スタイルのエスケープシーケンスを認識・処理します (したがって、実際に Emacs の Term ウィンドウ内で Emacs を実行することもできます)。

termフェイスは、端末エミュレーターのテキストのデフォルトの外観を指定します (デフォルトは defaultフェイスと同じ外観です)。端末の制御コードがテキストの外観を変更するために使用される場合、これらは端末エミュレーター内で、フェイス term-color-black、term-color-red、term-color-green、term-color-yellow、term-color-blue、term-color-magenta、term-color-cyan、term-color-white、term-color-underline、term-color-boldで表示されます。Section 11.8 [Faces], page 82 を参照してください。

シリアルポートに接続されたデバイスと通信するために、Term モードを使うこともできます。Section 31.5.11 [Serial Terminal], page 468 を参照してください。

サブシェルをロードするために使用されるファイル名は、Shell モードと同じ方法で決定されます。複数の端末エミュレーターを作成するには、Shell モードと同じように、`M-x rename-uniquely`を使って、バッファー`*terminal*`を違う名前にリネームします。

Shell モードとは異なり、Term モードは入力を調べてカレントディレクトリーを追跡することはしません。しかし、いくつかのシェルはカレントディレクトリーを Term に告げることができます。これは`bash`のバージョン 1.15 以降では自動的に行なわれます。

31.5.9 Term モード

Term モードで line モードと char モードを切り替えるには、以下のコマンドを使用します:

`C-c C-j` line モードに切り替えます (`term-line-mode`)。すでに line モードのときは何もしません。

`C-c C-k` char モードに切り替えます (`term-char-mode`)。すでに char モードのときは何もしません。

以下のコマンドは char モードだけで利用可能です:

`C-c C-c` サブシェルに、リテラルの`C-c`を送ります (`term-interrupt-subjob`)。

`C-c` *`char`* これは通常の Emacs での `C-x` *`char`*と等価です。たとえば `C-c o`は、通常 `C-x o`にグローバルにバインドされている‘`other-window`’を呼び出します。

Term モードには page-at-a-time(1 度に 1 ページ) 機能があります。これが有効な場合、出力の画面の最後で一時停止します。

`C-c C-q` page-at-a-time 機能を切り替えます (`term-pager-toggle`)。このコマンドは line モードと char モードの両方で機能します。この機能が有効な場合、モードラインには単語‘`page`’が表示され、Term が 1 画面に収まらない出力を受け取ったときは、一時停止してモードラインに‘`**MORE**`’を表示します。`SPC`とタイプすると次の 1 画面分の出力を表示し、`?`でオプションを見ることができます。このインターフェースは、`more`プログラムと同様です。

31.5.10 リモートホストのシェル

通常の端末から使うコマンド (たとえば`ssh`コマンド) を使用して、Term ウィンドウからリモートコンピューターにログインすることができます。

パスワードを尋ねるようなプログラムは、通常パスワードをエコーしないので、パスワードはバッファーにも表示されません。バッファーが char モードの場合、実際の端末で使用しているのと同じ動作になります。line モードの場合、パスワードは一時的に表示されますが、リターンをタイプするとパスワードは消去されます (これは自動的に行なわれます。特別なパスワード処理は行なっていません)。

別のマシンにログインしているときは、リモートログインコマンドのために、環境変数`TERM`をセットすることにより、端末タイプを指定する必要があります (bash を使用している場合、リモートログインコマンドの前に、カンマで区切らず値割り当てを記述することにより、これを行なうことができます)。端末タイプ‘`ansi`’または‘`vt100`’は、ほとんどのシステムで動作するでしょう。

31.5.11 シリアル端末

コンピューターに、シリアルポートに接続されたデバイスがあるとき、`M-x serial-term`とタイプすることにより、デバイスと通信することができます。このコマンドは新しい Term モードのバッファー

のために、ポート番号、スピード、スイッチを尋ねます。Emacs は、通常の Term モードの端末で行なうのと同じように、このバッファーを通じてシリアルデバイスと通信します。

シリアルポートのスピードはビット毎秒で計ります。もっとも一般的なスピードは 9600 ビット毎秒です。モードラインをクリックすることにより、このスピードを対話的に変更できます。

シリアルポートはモードラインの '8N1'をクリックしても設定できます。デフォルトでは、シリアルポートは '8N1'に設定されており、これは各バイトは 8 ビットからなり、パリティービットチェックなし、ストップビットが 1 であることを意味します。

スピードや設定が間違っている場合、デバイスと通信できず、おそらくウィンドウにはゴミが出力されるでしょう。

31.6 サーバーとしての Emacs の使用

さまざまなプログラムが、特定のテキスト断片を編集するために、あなたが選択したエディターを呼び出すことができます。たとえば、バージョンコントロールシステム (Section 25.1 [Version Control], page 332 を参照してください) は、バージョンコントロールログを入力するためのエディターを呼び出し、Unix の `mail`ユーティリティーは送信メッセージの入力にエディターを呼び出します。慣例により、選択するエディターは、環境変数 `EDITOR`で指定されます。しかし `EDITOR`を '`emacs`'にセットした場合、Emacs が呼び出されますが、これは便利な方法ではありません— なぜなら新しい Emacs プロセスが開始されるからです。なぜこれが不便かというと、新たな Emacs プロセスは、既存の Emacs プロセスのバッファー、コマンドヒストリー、その他の情報を共有しないからです。

Emacs を *edit server(*編集サーバー*)* としてセットアップして、Emacs が外部からの編集リクエストを "listen" し、それに応じて動作させることにより、この問題を解決できます。Emacs サーバーを開始するにはさまざまな方法があります:

- `M-x server-start`とタイプするか、init ファイル (Section 33.4 [Init File], page 528 を参照してください) に式 `(server-start)`を記述して、既存の Emacs プロセスでコマンド `server-start`を実行します。既存の Emacs プロセスがサーバーになり、Emacs を終了すると、サーバーはその Emacs プロセスとともに終了します。
- コマンドラインオプション '`--daemon`'のいずれか 1 つを使用して、Emacs を *daemon* として実行します。Section C.2 [Initial Options], page 576 を参照してください。Emacs がこの方法で開始された場合、初期化の後に `server-start`を呼び出して、初期フレームを開きません。その後クライアントからの編集リクエストを wait します。
- コマンドラインオプション '`--alternate-editor=""`'でコマンド `emacsclient`を実行します。Emacs デーモンがまだ実行されていない場合に限り Emacs デーモンを開始します。
- オペレーティングシステムがスタートアップ管理に `systemd`を使用する場合、提供される *systemd* ユニットファイル *(systemd unit file)* を使用して、ログイン時に自動的にデーモンモードの Emacs を開始できます。これをアクティブにするには:

  ```
  systemctl --user enable emacs
  ```

  (Emacs が標準的な場所にインストールされていない場合は、`~/.config/systemd/user/`のような標準的なディレクトリーに `emacs.service`をコピーする必要があるかもしれません。)
- 指定されたソケットに接続イベントが発生したとき、外部プロセスは Emacs サーバーを呼び出して、そのソケットを新たな Emacs サーバーのプロセスに渡すことができます。このインスタンスは `systemd`のソケット機能です。`systemd`サービスはソケットを作成して、そのソケットへの接続を listen します。`emacsclient`が最初にこれに接続したとき、`emacsclient`による接続のために、`systemd`は Emacs サーバーを起動して、そのソケットを渡すことができます。`emacsclient`が最初にこれに接続したとき、`emacsclient`による接続のために、`systemd`は

Emacs サーバーを起動して、そのソケットを渡すことができます。この機能を使用するためのセットアップは:

`~/.config/systemd/user/emacs.socket`:

```
[Socket]
ListenStream=/path/to/.emacs.socket
DirectoryMode=0700

[Install]
WantedBy=sockets.target
```

(上述のファイル `emacs.service`もインストールされていなければなりません。)

パス `ListenStream`は、Emacs が `emacsclient`からの接続を listen するパスです。これは、あなたが選択するファイルです。

Emacs サーバーを一度開始すると、`emacsclient`というシェルコマンドを使用して Emacs に接続して、ファイルを visit するよう指示できます。環境変数 `EDITOR`に '`emacsclient`'をセットすれば、外部プログラムは編集のために既存の Emacs プロセスを使用できます。[3]

変数 `server-name`を使って、一意なサーバー名を与えることにより、同一マシン上で複数の Emacs サーバーを実行することができます。たとえば `M-x set-variable RET server-name RET "foo" RET`は、サーバー名を '`foo`'にセットします。`emacsclient`プログラムは TCP ソケット使用の有無に応じて '`-s`'オプション、または '`-f`'オプション (Section 31.6.3 [emacsclient Options], page 472 を参照) で、名前によりサーバーを指定できます (Section 31.6.3 [emacsclient Options], page 472 を参照)。

複数の Emacs デーモン (Section C.2 [Initial Options], page 576 を参照) を実行したい場合には、以下のようにデーモンごとにそれぞれ独自のサーバー名を与えることができます:

```
emacs --daemon=foo
```

オプションとして特定の条件が満足された際に、Emacs サーバーを自動的に停止できます。これを行うには、init ファイルで以下の引数のいずれかを指定して関数 `server-stop-automatically`を呼び出してください (Section 33.4 [Init File], page 528 を参照)。

- 引数が `empty`ならクライアント、ファイルを visit している未保存のバッファー、実行中プロセスがすべて無くなるとサーバーは停止します。
- 引数が `delete-frame`なら、最後のクライアントフレームが閉じられる際にファイルを visit している未保存のバッファーを保存するか、終了していないプロセスそれぞれについて停止してよいか尋ねて、問題なければサーバーを停止します。
- 引数が `kill-terminal`なら、`C-x C-c` (`save-buffers-kill-terminal`) で最後のクライアントフレームが閉じられる際にファイルを visit している未保存のバッファーを保存するか、終了していないプロセスそれぞれについて停止してよいか尋ねて、問題なければサーバーを停止します。

一意なサーバー名によりサーバーを定義した場合、他の Emacs インスタンスからそのサーバーに接続し、`server-eval-at`関数を使用して、そのサーバーで Lisp 式を評価できます。たとえば `(server-eval-at "foo" '(+ 1 2))`は、式 `(+ 1 2)`をサーバー '`foo`'で評価して、3を返します (そのような名前のサーバーが存在しない場合はエラーをシグナルします)。現在のところ、これは主に開発者に有用な機能です。

[3] 別の環境変数を使うプログラムもいくつかあります。たとえば、TEX が '`emacsclient`'を使うようにするには、環境変数 `TEXEDIT`を '`emacsclient +%d %s`'にセットします。

オペレーティングシステムのデスクトップ環境が freedesktop.org-compatible (これはほとんどの GNU/Linux やその他の最近の Unix 風 GUI が該当する) なら、`emacsclient`で Emacs サーバーに接続するために‘`Emacs (Client)`’メニューエントリーを使用できます。デーモンがすでに実行中でなければ開始されます。

31.6.1 TCP Emacs server

Emacs サーバーは通常、接続にたいしてローカルの Unix ドメインソケットを listen します。MS-Windows のようないくつかのオペレーティングシステムは、ローカルソケットをサポートしません。そのような場合、サーバーはかわりに TCP ソケットを listen します。ローカルソケットがサポートされている場合でも、サーバーに TCP ソケットを listen させるほうが便利な場合があります (たとえばリモートマシンから Emacs サーバーに接続する必要がある場合)。ローカルソケットのかわりに TCP ソケットを Emacs に listen させる場合は、`server-use-tcp`に非`nil`をセットできます。これは OS がローカルソケットをサポートしない場合は、デフォルトです。

Emacs サーバーが TCP を使うように設定されている場合は、デフォルトでは localhost インターフェイスのランダムなポートを listen します。`server-host`と`server-port`を使用して、これを他のインターフェイス、および/または固定されたポートに変更できます。

TCP ソケットは、ファイルシステムのパーミッションの対象ではありません。TCP ソケットを通じて、誰が Emacs サーバーと通信できるか何らかの制御を得るには、`emacsclient`プログラムがサーバーに認証キーを送信しなければなりません。このキーは通常、Emacs サーバーによりランダムに生成されます。これが推奨されるモードです。

必要なら、`server-auth-key`変数をセットすることにより、認証キーに静的な値をセットできます。このキーは、スペースを除くプリント可能な 64 文字の ASCII 文字 (これは‘`!`’から‘`~`’、10 進コードの 33 から 126 を意味します) から構成されなければなりません。ランダムキーを得るために、`M-x server-generate-key`を使用できます。

TCP Emacs サーバーを開始したとき、Emacs は`emacsclient`がサーバーに接続するために使用する、TCP 情報を含むサーバーファイル *(server file)* を作成します。変数`server-auth-dir`は、サーバーファイルを含むデフォルトディレクトリーを指定します。デフォルトでは、これは`~/.emacs.d/server/`です。アクセス権限 (file permission) をもつ local ソケットが存在しない場合は、このディレクトリーのパーミッションにより、どのユーザーの`emacsclient`プロセスが Emacs サーバーと対話 (talk) できるか判断されます。`server-name`が一絶対ファイル名なら、サーバーファイルはそのファイル名で作成されます。

特定の server ファイルにより TCP サーバーに接続するよう`emacsclient`に指示するには、オプション‘`-f`’または‘`--server-file`’を使用するか、環境変数`EMACS_SERVER_FILE`をセットします (Section 31.6.3 [emacsclient Options], page 472 を参照)。`server-auth-dir`が非標準的な値にセットされていたり、`server-name`に絶対ファイル名がセットされていると、デフォルトの`server-auth-dir`は、相対ファイル名を解決するために使用するディレクトリーとして、`emacsclient`内にハードコーディングされているため、server ファイルにたいする絶対ファイル名が必要になります。

31.6.2 `emacsclient`の呼び出し

`emacsclient`プログラムを使う一番簡単な方法は、シェルコマンド‘`emacsclient` *`file`*’を実行する方法です。ここで *file*はファイル名です。これは Emacs サーバーに接続して、Emacs プロセスの既存のフレームの 1 つ— グラフィカルなフレーム、またはテキスト端末のフレーム (Chapter 18 [Frames], page 194 を参照してください) — で *file*を visit するよう指示します。それから、そのフレームを選択して編集を開始できます。

Emacs サーバーが存在しない場合、emacsclientはエラーメッセージと共に終了します (このハプニングによるエラー終了は、emacsclientにたいして‘--alternate-editor=\"\"’オプションを使用して回避できる。Section 31.6.3 [emacsclient Options], page 472 を参照されたい)。Emacs プロセスに既存のフレームがない場合 — これはサーバーがデーモン (Section 31.6 [Emacs Server], page 469 を参照してください) として開始されたときに発生し得ます — は、emacsclientを呼び出した端末で Emacs フレームをオープンします。

オプション‘-c’でグラフィカルなディスプレイ、‘-t’を使用しテキスト端末で新しいフレームを開くように、emacsclientに強制することもできます。Section 31.6.3 [emacsclient Options], page 472 を参照してください。

単一のテキスト端末で実行している場合、emacsclientのシェルと Emacs サーバーを、次の 2 つの方法で切り替えることができます。1 つ目は、Emacs サーバーと、別の仮想端末で emacsclient を実行して、emacsclientを呼び出した後に Emacs サーバーの仮想端末に切り替える方法です。2 つ目は、Emacs サーバー自身から Shell モード (Section 31.5.2 [Interactive Shell], page 459 を参照してください)、または Term モード (Section 31.5.9 [Term Mode], page 468 を参照してください) を使って emacsclientを呼び出す方法です。emacsclientは、Emacs 配下のサブシェルのときだけブロックするので、依然としてファイルの編集に Emacs を使用できます。

Emacs サーバーで *file*の編集を終えたら、そのバッファーで C-x # (server-edit) とタイプします。これはファイルを保存して、emacsclientプログラムに終了を告げるメッセージを送り返します。通常、EDITORを使うプログラムは、何か他のことを行なう前にエディター— この場合は emacsclient — の終了を待ちます。

かわりに編集をキャンセルしたければ、M-x server-edit-abort コマンドを使用します。これは emacsclientプログラムにメッセージを送り返して、バッファーを何も保存せずに abnormal な exit ステータスで exit するよう指示します。

複数のファイル名を引数にして emacsclientを呼び出すこともできます。‘emacsclient *file1 file2 ...*’は、Emacs サーバーに *file1*、*file2*、... を visit するよう指示します。Emacs は *file1*を visit しているバッファーを選択して、他のバッファーをバッファーリストの最後に隠します (Chapter 16 [Buffers], page 176 を参照してください)。指定されたすべてのファイルを終了したら (たとえば各サーバーバッファーで C-x #をタイプしたら)、emacsclientプログラムは終了します。

サーバーバッファーを終了すると、そのバッファーがサーバーバッファーを作成する前から存在していたバッファーでないかぎり、そのバッファーも kill されます。しかし server-kill-new-buffers を nilにセットした場合、別の基準が使用されます。この場合、サーバーバッファーの終了は、ファイル名が正規表現 server-temp-file-regexpにマッチするときは、バッファーを kill します。これはある種の一時ファイルを区別するための仕組みです。

各 C-x #は、さまざまなファイルを編集する、保留されている外部要求が他にないかチェックして、次のそのようなファイルを選択します。サーバーバッファーに辿り着くのに、C-x #だけを使わなければならない訳ではありません。サーバーバッファーに手動で切り替えることもできます。しかし C-x #は、emacsclientに終了したことを告げる方法なのです。

変数 server-windowの値をウィンドウやフレームにした場合、C-x #は常に次のサーバーバッファーを、そのウィンドウまたはそのフレームに表示します。

emacsclientが接続した際には、サーバーは通常はクライアントフレームの exit 方法を告げるメッセージを出力します。server-client-instructionsを nilにセットすると、このメッセージは抑制されます。

31.6.3 emacsclientのオプション

以下のようなオプション引数を emacsclientプログラムに渡すことができます:

```
emacsclient -c +12 file1 +4:3 file2
```

引数‘*+line*’および‘*+line*:*column*’は、その後に続くファイルの行番号、または行番号と列番号を指定します。これらはEmacs自身に対するコマンドライン引数と同じように機能します。Section C.1 [Action Arguments], page 574 を参照してください。

その他の `emacsclient`により認識されるオプション引数を以下にリストします:

‘`-a` *command*’
‘`--alternate-editor=`*command*’
: `emacsclient`が、Emacsとの接続に失敗したとき実行するシェルコマンドを指定します。これはスクリプト内で`emacsclient`を実行するとき便利です。このコマンドには、`\"`like this`\"`のようにクォートされた引数が含まれるかもしれません。現在のところ、クォートのエスケープはサポートされていません。

  特別な例外として、*command*が空文字列の場合、接続に失敗したら`emacsclient`は(‘`emacs --daemon`’のように)Emacsをデーモンモードで開始して、再度接続を試みます。

  環境変数`ALTERNATE_EDITOR`は、‘`-a`’オプションと同じ効果をもちます。両方が指定された場合は、後者のオプションが優先されます。

‘`-c`’
‘`--create-frame`’
: 既存のEmacsフレームを使うかわりに、新しいグラフィカルなクライアントフレーム*(client frame)*を作成します。クライアントフレームでの、`C-x C-c`の特別な振る舞いについては、以下を参照してください。(Xサーバーに接続できない等で)新たにグラフィカルなフレームを作成しない場合、あたかも‘`-t`’オプションが指定されたかのように、テキスト端末でクライアントフレームの作成を試みます。

  MS-Windowsでは、単一のEmacsセッションがグラフィカルなフレームとテキスト端末のフレームの両方を表示することはできず、複数のテキスト端末のフレームを表示することもできません。したがってEmacsサーバーがテキスト端末で実行されている場合、‘`-c`’オプションは‘`-t`’オプションのように、サーバーのカレントテキスト端末で新たなフレームを作成します。Section H.1 [Windows Startup], page 608 を参照してください。

  ‘`-c`’オプションを指定するときに、ファイル名の引数を省略した場合、新しいフレームはデフォルトで`*scratch*`バッファーを表示します。変数`initial-buffer-choice`でこの振る舞いをカスタマイズできます (Section 3.1 [Entering Emacs], page 15 を参照してください)。

‘`-r`’
‘`--reuse-frame`’
: グラフィカルなクライアントフレームが存在しなければ新たに作成、存在する場合には既存のEmacsフレームを使用します。

‘`-F` *alist*’
‘`--frame-parameters=`*alist*’
: 新たに作成される、グラフィカルなフレームのパラメーターをセットします (Section 18.11 [Frame Parameters], page 206 を参照してください)。

‘-d *display*’
‘--display=*display*’
: 与えられたファイルを、(複数の X ディスプレイがあることを想定して)X ディスプレイ *display*で開くよう Emacs に指示します。

‘-e’
‘--eval’ : ファイルを visit するかわりに、Emacs Lisp コードを評価するよう Emacs に指示します。このオプションが与えられた場合、emacsclientは引数を visit するファイルではなく、式のリストと解釈します。

‘-f *server-file*’
‘--server-file=*server-file*’
: TCP を通じて Emacs サーバーに接続するための server ファイル (Section 31.6.1 [TCP Emacs server], page 471 を参照) を指定します。かわりに server ファイルを示すように、環境変数 EMACS_SERVER_FILEをセットできます (コマンドラインオプションは、環境変数をオーバーライドする)。

  Emacs サーバーは通常は接続にたいしてローカルソケットを listen しますが、TCP を通じた接続もサポートします。TCP により Emacs サーバーに接続するためには、emacsclientが Emacs サーバーへの接続に関する詳細を含む *server* ファイル *(server file)* を読み込む必要があります。このオプションで server ファイル名を指定します。これは~/.emacs.d/serverからの相対ファイル名か、絶対ファイル名です。Section 31.6.1 [TCP Emacs server], page 471 を参照してください。

‘-n’
‘--no-wait’
: すべてのサーバーバッファーが終了するのを待つかわりに、emacsclientが即座に終了するようにします。Emacs のサーバーバッファーで編集したいときに、このオプションを指定します。C-x #をタイプしても、これらは kill されません。

‘-w’
‘--timeout=*N*’
: Emacs からの応答を諦めるまで *N*秒待機します。この待機の間に応答がなければ、emacsclientは警告を表示して exit します。デフォルトの ‘0’は永遠に待機することを意味します。

‘--parent-id=*id*’
: XEmbed プロトコルを通じて、ID が *id*の親 X ウィンドウでクライアントフレームとして、emacsclientフレームをオープンします。現在のところ、これは主に開発者に有用なオプションです。

‘-q’
‘--quiet’ : emacsclientが Emacs を待つメッセージ、またはリモートのサーバーソケットに接続するメッセージを表示しないようにします。

‘-u’
‘--suppress-output’
: サーバーからリターンされた結果を、emacsclientが表示しないようにします。これは主に ‘-e’と併用し、評価が結果ではなく副作用を目的としているとき有用です。

‘`-s` *server-name*’
‘`--socket-name=`*server-name*’
: 名前が *server-name*の Emacs サーバーに接続します (このオプションは MS-Windows ではサポートされない)。サーバー名は、Emacs サーバー上の変数 `server-name`により与えられます。このオプションが省略された場合、`emacsclient`はデフォルトソケットに接続します。Emacs サーバーの `server-name`に絶対ファイル名をセットした場合は、`emacsclient`がそのサーバーに接続するよう指示するために、*server-name*に同じ絶対ファイル名を与えてください。Emacs をデーモンとして開始 (Section C.2 [Initial Options], page 576 を参照) して、そのデーモンにより開始されたサーバーに名前を指定した場合には、このオプションを使用する必要があります。

  かわりに server ソケットを示すように、環境変数 `EMACS_SOCKET_NAME`をセットできます (コマンドラインオプションは環境変数をオーバーライドする)。

‘`-t`’
‘`--tty`’
‘`-nw`’
: 既存の Emacs フレームを使うかわりに、カレントテキスト端末に新たなクライアントフレームを作成します。これはテキスト端末のフレームを作成する点を除き、上記で説明した‘`-c`’と同じように振る舞います (Section 18.21 [Non-Window Terminals], page 215 を参照してください)。

  MS-Windows では、Emacs サーバーがグラフィカルなディスプレイを使っている場合は‘`-c`’のように振る舞いますが、Emacs サーバーがテキスト端末で実行されている場合は、カレントテキスト端末に新たなフレームを作成します。

‘`-T` *tramp-prefix*’
‘`--tramp-prefix=`*tramp-prefix*’
: Emacs が TRAMP (*The Tramp Manual* を参照) を使用してリモートマシンにファイルを配すとき (Section 15.15 [Remote Files], page 169 を参照)、ファイル名に追加されるプレフィクスをセットします。これは主に、TCP 上での Emacs サーバーの使用と併用すると有用です (Section 31.6.1 [TCP Emacs server], page 471 を参照)。listen するポートを ssh フォワードして、リモートマシンで *server-file*を利用可能にすることにより、リモートマシン上のプログラムは `EDITOR`、および同種の環境変数にたいする値に `emacsclient`を使用できますが、リモートマシンから Emacs サーバーと通信するかわりに、そのファイルは TRAMP を使用してローカルの Emacs セッションにより visit されます。

  環境変数 `EMACSCLIENT_TRAMP`は、‘`-T`’オプションと同じ効果をもちます。両方が指定された場合は、コマンドラインのオプションが優先されます。

  たとえば 2 つのホスト‘`local`’および‘`remote`’があり、ローカルの Emacs は TCP ポート 12345 を listen しているとします。さらに`/home`が共有ファイルシステム上にあるとすると、server ファイル`~/.emacs.d/server/server`は両方のホストから読み込めることになります。

```
local$ ssh -R12345:localhost:12345 remote
remote$ export EDITOR="emacsclient \
        --server-file=server \
        --tramp=/ssh:remote:"
remote$ $EDITOR /tmp/foo.txt #Should open in local emacs.
```

オプション‘`-c`’または‘`-t`’で作成された、グラフィカルなフレームおよびテキスト端末のフレームは、クライアントフレーム *(client frames)* とみなされます。クライアントフレームから作成した新たなフ

レームも、クライアントフレームとみなされます。クライアントフレームで C-x C-c (save-buffers-kill-terminal) とタイプした場合、それが通常行なうような Emacs セッションの kill(Section 3.2 [Exiting], page 16 を参照してください) は行なわれません。かわりに Emacs はクライアントフレームを削除します。さらに、制御を取り戻すために待っている emacsclientをクライアントフレームがもつ場合 (たとえば '-n'オプションを与えなかった場合)、Emacs は同じくクライアントの他のすべてのフレームを削除し、(それらすべてにたいして C-x #をタイプしたかのように) クライアントのサーバーバッファーが終了したとマークします。クライアントフレームが削除された後、残ったフレームが存在しない場合、Emacs セッションは終了します。

例外として、Emacs がデーモンとして開始された場合、すべてのフレームはクライアントフレームとみなされ、C-x C-cが Emacs を kill することはありません。デーモンセッションを終了するには、M-x kill-emacsとタイプします。

'-t'と '-n'は、矛盾するオプションであることに注意してください。'-t'はカレントテキスト端末に新たにクライアントフレームを作成して制御することを指示し、'-n'はテキスト端末で制御を行なわないことを指示するからです。両方のオプションを与えた場合、'-t'の効果は打ち消されて、Emacs は新たにフレームを作成せず、既存のフレームで指定されたファイルを visit します。

31.7 ハードコピーの印刷

Emacs は、バッファーの全体、または一部のハードコピーを印刷するコマンドを提供します。以下で詳しく述べるとおり、直接印刷コマンドを呼び出したり、メニューバーの 'File'メニューを使うことができます。

このセクションで説明するコマンドとは別に、Dired(Section 27.7 [Operating on Files], page 387 を参照してください) からハードコピーを印刷したり、ダイアリー (Section 28.10.2 [Displaying the Diary], page 412 を参照してください) から印刷することもできます。コマンド M-x htmlfontify-bufferで、Emacs バッファーを HTML に "印刷" することもできます。これは Emacs のフェイスを CSS ベースのマークアップに置換して、カレントバッファーを HTML に変換します。さらに Org モード (Section 22.10 [Org Mode], page 267 を参照してください) では、Org ファイルを PDF のような様々なフォーマットに印刷できます。

M-x print-buffer
: ファイル名を含むページヘッダーとともに、カレントバッファーのハードコピーを印刷します。

M-x lpr-buffer
: ページヘッダーなしで、カレントバッファーのハードコピーを印刷します。

M-x print-region
: print-bufferと同様ですが、カレントリージョンだけを印刷します。

M-x lpr-region
: lpr-bufferと同様ですが、カレントリージョンだけを印刷します。

ほとんどのオペレーティングシステムでは、上記のハードコピーコマンドは、lprプログラムを呼び出して、印刷のためにファイルをサブミットします。印刷プログラムを変更するには、変数 lpr-commandをカスタマイズします。追加のスイッチを指定して、印刷プログラムにに与えるには、リスト変数 lpr-switchesをカスタマイズします。この変数の値はオプション文字列のリストで、それぞれが '-' で始まります (たとえばオプション文字列"-w80"は 1 行に 80 列を指定します)。デフォルトは、空リスト nilです。

使用するプリンターを指定するには、変数 printer-nameをセットします。デフォルトは nilで、これはデフォルトプリンターを指定します。これにプリンター名 (文字列) をセットした場合、

その名前は '-P'スイッチとともに lprに渡されます。lprコマンドを使用しない場合は、スイッチを lpr-printer-switchで指定する必要があります。

同様に変数 lpr-headers-switchesは、ページヘッダーを作成するために使用する追加のスイッチを指定します。変数 lpr-add-switchesは、印刷プログラムに '-T'および '-J'オプション (lpr向きのオプションです) を渡すかどうかを制御します。nilは、これらを追加しないことを意味します (印刷プログラムが lpr) 互換でない場合は、この値を使用するべきです)。

31.7.1 PostScript のハードコピー

これらのコマンドはバッファーの内容を PostScript に変換して、それを印刷または別の Emacs バッファーに出力します。

M-x ps-print-buffer
: PostScript 形式で、カレントバッファーのハードコピーを印刷します。

M-x ps-print-region
: PostScript 形式で、カレントリージョンのハードコピーを印刷します。

M-x ps-print-buffer-with-faces
: PostScript 形式で、カレントバッファーのハードコピーを印刷し、テキストで使用されているフェイスは PostScript 機能により表示します。

M-x ps-print-region-with-faces
: PostScript 形式で、カレントリージョンのハードコピーを印刷し、テキストで使用されているフェイスで表示します。

M-x ps-spool-buffer
: カレントバッファーのテキストにたいする PostScript イメージを生成およびスプールします。

M-x ps-spool-region
: カレントリージョンにたいする PostScript イメージを生成およびスプールします。

M-x ps-spool-buffer-with-faces
: カレントバッファーにたいする PostScript イメージを生成およびスプールし、使用されているフェイスで表示します。

M-x ps-spool-region-with-faces
: カレントリージョンにたいする PostScript イメージを生成およびスプールし、使用されているフェイスで表示します。

M-x ps-despool
: Send the spooled PostScript to the printer.

M-x handwrite
: カレントバッファーにたいして、手書きされたような PostScript を生成・印刷します。

ps-print-bufferおよび ps-print-regionコマンドは、バッファーの内容を PostScript 形式で印刷します。一方はバッファー全体を印刷し、もう一方はリージョンだけを印刷します。コマンド ps-print-buffer-with-facesおよび ps-print-region-with-facesも同様に振る舞いますが、バッファーテキストのフェイス (フォントとカラー) を表示するために PostScript 機能を使用します。

プレフィクス引数 (C-u) を使用した場合、これらのコマンドは対話的にユーザーにファイル名の入力を求め、PostScript イメージをプリンターに送るかわりに、そのファイルに保存します。

‘print’のかわりに‘spool’が名前につくコマンドは、PostScript 出力をプリンターに送るかわりに、Emacs バッファーに出力することを意味します。

コマンド ps-despoolを使用して、スプールされたイメージをプリンターに送ります。このコマンドは‘-spool-’コマンド (上記コマンド参照) により生成された PostScript をプリンターに送ります。プレフィクス引数 (C-u) を指定した場合は、ファイル名の入力を求め、スプールされた PostScript イメージをプリンターに送るかわりに、そのファイルに保存します。

M-x handwriteはもっと他愛ない機能です。これはカレントバッファーの PostScript 表現を、草書体の手書きドキュメントのように生成します。これはグループ handwriteでカスタマイズできます。この関数は ISO 8859-1 文字だけサポートします。

31.7.2 PostScript ハードコピーにたいする変数

すべての PostScript ハードコピーコマンドは、どのように出力を印刷するかの指定に、変数 ps-lpr-commandと ps-lpr-switchesを使用します。ps-lpr-commandは実行するコマンド名、ps-lpr-switchesは使用するコマンドラインオプション、ps-printer-nameはプリンターを指定します。最初の 2 つの変数をセットしない場合、変数の初期値を lpr-commandと lpr-switchesから取得します。ps-printer-nameが nilの場合、printer-nameが使用されます。

変数 ps-print-headerは、これらのコマンドが各ページにヘッダー行を追加するかを制御します。これを nilにセットするとヘッダーはオフになります。

プリンターがカラーをサポートしない場合は、ps-print-color-pを nilにセットして、カラー処理をオフにするべきです。デフォルトでは、ディスプレイがカラーをサポートしていれば、Emacs はカラー情報をもったハードコピー出力を生成します。モノクロプリンターでは、カラーはグレーの濃淡でエミュレートされます。スクリーンカラーがグレーの濃淡だけしか使用していなくても、これはほとんど読み取れないか、読みにくい出力を生成するかもしれません。

モノクロプリンターでより良いカラー表示を得るために、ps-print-color-pに black-white をセットすることもできます。これは bold と italic のフェイス属性で補強された、カスタマイズ可能なグレースケールのリストによりカラーを表現するための ps-black-white-facesお情報を使用することにより機能します。

デフォルトでは、変数 ps-use-face-backgroundが非 nilでなければ、PostScript 印刷はフェイスのバックグラウンドカラーは無視します。これは望ましくないゼブラストライプや、バックグラウンドイメージとテキストの干渉を避けるためです。

変数 ps-paper-typeは、フォーマットする用紙サイズを指定します。妥当な値には a4、a3、a4small、b4、b5、executive、ledger、legal、letter、letter-small、statement、tabloidが含まれます。デフォルトは letterです。変数 ps-page-dimensions-databaseを変更することにより、追加の用紙サイズを定義できます。

変数 ps-landscape-modeは、ページの印刷向きを指定します。デフォルトは nilで、これは縦向き印刷モード (portrait mode) を意味します。非 nil値は横向き印刷モード (landscape mode) を指定します。

変数は列番号を指定します。この変数は横向きモードと縦向きモードの両方で効果をもちます。デフォルトは 1 です。

変数 ps-font-familyは、通常のテキストを印刷するために使用するフォントファミリーを指定します。妥当な値には、Courier、Helvetica、NewCenturySchlbk、Palatino、Timesが含まれます。変数 ps-font-sizeは、通常のテキストのためのフォントサイズを指定し、デフォルトは 8.5 ポイントです。ps-font-sizeの値には、2 つの浮動小数点によるコンスでも指定できます。その場合、1 つは縦向き印刷モード、もう一方は横向き印刷モードでの指定です。

Emacs は、通常の PostScript プリンターより多くのスクリプトと文字をサポートします。したがってバッファーのいくつかの文字は、プリンターに組み込まれたフォントを使って印刷できないかもしれません。プリンターが提供するフォントを GNU Intlfonts パッケージのフォントで増強したり、Emacs に Intlfonts フォントだけを使うように命令できます。変数 `ps-multibyte-buffer`がこれを制御します。デフォルト値の `nil`は、ASCIIおよび Latin-1 文字を印刷するのに適しています。`non-latin-printer`は、プリンターに ASCII、Latin-1、Japanese、Korean 文字にたいするフォントが組み込まれている場合の値です。すべての文字にたいして使用される、Intlfonts パッケージの BDF フォントのための値が `bdf-font`です。最後に値 `bdf-font-except-latin`は、ASCIIおよび Latin-1 文字にたいしては組み込みフォントを使用し、それ以外の文字にたいしては Intlfonts の BDF フォントを使用するよう指示します。

BDF フォントを使えるようにするには、それをどこで見つけることができるかを、Emacs が知る必要があります。変数 `bdf-directory-list`は、Emacs がこれらのフォントを探すべきディレクトリーのリストを保持します。デフォルト値には、1 つのディレクトリー `/usr/local/share/emacs/fonts/bdf`が含まれます。

これらのコマンドに対する、その他多くのカスタマイズ変数は、Lisp ファイル `ps-print.el`および `ps-mule.el`で定義・説明されています。

31.7.3 印刷のためのパッケージ

ハードコピーを印刷するための Emacs の基本的な機能は、Printing パッケージを使って拡張できます。これは何を印刷するかの選択、印刷前の PostScript ファイルのプレビュー、プリントヘッダーなどのさまざまな印刷オプションの設定、横向きモードや縦向きモード、duplex モードにたいする使いやすいインターフェースを提供します。GNU/Linux システムまたは Unix システムでは、Printing パッケージは、GhostScript プログラムの一部として配布される、`gs`および `gv`ユーティリティーに依存しています。MS-Windows では、Ghostscript のポート版の `gstools`が使用されます。

Printing パッケージを使用するには、init ファイル (Section 33.4 [Init File], page 528 を参照してください) に `(require 'printing)`を追加して、その後に `(pr-update-menus)`と記述します。この関数はメニューバーの通常の印刷コマンドを、さまざまな印刷オプションを含む '`Printing`'サブメニューに置き換えます。`M-x pr-interface RET`とタイプすることもできます。これはカスタマイズバッファーに似た`*Printing Interface*`バッファーを作成し、そこで印刷オプションをセットできます。何をどのようにして印刷するかセットした後、'`Print`'ボタン (それを `mouse-2`でクリックするか、その上にポイントを移動して `RET`をタイプします) で印刷ジョブを開始します。さまざまなオプションについての詳細は、'`Interface Help`'ボタンを使用してください。

31.8 テキストのソート

Emacs は、バッファーのテキストをソートするためのコマンドをいくつか提供しています。これらはすべて、リージョンの内容にたいして処理を行ないます。これらのコマンドはリージョンのテキストを、多数のソートレコード *(sort records)* に分割し、各レコードをソートキー *(sort key)* で識別した後、ソートキーにより決定される順序にレコードを並び替えます。レコードは並び替えられ、レコードのキーはアルファベット順、または数値ソートでは数値順になります。アルファベット順のソートでは、ASCII文字順にしたがい、'`A`'から '`Z`'のすべての大文字は、小文字の '`a`'の前になります。(しかし、以下で説明する `sort-fold-case`はこれを変更する)。

種々のソートコマンドは、テキストをどのようにソートレコードに分割するか、そして各レコードのどの部分がソートキーとして使用されるかという点で異なります。ほとんどのコマンドは行ごとにソートレコードを分割しますが、ソートレコードとしてパラグラフやページを使うコマンドもありま

す。ほとんどのソートコマンドは、ソートレコード全体をソートキーとして使用しますが、レコードの一部をソートキーとして使用するものもあります。

`M-x sort-lines`
: リージョンを行に分割して、行のテキスト全体を比較してソートします。数引数は降順でソートすることを意味します。

`M-x sort-paragraphs`
: リージョンをパラグラフに分割して、パラグラフのテキスト全体 (先頭の空行は除く) を比較してソートします。数引数は降順にソートすることを意味します。

`M-x sort-pages`
: リージョンをページに分割して、ページのテキスト全体 (先頭の空行は除く) を比較してソートします。数引数は降順にソートすることを意味します。

`M-x sort-fields`
: リージョンを行に分割して、各行のあるフィールドの内容を比較することによりソートします。フィールドは空白で区切ることにより定義されるので、ある行の最初の連続する非空白文字がフィールド 1 を構成し、2 番目のそのような文字構成がフィールド 2、... となります。

  どのフィールドでソートするかは数引数で指定します。1 はフィールド 1、2 はフィールド 2、... となります。デフォルトは 1 です。負の引数はフィールドを左からではなく右から数えることを意味します。したがって、-1 は最後のフィールドでソートするという意味です。ソートされるフィールドの内容が同じ行が複数存在する場合、バッファーの元の並び順が維持されます。

`M-x sort-numeric-fields`
: `M-x sort-fields`と同様ですが、各行の指定されたフィールドを整数に変換して、その数字を比較します。テキストとして比較したとき '10'は '2'の前になりますが、数字として比較すると '2'の後になります。デフォルトでは、数字は `sort-numeric-base`により解釈されますが、'0x'または '0'で始まる数字は 16 進および 8 進で解釈されます。

`M-x sort-columns`
: `M-x sort-fields`と同様ですが、各行で比較に使用されるテキストは固定長の列範囲です。プレフィクス引数を指定した場合は、逆順でソートします。このコマンドの詳細は以下を参照してください。

`M-x reverse-region`
: リージョンの行を逆順にソートします。フィールドでソートするコマンドは降順でソートする機能がないので、フィールドまたは列で降順にソートできるのは便利です。

たとえば、以下のようなバッファー内容のとき:

```
On systems where clash detection (locking of files being edited) is
implemented, Emacs also checks the first time you modify a buffer
whether the file has changed on disk since it was last visited or
saved.  If it has, you are asked to confirm that you want to change
the buffer.
```

バッファー全体に `M-x sort-lines`を適用すると以下のようになります:

```
On systems where clash detection (locking of files being edited) is
implemented, Emacs also checks the first time you modify a buffer
saved.  If it has, you are asked to confirm that you want to change
the buffer.
```

```
whether the file has changed on disk since it was last visited or
```

大文字の‘O’はすべての小文字の前にソートされます。かわりに C-u 2 M-x sort-fieldsを使用すると、以下が得られます:

```
implemented, Emacs also checks the first time you modify a buffer
saved.  If it has, you are asked to confirm that you want to change
the buffer.
On systems where clash detection (locking of files being edited) is
whether the file has changed on disk since it was last visited or
```

ここで、ソートキーは‘Emacs’、‘If’、‘buffer’、‘systems’、‘the’です。

M-x sort-columnsは説明が必要でしょう。ある列にポイントを配し、別の列をマークして、ソート列を指定します。ソートしたいテキストの最初の行の先頭にポイントまたはマークを配すことはできないので、このコマンドは特殊な “リージョン” の定義を使用します。ポイントがある行はすべてリージョンの一部と判断されます。マークがある行もすべてリージョンの一部と判断されます。同様にその間にあるすべての行もリージョンの一部となります。

たとえば、10 列目から 15 列目までの情報でテーブルをソートするには、テーブルの最初の行の 10 列目にマークを置き、最後の行の 15 列目にポイントを置いて、sort-columnsを実行します。最初の行の 15 列目にマークを置き、最後の行の 10 列目にポイントを置いても、同じように実行できます。

これはポイントとマークで指定された矩形領域をソートすると考えることができます。1 つ違うのは、各行の矩形領域の左右にあるテキストも、矩形領域内のテキストとともに移動するという点です。Section 9.5 [Rectangles], page 68 を参照してください。

sort-fold-caseが非 nilの場合、多くのソートコマンドは比較時の大文字小文字の違いを無視します。

31.9 バイナリーファイルの編集

バイナリーファイルを編集する特別なメジャーモードに、Hexl モードがあります。これを使用するには、ファイルを visit するために C-x C-fのかわりに、M-x hexl-find-fileを使用します。このコマンドはファイル内容を 16 進に変換して、その変換結果を編集できるようにします。ファイルを保存するときは、自動的にバイナリーに変換されます。

既存のバッファーを 16 進 (hex) に変換するために、M-x hexl-modeを使用することもできます。これは普通にファイルを visit してから、それがバイナリーファイルだと気づいた場合などに便利です。

Hexl モードでは、テキスト挿入は常に上書きとなります。これは、アクシデントによりファイル内データのアラインメントを破壊する危険を軽減します。通常のテキスト文字は、それらの文字自身ををを挿入 (それらの文字で上書き) します。文字コードにより特殊文字を挿入するコマンドあります。Hexl モードでは、ほとんどのカーソル移動キー、同様に C-x C-sは、同じ効果を生むコマンドにバインドされています。以下は、特に Hexl モードで重要なその他コマンドです:

C-M-d　10 進でタイプされたバイトコードを挿入します。

C-M-o　8 進でタイプされたバイトコードを挿入します。

C-M-x　16 進でタイプされたバイトコードを挿入します。

C-M-a　512 バイトページの先頭に移動します。

C-M-e　512 バイトページの最後に移動します。

C-x [　1K バイトページの先頭に移動します。

C-x]　1K バイトページの最後に移動します。

`M-g` 16 進で指定されたアドレスに移動します。

`M-j` 10 進で指定されたアドレスに移動します。

`C-c C-c` Hexl モードを抜けて、`hexl-mode`モードを呼び出す前の、このバッファーのメジャーモードに戻ります。

他の Hexl コマンドには、バイナリーバイトの文字列 (シーケンス) の挿入、`short`、`int`単位での移動などがあります。詳細については、`C-h a hexl- TAB`とタイプしてください。

Hexl モードをテキストファイルの編集にも使えます。これはテキストファイルに変わった文字が含まれていたり、変わったエンコーディング (Section 19.5 [Coding Systems], page 223 を参照) が使用されている場合に便利かもしれません。この目的用にバイトを挿入する Hexl コマンドは ASCII 文字、およびマルチバイト文字を含む非 ASCII文字も挿入できます。Hexl でテキストファイルを編集するには通常通りにファイルを visit してから、`M-x hexl-mode RET` とタイプして Hexl モードに切り替えます。これでテキスト文字をタイプして挿入することができます。しかし無効なマルチバイトシーケンスを作成する危険を避けるために、マルチバイト文字の挿入には特別な配慮が必要です。ファイル中のマルチバイトシーケンスの開始バイトにポイントがあるときに、そのような文字のタイプを開始する必要があります。

31.10 Emacs セッションの保存

デスクトップライブラリーを使用して、あるセッションから別のセッションに Emacs の状態を保存することができます。保存される Emacs のデスクトップ構成 *(desktop configuration)* にはバッファーとそのファイル名、メジャーモード、バッファー位置、ウィンドウおよびフレームの構成、それにいくつかの重要なグローバル変数が含まれます。

この機能を有効にするには Customization バッファー (Section 33.1 [Easy Customization], page 499 を参照) を使用して `desktop-save-mode`を `t`にセットするか、init ファイル (Section 33.4 [Init File], page 528 を参照) に以下の行を追加します:

```
(desktop-save-mode 1)
```

init ファイルで `desktop-save-mode`をオンにしていれば、Emacs が起動時に `desktop-path` (デフォルトはまず `user-emacs-directory`、次にホームディレクトリー) から保存されたデスクトップを探して、見つかった最初のデスクトップを使用します。`desktop-save-mode`がオンの状態で Emacs が実行されている間は、デスクトップ構成の変更時は常にデスクトップが自動的に保存されます。Emacs がデスクトップの変更をチェックする頻度は、変数 `desktop-auto-save-timeout` によって決定されます。デスクトップは Emacs を exit する際にも保存されます。

保存されたデスクトップ構成をリロードしたくないときは、Emacs を実行するコマンドラインでオプション '`--no-desktop`'を指定します。これはカレントセッションにたいして、`desktop-save-mode`をオフにします。'`--no-init-file`'オプションを指定して Emacs を開始することにより、通常 `desktop-save-mode`をオンにしている init ファイルをバイパスして、デスクトップのリロードを無効にすることもできます。

別のディレクトリーに別個にデスクトップ構成を保存することができます。`desktop-path`にセットされているディレクトリーの前に `.` (カレントディレクトリー) を追加するようカスタマイズすることによって、保存済みのデスクトップ構成のあるディレクトリーから Emacs を起動した際にその構成がリストアされるようになります。`M-x desktop-change-dir`とタイプすれば、カレントのデスクトップを保存して別のディレクトリーに保存されているデスクトップをリストアできます。前にリロードしたデスクトップをリバートするには `M-x desktop-revert`とタイプしてください。

Emacs がデスクトップを保存するファイルは、他の Emacs セッションによる意図せぬ上書きを防ぐためにそのセッションの実行中はロックされます。このロックは通常は Emacs の exit 時に解除

されますが、Emacs またはシステムがクラッシュするとそのロックが残ってしまい、デフォルトでは Emacs の再起動時にロックされたデスクトップファイルのどちらを使用するか尋ねます。この質問は変数 `desktop-load-locked-desktop`を `nil` (デスクトップをロードしない)、または `t` (確認なしでデスクトップをロードする) のいずれかにカスタマイズすることにより抑止できます。`check-pid` という特別な値にカスタマイズすることもできます。これはローカルマシン上でそのデスクトップをロックした Emacs が実行されていなければそのファイルをロードすることを意味します。この値はロックした Emacs が別のマシンでまだ実行中という状況で使用するべきではありません。マルチユーザー環境において、NFS 等を用いてホームディレクトリーがリモートにマウントされているのかもしれません。

Emacs がデーモンモードで起動する際にはユーザーに質問することができないので、ロックされたデスクトップが見つかっても `desktop-load-locked-desktop`が `t`以外であればロードしません。デーモンモードでのデスクトップのリストアは、その他の理由により問題になりがちなことに注意してください。たとえばデーモンは GUI 機能を使用できないので、フレーム位置やサイズ、装飾のようなパラメーターはリストアできません。この理由により `server-after-make-frame-hook`にフック関数 `desktop-read` (下記参照) を追加して、それが呼び出されることにより最初のクライアントが接続するまで、デスクトップのリストアを遅延させたいと思うかもしれません (Section “Creating Frames” in *The Emacs Lisp Reference Manual* を参照)。

カレントデスクトップを即座に強制的に保存したければ、任意のタイミングでコマンド `M-x desktop-save`を使うことができます。これはデスクトップを自動リストアしたくないので `desktop-save-mode`をオンにしていない、あるいはデスクトップに重要な変更を施し、その構成を Emacs やシステムのクラッシュで失わないよう確実にしたい場合に役に立ちます。Emacs のカレントセッションがまだデスクトップを何もロードしていなければ、`M-x desktop-read`を使用して以前に保存したデスクトップをリストアできます。

デフォルトでは、デスクトップはフレームとウィンドウの構成の保存とリストアを試みます。これを無効にするには、`desktop-restore-frames`を `nil`にセットしてください (この振る舞いを調整するためにマスタマイズできる関連オプションについては変数のドキュメントを参照のこと)。

desktop がフレームとウィンドウの設定をリストアするときは、フレームパラメーターの記録された値を使用します。init ファイル (Section 33.4 [Init File], page 528 を参照) でそれらのパラメーターに何をセットしていても無視されます。これは、リストアされたフレームにたいするフォントやフェイスのようなフレームパラメーターは、前の Emacs セッションを終了したときに保存された desktop ファイルから取得され、init ファイルでのそれらのパラメーターにたいするセッティングは無視されることを意味します。これを無効にするには、リストアしたくないフレームパラメーターを除外するように、`frameset-filter-alist`の値をカスタマイズしてください。その後は init ファイルで行ったカスタマイズに応じてパラメーターがセットされる筈です。

デフォルトでは、リモートファイルを visit しているバッファーに関する情報は保存されません。変数 `desktop-files-not-to-save`をカスタマイズして、これを変更することができます。

デフォルトでは、デスクトップのすべてのバッファーは 1 度に復元されます。しかし、デスクトップにたくさんのバッファーがあるときは遅くなるかもしれません。変数 `desktop-restore-eager` で、即座に復元するバッファーの最大数を指定できます。残りのバッファーは Emacs のアイドル時に、ゆっくり (lazily) と復元されます。

Emacs のデスクトップを空にするには、`M-x desktop-clear`とタイプします。これはたとえば次に `M-x desktop-read`を呼び出して別のデスクトップに切り替えたい場合に役に立つかもしれません。`desktop-clear`コマンドは内部バッファーを除くすべてのバッファーを kill して、`desktop-globals-to-clear`にリストされたグローバル変数をクリアーします。特定のバッファー

を残したい場合は、変数 desktop-clear-preserve-buffers-regexpをカスタマイズします。この変数の値には、kill しないバッファーの名前にマッチする正規表現を指定します。

あるセッションから別のセッションへ、ミニバッファーのヒストリーを保存したい場合は、savehist ライブラリーを使用してください。

31.11 再帰編集レベル

再帰編集 *(recursive edit)* とは、ある Emacs コマンドの途中で、別の Emacs コマンドを使用して自由に編集を行なうような状況を指します。たとえば query-replaceの途中で C-rをタイプすると、カレントバッファーを変更することができる再帰編集に入ります。再帰編集から抜けると、query-replace に戻ります。Section 12.10.4 [Query Replace], page 124 を参照してください。

再帰編集を抜ける *(exit)* とは、実行を継続中の、終了していないコマンドに戻ることを意味します。再帰編集を抜けるコマンドは C-M-c (exit-recursive-edit) です。

再帰編集を中断 *(abort)* することもできます。これは exit と似ていますが、終了していないコマンドも即座に終了します。これを行なうには、コマンド C-] (abort-recursive-edit) を使用します。Section 34.1 [Quitting], page 536 を参照してください

モードラインで常にメジャーモードとマイナーモードの周りを囲む丸カッコ (parentheses) を、さらに角カッコ (square brackets) で囲んで表示することにより、再帰編集中であることが示されます。再帰編集は特定のウィンドウやバッファーにたいするものではなく、Emacs 全体が再帰編集中であるため、すべてのウィンドウのモードラインは同じように表示されます。

再帰編集中に、さらに再帰編集に入ることも可能です。たとえば query-replaceの途中で C-rとタイプした後、デバッガーに入るコマンドをタイプしたとします。これは C-rにたいする再帰編集レベルから、デバッガーにたいする再帰編集を開始します。モードラインには、現在進行中の再帰編集レベルが角カッコのペアで表示されます。

(デバッガーでの cコマンドのように) 内側の再帰編集を抜けると、次に上のレベルのコマンドの実行が再開されます。コマンドが終了したら、他の再帰編集レベルに抜けるために C-M-cを使用することができます。exit は最内レベルだけに適用されます。abort も 1 レベルの再帰編集だけを抜けて、前の再帰編集レベルのコマンドに即座に戻ります。もし望むなら次の再帰編集レベルも abort できます。

かわりにコマンド M-x top-levelは、すべてのレベルの再帰編集を abort して、即座にトップレベルのコマンドリーダーに戻ります。ミニバッファーがアクティブなときは、ミニバッファーも抜けます。

再帰編集の中で編集されるテキストは、トップレベルで編集しているテキストと同じである必要はありません。これは、何にたいしての再帰編集かに依存します。再帰編集を呼び出したコマンドが最初に別のバッファーを選択する場合、そのバッファーが再帰編集を行なうバッファーになります。どんな場合でも再帰編集中に、(バッファーを切り替えるキーがリバインドされていないかぎり) 通常の方法でバッファーを切り替えることができます。再帰編集の中で、ファイルを visit したりその他のことを行ない、残りのすべての編集を行なうことも、おそらく可能です。しかしこれは、(スタックオーバーフローのような) 驚くべき効果をもたらすことがあります。そのため必要なくなったときは、再帰編集を exit または abort することを忘れないでください。

一般的に、わたしたちは GNU Emacs では再帰編集レベルを最小限にしようと努めています。これは、特定の順 — 最内のレベルからトップレベルに向かって — で戻ることをあなたに強いるからです。可能な場合は、異なる作業には別のバッファーを供し、それらを切り替えられるようにします。いくつかのコマンドは、元のメジャーモードに戻るコマンドを提供する、新しいメジャーモードに切り替えます。これらの試みは、あなたの選択にしたがい、まだ終了していないタスクにもどる、柔軟性を与えます。

31.12 ハイパーリンクと Web ナビゲーション機能

以下のサブセクションでは、URL や、Emacs バッファーのテキスト内にある他の種類のリンクを扱う便利な機能を説明します。

31.12.1 EWW によるウェブブラウズ

EWW(Emacs Web Wowser) は、Emacs 用のウェブブラウザーのパッケージです。これは Emacs バッファーで URL ブラウズすることを可能にします。コマンド `M-x eww`により、URL を開いたり、ウェブを検索します。コマンド `M-x eww-open-file`を使用して、ファイルを開くことができます。`browse-url`にたいするウェブブラウザーとして。EWW を使うことができます (Section 31.12.3 [Browse-URL], page 485 を参照してください)。完全な詳細に付いては、*The Emacs Web Wowser Manual* を参照してください。

31.12.2 埋め込み WebKit ウィジェット

Emacs が適切なサポートパッケージとともにコンパイルされている場合、Emacs のバッファーにブラウザーウィジェット (browser widgets) を表示できます。コマンド `M-x xwidget-webkit-browse-url` は、ブラウザーウィジェットに表示する URL を尋ねます。通常はポイント位置、またはポイント位置の前にある URL がデフォルトの URL になります。しかしアクティブなリージョン (Chapter 8 [Mark], page 52 を参照してください) がある場合は、余分な空白文字を削除してから、そのリージョンからデフォルトの URL を取得します。コマンドはその後、指定された URL を表示するめ込みブラウザー (embedded browser) をもつ、新しいバッファーを作成します。このパッファーは Xwidget-WebKit モード (Image モードと似ています。see Section 15.19 [Image Mode], page 172 を参照してください) で、ウィジェットのスクロール、サイズ変更、リロードを 1-key で行なうコマンドを提供します。キーバインドを確認するには、そのバッファーで `C-h b` とタイプしてください。

デフォルトでは xwidget の webkit バッファー内部で自己挿入文字をタイプしても何も行われないか、あるいは何か特別なアクションがトリガーされます。これらの自己挿入文字やその他の一般的な編集キーが押されたらその文字が挿入されるようにするために、それらの文字を WebKit xwidget に渡すように再定義する `xwidget-webkit-edit-mode`を有効にできます。

xwidget webkit バッファー内部で `e`をタイプして `xwidget-webkit-edit-mode`を有効にすることもできます。

`xwidget-webkit-isearch-mode`はインクリメンタル検索 (Section 12.1 [Incremental Search], page 105 を参照) のように振る舞いますが、カレントバッファーではなく WebKit widget のコンテンツを処理するマイナーモードです。xwidget-webkit バッファーの内部では `C-s`と `C-r`にバインドされています。`C-r`で呼び出されると初期の検索方向は逆向きに行われます。

任意の自己挿入文字列をタイプすることで、その文字がカレントけの問い合わせつき検索に挿入されます。WebKit ウィジェットは `C-s`とタイプすれば次の検索結果、`C-r`なら前の検索結果を表示します。

インクリメンタル検索を終えるには `C-g`をタイプしてください。

`xwidget-webkit-browse-history`はカレントの WebKit バッファーが以前にロードしたページを含むバッファーを表示して、`RET`を押してそれらのページに移動するためのコマンドです。

このコマンドは `H`にバインドされています。

31.12.3 URL のフォロー

`M-x browse-url RET` *`url`* `RET`
: ウェブブラウザーへの URL のロード。

Browse-URL パッケージは、Emacs で簡単に URL をフォロー (辿る) することを可能にします。ほとんどの URL は、ウェブブラウザーを呼び出すことによりフォローされます。‘mailto:’の URL は、指定されたアドレスにメールを送るために、Emacs コマンドの compose-mailでフォローされます。

コマンド M-x browse-urlは、URL の入力を求め、それをフォローします。ポイントが URL のようなテキストの近くにある場合、その URL がデフォルトとして提示されます。Browse-URL パッケージは、browse-url-at-pointや browse-url-at-mouseのような、キーにバインドしたいと思うような、他のコマンドも提供します。

Customize グループ browse-urlの、さまざまなオプションを通じて、Browse-URL の振る舞いをカスタマイズできます。特にオプション browse-url-mailto-functionでは‘mailto:’の URL をフォローする方法の定義、browse-url-browser-functionではデフォルトブラウザを指定します。詳細は C-h P browse-url RETとタイプしてパッケージのコメントを参照してください。

browse-url-handlersをカスタマイズすることにより、特定の URL を他の関数で閲覧するように定義できます。これは正規表現か述語、およびマッチした URL をブラウズするための関数のペアからなる alist です。

より詳細な情報は C-h P browse-url RETとタイプしてパッケージのコメントを参照してください。

Emacs には URLをファイルであるかのように処理するためのサポートをもつマイナーモードもあります。url-handler-modeはファイル名を扱う Emacs のほとんどのコマンドとプリミティブに効果があります。このモードに切り替え後は、たとえばウェブページの HTMLを閲覧するために C-x C-f https://www.gnu.org/ RETとタイプして、それを編集してローカルファイルに保存することができます。

31.12.4 URL のアクティブ化

M-x goto-address-mode
: カレントバッファーの URL とメールアドレスをアクティブにします。

M-x global-goto-address-mode
: すべてのバッファーで goto-address-modeをアクティブにします。

M-x goto-address-modeとタイプすることにより、Emacs にカレントバッファーの URL を特別にマークさせることができます。このバッファーローカルなマイナーモードが有効な場合、バッファーのすべての URL を探して、それらをハイライトするとともに、クリックできるボタンに変更します。そのようなテキストの上にポイントを移動して C-c RET (goto-address-at-point) とタイプするか、mouse-2をクリック、または mouse-1を素早くクリックすることにより (Section 18.3 [Mouse References], page 197 を参照してください)、その URL をフォローできます。URL のフォローは、browse-urlをサブルーチンとして呼び出すことにより行なわれます (Section 31.12.3 [Browse-URL], page 485 を参照してください)。

モードフックや受信メッセージを表示するフック (たとえば Rmail の rmail-show-message-hook) に goto-address-modeを追加するのは便利かもしれません。Gnus や MH-E には独自に類似の機能があるので必要ありません。

31.12.5 ポイント位置のファイルや URL を開く

FFAP パッケージは、C-x C-fのようなファイルを探すためのキーにバインドされているコマンドを、より直感的なデフォルトを提供するコマンドに置き換えます。これらのコマンドにプレフィクス引数を与えたときは、通常のコマンドと同様に振る舞います。それ以外の場合、ポイント周辺のテ

キストからデフォルトのファイル名を取得します。バッファーから見つかったのがファイル名ではなくURLの場合、このコマンドはそれを閲覧するために`browse-url`を使用します (Section 31.12.3 [Browse-URL], page 485 を参照してください)。

この機能は、メールバッファーやニュースバッファー内の参照、`README`ファイル、`MANIFEST`ファイルなどをフォローするのに便利です。詳細については、`C-h P ffap RET`とタイプして、パッケージのコメントを参照してください。

FFAPを有効にするには、`M-x ffap-bindings`とタイプします。これにより、以下のキーバインドが作成され、Rmail、Gnus、VMアーティクルバッファーでの、追加FFAP機能にたいするフックもインストールされます。

`C-x C-f` *filename* `RET`
: *filename*を検索します (`find-file-at-point`)。デフォルトのファイル名は、ポイント周辺のテキストから推測します。

`C-x C-r` *filename* `RET`
: `ffap-read-only`。`find-file-read-only`に相当します。

`C-x C-v` *filename* `RET`
: `ffap-alternate-file`。`find-alternate-file`に相当します。

`C-x d` *directory* `RET`
: ポイント位置のディレクトリーをデフォルトとして、*directory*でDiredを開始します (`dired-at-point`)。

`C-x C-d` *directory* `RET`
: `ffap-list-directory`。`list-directory`に相当します。

`C-x 4 f` *filename* `RET`
: `ffap-other-window`。`find-file-other-window`に相当します。

`C-x 4 r` *filename* `RET`
: `ffap-read-only-other-window`。`find-file-read-only-other-window`に相当します。

`C-x 4 d` *directory* `RET`
: `ffap-dired-other-window`。`dired-other-window`と同様です。

`C-x 5 f` *filename* `RET`
: `ffap-other-frame`。`find-file-other-frame`に相当します。

`C-x 5 r` *filename* `RET`
: `ffap-read-only-other-frame`。`find-file-read-only-other-frame`に相当します。

`C-x 5 d` *directory* `RET`
: `ffap-dired-other-frame`。`dired-other-frame`に相当します。

`C-x t C-f` *filename* `return`
: `ffap-other-tab`。`find-file-other-tab`に相当します。, analogous to .

`C-x t C-r` *filename* `return`
: `ffap-read-only-other-tab`。`find-file-read-only-other-tab`に相当します。

`M-x ffap-next`
: バッファーから次のファイル名または URL を検索して、そのファイルまたは URL を開きます。

`S-mouse-3`
: `ffap-at-mouse`は、マウスがクリックされた周辺のテキストから推測されたファイルを開きます。

`C-S-mouse-3`
: カレントバッファーに記述されたファイルと URL のメニューを表示して、選択されたものを開きます (`ffap-menu`)。

31.13 ゲーム、その他の娯楽

`animate`パッケージは、テキストをダンスさせます (たとえば`M-x animate-birthday-present`)。

`M-x blackbox`、`M-x mpuz`、`M-x 5x5`はパズルです。`blackbox`はボックス内のボールの位置を、トモグラフィー (断層撮影) により当てるパズルです。`mpuz`は掛け算パズルを表示します。掛け算の中の英字が何の数字かを当てなければなりません。数字を入力するには、英字をタイプしてから、その数字をタイプします。`5x5`の目標は、すべてのマスを埋めることです。

`M-x bubbles`は、より少ない回数の移動で、多くの bubble(シャボン玉) を取り除くゲームです。

`M-x decipher`は、単純なアルファベット置換で暗号化されたバッファーを解読するのに役立ちます。

`M-x dissociated-press`は、Emacs のカレントバッファーのテキストをスクランブルします。スクランブルは単語単位または文字単位で行なわれ、`*Dissociation*`という名前のバッファーに出力されます。正の引数は文字単位での操作を指定し、数にはオーバーラップする文字数を指定します。負の引数は単語単位での操作を指定し、数にはオーバーラップする単語数を指定します。Dissociated Press はマルコフ連鎖と酷似した結果を生成しますが、それとは独自の ig オリジナルな創案です。手法としては、単語または文字の後ろにランダムにジャンプするマルコフ連鎖とは異なり、ランダムなジャンプの間にあるサンプルから、連続する複数の文字をコピーします。ユーザーに受け入れられ、正確でありたいなら、ドキュメントには dissociwords を使用しないでください。

`M-x dunnet`は、テキストベースのアドベンチャーゲームを開始します。

個人的な満足感を得たいなら、`M-x gomoku`に挑戦してみてください。これはあなたと五目並べゲームを対戦します。

少し退屈していたら `M-x hanoi`に挑戦してみてください。かなり退屈しているなら、数引数を指定します。とてもとても退屈なら、引数 9 に挑戦してみましょう。さあ、座って眺めましょう。

`M-x life`は、Conway の Life cellular automaton を実行します。

`M-x morse-region`は、リージョンのテキストをモールス信号に変換し、`M-x unmorse-region`で元に戻します。`M-x nato-region`は、リージョンのテキストを NATO 発音記号 (NATO phonetic alphabet) に変換し、`M-x denato-region`で元に戻します。

`M-x pong`、`M-x snake`、`M-x tetris`は、有名な Pong、Snake、Tetris の実装です。

`M-x solitaire`はソリティアーゲームをプレーします。これはピンを他のピンを超えてジャンプさせるゲームです。

`M-x zone`は、Emacs がアイドル時にプレーするゲームです。

円盤状記憶媒体のビットのフリップに butterflies(蝶々) を使用する “真のプログラマー” は、`M-x butterfly`をデプロイします。`https://xkcd.com/378`を参照してください。

最後に、もし不満を感じているときは、有名な精神分析医の Eliza に問題の説明を試みてください。これは `M-x doctor`とタイプするだけです。各入力の最後には、`RET`を 2 回タイプしてください。

32 Emacs Lisp パッケージ

Emacs は Emacs Lisp ライブラリーであるパッケージ *(packages)* で追加機能を実装することにより拡張可能です。これらは自分で記述することも、他の人が提供することもあります。そのようなパッケージを将来の Emacs セッションで利用できるようにインストールしたい場合には、それらをコンパイルして Emacs が Lisp ライブラリーを探すディレクトリーに配置する必要があります。手動によるインストール手法の詳細は Section 24.8 [Lisp Libraries], page 327 を参照してください。多くのパッケージには Lisp ファイルの先頭付近にインストールと使用方法の手順が大きくコメントされています。パッケージのインストールと使用方法の微調整にこれらの手順を使用できます。

パッケージは巨大な Emacs Lisp パッケージのコレクションであるパッケージアーカイブ *(package archives)* から提供されることもあります。それぞれのパッケージは個別の Emacs Lisp プログラムと、Info マニュアルのような別のコンポーネントを含むこともあります。Emacs にはそのようなアーカイブから簡単にパッケージをダウンロードしてインストールする機能が含まれています。このチャプターの残りの部分ではその機能について説明します。

パッケージアーカイブからのインストールで利用可能なパッケージをリストするには `M-x list-packages RET` とタイプします。`M-x list-packages`により、すべてのパッケージのリストを含む、`*Packages*`という名前のバッファーが表示されます。このバッファーを通じてパッケージをインストールしたりアンインストールできます。Section 32.1 [Package Menu], page 490 を参照してください。

コマンド `C-h P` (`describe-package`) はパッケージ名の入力を求め、そのパッケージの属性や、実装する機能を説明するヘルプバッファーを表示します。

デフォルトでは、Emacs は emacs 開発者により保守され GNU プロジェクトによりホスティングされるパッケージアーカイブからパッケージをダウンロードします。オプションでサードパーティーにより保守されるアーカイブから、パッケージをダウンロードすることもできます。Section 32.3 [Package Installation], page 493 を参照してください。

Emacs Lisp のプログラムをインストール可能なパッケージに変更する情報に関しては、Section “Packaging” in *The Emacs Lisp Reference Manual* を参照してください。

32.1 Package Menu バッファー

コマンド `M-x list-packages`は、パッケージメニュー *(package menu)* を立ち上げます。これは Emacs が把握するすべてのパッケージをリストするバッファーです。リストの各行には以下の情報が表示されます:

- パッケージ名 (例: ‘`auctex`’)。
- パッケージのバージョン番号 (例: ‘`11.86`’)。
- パッケージのステータスは通常は ‘`available`’ (パッケージアーカイブからダウンロード可能)、‘`installed`’、‘`built-in`’ (デフォルトで Emacs に同梱) のいずれかです。Section 32.2 [Package Statuses], page 493.<を参照してください。
- 複数のパッケージアーカイブが有効ならそのパッケージがどのパッケージアーカイブのものか。
- パッケージの短い説明。

`list-packages`は、パッケージアーカイブサーバーから利用可能なパッケージのリストを取得するために、ネットワークにアクセスします。ネットワークが利用できない場合、一番最近取得したリストにフォールバックします。

パッケージリストバッファーで主に使うことになるコマンドが `x`コマンドです。ポイント位置にあるパッケージがまだインストールされていなければ、このコマンドがそのパッケージをインストール

します。そしてもしインストール済みのパッケージなら、このコマンドはそのパッケージを削除するのです。

パッケージメニューでは、以下のコマンドが利用可能です:

`h` パッケージメニューの使い方を要約した短いメッセージを表示します (`package-menu-quick-help`)。

`?`

`RET` カレント行のパッケージにたいして、`C-h P`コマンド (Chapter 32 [Packages], page 490 を参照してください) により表示されるヘルプウィンドウと同様の、ヘルプバッファーを表示します (`package-menu-describe-package`)。

`i` カレント行のパッケージをインストールのためにマークします (`package-menu-mark-install`)。パッケージのステータスが '`available`'の場合、行の先頭に文字 '`I`'を追加します。`x`とタイプすると、パッケージをダウンロードしてインストールします (以下参照)。

`d` カレント行のパッケージを削除のためにマークします (`package-menu-mark-delete`)。パッケージのステータスが '`installed`'の場合、行の先頭に文字 '`D`'を追加します。`x`とタイプすると、パッケージを削除します (以下参照)。パッケージ削除の結果、何が起こるかについての情報は、Section 32.4 [Package Files], page 496 を参照してください。

`w` カレント行にあるパッケージのウェブサイトをブラウザでオープンします。ブラウザのオープンには `browse-url`を使用します。

`~` 削除のためにすべての時代遅れのパッケージ obsolete packages をマークします (`package-menu-mark-obsolete-for-deletion`)。これは状態が '`obsolete`'の、すべてのパッケージを削除のためにマークします。

`u`

`DEL` `i`や `d`コマンドによりカレント行に追加された、以前のインストールや削除マークを外します (`package-menu-mark-unmark`)。

`U` 新しいバージョンが利用可能なパッケージすべてをアップグレード用にマークします (`package-menu-mark-upgrades`)。このコマンドによって利用可能な新たなバージョンにはインストール用のマーク、インストール済みの古いバージョンには ('`obsolete`'という状態のマークとともに) 削除用のマークがつけられます。デフォルトではビルトインパッケージには利用可能な新しいバージョン用にマークしませんが、`package-install-upgrade-built-in`をカスタマイズして変更できます。Section 32.3 [Package Installation], page 493 を参照してください。`package-install-upgrade-built-in`を非 `nil`値にカスタマイズする場合には、上書きされたくないビルトインパッケージの更新を避けるために、`U`コマンドがマークしたビルトインパッケージすべてを確認してください。

`x` `i`でマークされたすべてのパッケージをダウンロードしてインストールするとともに、`d`でマークされたすべてのパッケージを削除します (`package-menu-execute`)。これによりマークは削除されます。マークされているパッケージがなければ、このコマンドはポイントの下にあるパッケージのインストール (インストール済みでない場合)、またはパッケージの削除 (インストール済みの場合) を行います。

`g`

`r` パッケージリストを更新します (`revert-buffer`)。これは再度パッケージアーカイブから利用可能なパッケージのリストを取得してパッケージリストを再表示します。

`H` 名前が regexp にマッチするパッケージを非表示にします (`package-menu-hide-package`)。これは regexp の入力を求めて名前がマッチするパッケージを非表示にします。regexp のデフォルト値はポイント位置の名前のパッケージだけを非表示にするので、プロンプトで単に `RET`を押下するとカレントパッケージだけを非表示にします。

`(` 古いバージョンのパッケージ、および優先度低のアーカイブから取得したバージョンの可視性を切り替えます (`package-menu-toggle-hiding`)。

`/ a` パッケージリストをアーカイブでフィルターします (`package-menu-filter-by-archive`)。これはアーカイブ (例: ‘`gnu`’) の入力を求めて、そのアーカイブ由来のパッケージだけを表示します。アーカイブ名をカンマで区切ってタイプすれば、複数のアーカイブを指定できます。

`/ d` description(説明) によりパッケージリストをフィルターします (`package-menu-filter-by-description`)。これは正規表現の入力を求めて、その regexp にマッチする description のパッケージだけを表示します。

`/ k` パッケージリストをキーワードでフィルターします (`package-menu-filter-by-keyword`)。これはキーワード (例: ‘`games`’) の入力を求めて、そのキーワードのパッケージだけを表示します。キーワードをカンマで区切ってタイプすれば、複数のキーワードを指定できます。

`/ N` 名前や description によりパッケージリストをフィルターします (`package-menu-filter-by-name-or-description`)。これは正規表現の入力を求めて、その regexp に名前か descriptor がマッチするパッケージだけを表示します。

`/ n` 名前によりパッケージリストをフィルターします (`package-menu-filter-by-name`)。これは正規表現の入力を求めて、その regexp にマッチする名前のパッケージだけを表示します。

`/ s` パッケージリストを status(状態) でフィルターします (`package-menu-filter-by-status`)。これは 1 つ以上の status(‘`available`’等, Section 32.2 [Package Statuses], page 493 を参照) の入力を求めて、その status にマッチするパッケージだけを表示します。status をカンマで区切ってタイプすれば、複数の status を指定できます。

`/ v` パッケージリストをバージョンでフィルターします (`package-menu-filter-by-version`)。これはまずバージョン文字列にたいする比較シンボルとして ‘`<`’、‘`>`’、‘`=`’ いずれかの入力を求めて、それに応じてタイプしたバージョンより小さい、大きい、あるいは等しいバージョンのパッケージを表示します。

`/ m` 非空のマークでパッケージリストをフィルターします (`package-menu-filter-marked`)。これはインストールまたは削除とマークされたパッケージだけを表示します。

`/ u` 利用可能なアップグレードが存在するパッケージだけを表示するようパッケージリストをフィルターします (`package-menu-filter-upgradable`)。このフィルターはデフォルトでは、ビルトインパッケージでは新しいバージョンが利用可能でも除外しますが、これは`package-install-upgrade-built-in`をカスタマイズして変更できます。Section 32.3 [Package Installation], page 493 を参照してください。

`/ /` カレントで適用されたパッケージリストのフィルターをクリアします (`package-menu-filter-clear`)。

たとえばパッケージをインストールするには、そのパッケージの行で `i`をタイプしてから、`x`をタイプします。

32.2 パッケージのステータス

パッケージは以下のステータスのいずれかを保有できます:

‘available’
: パッケージは未インストールだがパッケージアーカイブからのダウンロードとインストールが可能。

‘avail-obso’
: インストールするためにパッケージは利用可能だが新たなバージョンも利用可能。このステータスのパッケージはデフォルトでは非表示。

‘built-in’
: パッケージはデフォルトで Emacs に同梱されている。これはパッケージメニューからは削除できず、デフォルトではアップグレード対象とはみなされない (だが `package-install-upgrade-built-in`をカスタマイズして変更可, Section 32.3 [Package Installation], page 493 を参照のこと)。

‘dependency’
: 別パッケージの依存関係を満足するために自動的にインストールされたパッケージ。

‘disabled’
: パッケージは `package-load-list`変数を使用して無効化された。

‘external’
: このパッケージは built-in ではなく、`package-user-dir`で指定されたディレクトリーのものではない (Section 32.4 [Package Files], page 496 を参照)。外部パッケージは‘`built-in`’と同様に扱われて削除できない。

‘held’
: パッケージは held(固定) されている。Section 32.3 [Package Installation], page 493 を参照のこと。

‘incompat’
: 何らかの理由 (たとえばインストール不可能なパッケージに依存) によりパッケージはインストール不能。

‘installed’
: パッケージはインストール済み。

‘new’
: ‘`available`’と等価だが最後に `M-x list-packages`を呼び出した後にパッケージがパッケージアーカイブで新たに利用可能になった点が異なる。

‘obsolete’
: パッケージはインストール済みの古いバージョン。パッケージの当該バージョンに加えて、新たなバージョンもインストール済み。

32.3 パッケージのインストール

パッケージを一番便利にインストールするのはパッケージメニューを使う方法 (Section 32.1 [Package Menu], page 490 を参照) ですが、コマンド `M-x package-install`を使用することもできます。これはステータスが‘`available`’のパッケージ名の入力を求めて、それをダウンロードしてからインストールします。同様にあるパッケージのアップグレード行いたければ `M-x package-upgrade`コマンド、すべてのパッケージをアップグレードしたければ `M-x package-upgrade-all`コマンドを使うことができます。

デフォルトではアーカイブから新しいバージョンが利用可能なビルトインパッケージを`package-install`は考慮しません (パッケージがEmacsディストーションに含まれているパッケージはビルトイン)。特にビルトインパッケージはパッケージ入力を求める際の補完候補には表示されません。ただしプレフィックス引数とともに`package-install`を呼び出すと、アップグレード可能なビルトインパッケージも考慮するようになります。変数`package-install-upgrade-built-in`をカスタマイズすることによって、この挙動をデフォルトにすることができます。この変数の値が非`nil`であれば、たとえプレフィックス引数なしで`package-install`を呼び出した際にもビルトインパッケージが考慮されるようになります。`package-install-upgrade-built-in`はpackage-menuコマンド (Section 32.1 [Package Menu], page 490を参照) にも影響を与えることに注意してください。

これとは対照的に、`package-upgrade`と`package-upgrade-all`がビルトインパッケージを考慮することは決してありません。あるビルトインパッケージのアップグレードにこれらのコマンドを使いたければ`C-u M-x package-install RET`を通じてアップグレードするか、あるいは`package-install-upgrade-built-in`を非`nil`値にカスタマイズしてからパッケージメニューか`package-install`からそれらのパッケージを一度アップグレードする必要があります。

`package-upgrade-all`やパッケージメニューの`U`といったコマンドを使うと一度に多くのパッケージがアップデートされるので、`package-install-upgrade-built-in`を非`nil`値にカスタマイズする場合には注意してください。あなたの意図に反して、これらのコマンドがビルトインパッケージをアーカイブ由来の新バージョンに上書きして置き換えてしまうかもしれません。少数のビルトインパッケージだけをアップデートしたい場合には、これらのバルクコマンド (bulk command: 一度に大量の処理を行うコマンド) を使わないでください。

他のパッケージが提供する機能に依存するために、それらのパッケージがインストール済みであることを必要 *(require)* とするパッケージもあるでしょう。Emacsがそのようなパッケージをインストールするときは、必要なパッケージがインストールされていなければ、それらのパッケージのダウンロードとインストールも自動に行ないます (必要なパッケージが何らかの理由で利用できない場合、Emacsはエラーをシグナルしてインストールを中止します)。パッケージの必要条件リスト (requirements list) は、そのパッケージのヘルプバッファーに表示されます。

デフォルトでは、パッケージはEmacs開発者により保守される単一のパッケージアーカイブからダウンロードされます。これは変数`package-archives`により制御されます。この変数の値は、Emacsが認識するパッケージアーカイブのリストです。リストの各要素は`(`*`id . location`*`)`という形式でなければなりません。ここで、*id*はパッケージアーカイブの名前、*location*はパッケージアーカイブディレクトリーのURLか名前です。サードパーティーのアーカイブを使用したい場合はこのリストを変更できます— が、自己責任で行い、信用できるサードパーティーだけを使用してください!

パッケージアーカイブのメンテナーは、パッケージにサイン *(signing)* を付して、信頼度を増すことができます。これらはプライベートとパブリックのペアーからなる暗号化キーにより生成されます。プライベートキーは各パッケージにたいする署名ファイル *(signature file)* を作成するのに使用されます。パブリックキーにより、署名ファイルを使用してそのパッケージ作成者と、それが改ざんされていないかを確認できます。署名の検証は、EasyPG インターフェイス (Section “EasyPG” in *Emacs EasyPG Assistant Manual* を参照) を通じて the GnuPG package (`https://www.gnupg.org/`) を使用します (Section “EasyPG” in *Emacs EasyPG Assistant Manual* を参照)。有効な署名であっても、それが悪意がないパッケージであることを厳正に保証する訳ではなく、用心するべきです。パッケージアーカイブは、パブリックキーの入手方法について、説明を提供するべきです。`http://pgp.mit.edu/`のようなサーバーからキーをダウンロードするのも1つの方法です。Emacsにキーをインポートするには、`M-x package-import-keyring`を使用します。Emacsは変数`package-user-dir`で指定されるディレクトリー (デフォルトは`package-gnupghome-dir`のサブディレクトリー`gnupg`) にパッケージキーを格納します。これにより、Emacsが署名を検証する際

に、オプション gnupgで GnuPG を呼び出すようになります。package-gnupghome-dirが nil, の場合は、GnuPG のオプション '--homedir'は省略します。GNU パッケージアーカイブにたいするパブリックキーは Emacs と共に配布され、etc/package-keyring.gpgにあります。Emacs はこれを自動的に使用します。

ユーザーオプション package-check-signatureが非 nilの場合、Emacs はパッケージのインストール時に署名の検証を試みます。このオプションが値 allow-unsignedをもち、使用可能な OpenPGP 設定が見つかったらサインされたパッケージはチェックされますが、未サインのパッケージもまだインストールができます。パッケージにサインしないアーカイブを使用する場合には、それらを package-unsigned-archivesに追加できます (値が allow-unsignedで使用可能な OpenPGP が見つからなければこのオプションはあたかも値が nilであるかのように扱われる)。値が tなら少なくとも 1 つの署名が有効でなければならず、allならすべての署名が有効でなければなりません。

暗号化キーとサインについての詳細は、Section "GnuPG" in *The GNU Privacy Guard Manual* を参照してください。Emacs の GNU Privacy Guard にたいするインターフェースについては、Section "EasyPG" in *Emacs EasyPG Assistant Manual* を参照してください。

複数のパッケージアーカイブが有効で、同じパッケージにたいして異なるバージョンを提供する場合は、オプション package-pinned-packagesが便利かもしれません。指定したパッケージが指定されたアーカイブだけからダウンロードされるように、このリストにパッケージとアーカイブのペアーを追加できます。

複数の有効なパッケージアーカイブがあるときに便利な他のオプションとして、package-archive-prioritiesがあります。これは各アーカイブにたいして、優先度 (高い数字は高い優先度のアーカイブを指定します) を指定します。このオプションにより指定されない限り、アーカイブの優先度はデフォルトの 0 です。優先度高のアーカイブのパッケージが利用可能な場合、優先度低のアーカイブのパッケージはメニューに表示されません (これは package-menu-hide-low-priorityの値により制御されます)。

一度パッケージをダウンロードしてバイトコンパイル、インストールするとそのパッケージはカレント Emacs セッションで利用可能になります。パッケージを利用可能にするにはパッケージのディレクトリーを load-pathに追加してパッケージの autoload をロードします。パッケージの autoload の効果はパッケージごとにさまざまです。ほとんどのパッケージはいくつかの新たなコマンドを利用可能にするだけですが、Emacs セッションにたいして広範な影響を及ぼすものもあります。この種の情報についてはパッケージのヘルプバッファーを参照してください。

インストールされたパッケージはその後のすべての Emacs セッションで自動的にロードされます。これは Emacs 開始時の init ファイル処理前、早期 init ファイル (Section 33.4.6 [Early Init File], page 534 を参照) の処理後に行なわれます。例外として '-q'か '--no-init-file'オプション (Section C.2 [Initial Options], page 576 を参照してください) で呼び出されたときは、Emacs 開始時にパッケージを利用可能にしません。

スタートアップ時に Emacs が自動的にパッケージを利用可能しないようにするためには変数 package-enable-at-startupを nilに変更してください。この変数は正規の init ファイル (regular init file) のロード前に読み込まれるので早期 init ファイル (early init file) で行うようにしてください。現在のところこの変数は Customize を通じてセットできません。

多くのパッケージのインストール後には、ユーザーオプション package-quickstartのセッティングでスタートアップタイムを改善できるかもしれません。このオプションをセットすることによって、Emacs のスタートアップ時に毎回再計算するかわりに、多くの事項を Emacs に事前に計算させることができます。しかしこれを行うと、package-load-listの値の変更時のようなアクティベーション変更が必要になった際にコマンド package-quickstart-refreshを手動で実行する必要が生じます。

`package-enable-at-startup`を`nil`にセットしてもスタートアップ中およびスタートアップじにパッケージを利用可能にすることはできます。インストール済みパッケージをスタートアップ中に利用可能にするには init ファイル内で関数`package-activate-all`を呼び出し手ください。スタートアップ後に利用可能にするにはコマンド`M-: (package-activate-all) RET`を呼び出してください。

スタートアップ時に利用可能になるようにパッケージのロードをより精密に制御するために、変数 `package-load-list`を使用することができます。この変数の値にはリストを指定します。`(`*`name version`*`)` という形式のリスト要素は、*name*という名前のパッケージのバージョン *version* を利用可能にするように指示します。ここで *version*には、(そのパッケージの特定のバージョンに対応する) バージョン文字列、`t` (任意のインストール済みのバージョンを意味する)、または`nil` (バージョンを意味しない。パッケージを利用可能にすることを抑止してパッケージを無効にする) を指定します。リストの要素にはシンボル`all`も指定でき、これは他のリスト要素で名前指定されていない任意のパッケージのインストール済みバージョンを利用可能にすることを意味します。デフォルト値は単に`'(all)`となっています。

たとえば `package-load-list`を`'((muse "3.20") all)`にセットすると Emacs は ‘`muse`’のバージョン 3.20 のみ、および ‘`muse`’以外のパッケージのインストール済みの任意のバージョンをロードします。‘`muse`’の他のバージョンがインストールされていたとしてもそれらは無視されます。‘`muse`’パッケージは ‘`held`’というステータスでパッケージメニューにリストされるはずです。

Emacs のバイトコードは極めて安定していますがバイトコードが古くなってしまったり、コンパイル済みファイルが依存するマクロが新しいバージョンの Emacs で変更されてしまう可能性もあります。特定のパッケージをリコンパイルするためには`M-x package-recompile`、すべてのパッケージをリコンパイルするためには`M-x package-recompile-all` というコマンドを使うことができます (インストール済みパッケージの数が多ければ後者コマンドの実行にかなりの時間を要するかもしれない)。

32.4 パッケージのファイルとディレクトリー

各パッケージはパッケージアーカイブから単一ファイル形式 ——— 1 つの Emacs Lisp ソースファイル、または複数の Emacs Lisp ソースと他のファイルを含む tar ファイル — でダウンロードされます。パッケージファイルは、パッケージをインストールする Emacs コマンドにより自動的に取得、処理、配置されます。パッケージを作成する (Section “Packaging” in *The Emacs Lisp Reference Manual* を参照してください) のでない限り、通常これらを直接扱う必要はないでしょう。パッケージファイルから直接パッケージをインストールする必要があるときは、コマンド`M-x package-install-file` を使用してください。

1 度インストールされると、パッケージの内容はそのサブディレクトリーに配置されます (変数`package-user-dir`を変更することにより、ディレクトリーの名前を変更できます)。パッケージのサブディレクトリーは*`name-version`*という名前で、*name*はパッケージ名、*version*はバージョン文字列です。

`package-user-dir`に加えて、Emacs は`package-directory-list`にリストされたディレクトリーからインストール済みパッケージを探します。これらのディレクトリーはシステム管理者のためのディレクトリーで、Emacs パッケージをシステムワイドに利用可能にするためのものです。Emacs 自身がこれらのディレクトリーにパッケージをインストールすることはありません。`package-directory-list`にたいするパッケージのサブディレクトリーは、`package-user-dir`と同じ方法で配置されます。

パッケージの削除 (Section 32.1 [Package Menu], page 490 を参照してください) は、対応するパッケージのサブディレクトリーを削除します。これは`package-user-dir`にインストールされた

パッケージだけに機能します。システムワイドなパッケージディレクトリーにたいして呼び出された場合、削除コマンドはエラーをシグナルします。

32.5 パッケージソースのフェッチ

`package-install`はデフォルトではパッケージアーカイブから tarball をダウンロードしてそのファイルをインストールします。あなたがパッケージのソースをハッキングして、あなたの変更を他の人たちと共有したい場合には、これでは不十分かもしれません。そのような場合にはアップストリームのソースを直接取得して、それにたいして作業したいと思うのではないでしょうか。そのようにすることによって、パッチ開発やバグレポートが容易になることが多々あるからです。

パッケージのソースコードをソースから直接手に入れるために`package-vc-install`を使うのも 1 つの手段です。このコマンドは通常のパッケージの場合と同じように、すべてのファイルがバイトコンパイルされて、auto-load されることも保証してくれます。この方法でインストールされたパッケージは、他のパッケージとまったく同じように振る舞うでしょう。それらのパッケージは`package-upgrade`や`package-upgrade-all`を使ってアップグレード、`package-delete`を使って再び削除することができます。通常のパッケージのリストにさえこれらのパッケージが表示されます。パッケージのリストに加えずにソースの clone だけを望む場合には、`package-vc-checkout`を使用してください。

ソースをチェックアウトするとカレントの開発 HEAD にたいするバグを再現させたり、新しい機能を実装して不満を解消したいと思うかもしれません。このパッケージのメタデータにメンテナーへ連絡する方法が示されていれば、コマンド`package-report-bug`を使用してバグを電子メール経由で報告することができます。このレポートにはあなたがカスタマイズしたすべてのユーザーオプションが含まれます。メンテナーと共有したい変更を行ったら、まずその変更をコミットしてからコマンド`package-vc-prepare-patch`でその変更を共有できます。⟨undefined⟩ [Preparing Patches], page ⟨undefined⟩ を参照してください。

あなたが自分のパッケージを保守している場合は、リモートレポジトリに one するのではなくローカルでのチェックアウトを使いたいと思うかもしれません。`package-vc-install-from-checkout`を使ってこれを行うことができます。これはパッケージのディレクトリー (Section 32.4 [Package Files], page 496 を参照) からあなたのチェックアウトにシンボリックリンクを作成して、コードの初期化を行うコマンドです。初期化を繰り返して autoload を更新するために、`package-vc-rebuild`の使用が必要かもしれないことに注意してください。

32.5.1 パッケージソースの指定

ソースからパッケージをインストールするためにはパッケージのソースをどこから取得するか (たとえばコードレポジトリ)、そしてコードの構造に関する基本情報 (たとえば複数ファイルパッケージのメインファイル) を Emacs が承知していなければなりませんこれらのプロパティを記述するのがパッケージ仕様 *(package specification)* です。

パッケージアーカイブ (Section “Package Archives” in *The Emacs Lisp Reference Manual* を参照) からサポートされると、Emacs はそのアーカイブからパッケージの仕様を自動的にダウンロードできます。`package-vc-install`に渡される 1 つ目の引数がパッケージを命名するシンボルであれば、Emacs はそのパッケージ用にアーカイブが提供する仕様を使用します。

```
;; Emacs は GNU ELPA から BBDB の仕様をダウンロードする:
(package-vc-install 'bbdb)
```

`package-vc-install`の 1 つ目の引数はパッケージ仕様の場合もあります。これによってユーザーオプション`package-archives`にリストされている既知のアーカイブ以外の場所からソースパッケージをインストールできるようになります。パッケージ仕様は`(`*`name . spec`*`)`という形式のリストです。ここで *spec*は下記テーブルの任意のキーを使用するプロパティリストである必要があります。

コードレポジトリやバージョンコントロールシステムでの作業に使用する基本的な用語の定義については Section “VCS Concepts” in *The GNU Emacs Manual* を参照してください。

`:url` パッケージのソースコードを取得するレポジトリを指定する URL 文字列。

`:branch` インストールするコードのリビジョンを指定する文字列。バージョンのバージョン番号と混同しないこと。

`:lisp-dir`
Lisp ソースのロードに用いる、レポジトリに相対的なディレクトリー名文字列。デフォルトはレポジトリのルートディレクトリー。

`:main-file`
パッケージのメタデータを集約したプロジェクトのメインファイル名文字列。与えられない場合のデフォルトはパッケージ名に".el"を追加したファイル名。

`:doc` Info ファイルをビルドするためのレポジトリに相対的なドキュメンテーションファイル名文字列。Texinfo または Org のファイルが指定できる。

`:vc-backend`
パッケージのレポジトリのコピーのダウンロードに用いる VC バックエンドを指名する文字列 (Section “Version Control Systems” in *The GNU Emacs Manual* を参照)。省略時には提供された URL から Emacs が推測を試み、それに失敗するとプロセスは `package-vc-default-backend`の値にフォールバックする。

```
;; 情報を手作業で指定:
(package-vc-install
  '(bbdb :url "https://git.savannah.nongnu.org/git/bbdb.git"
         :lisp-dir "lisp"
         :doc "doc/bbdb.texi"))
```

33 カスタマイズ

このチャプターでは、Emacs の振る舞いをカスタマイズするシンプルな方法をいくつか説明します。

ここで説明する方法とは別に、Emacs をカスタマイズするために X resources を使用する情報については Appendix D [X Resources], page 591、キーボードマクロの記録と再生については Chapter 14 [Keyboard Macros], page 137 を参照してください。より広範で制限のない変更を行なうには、Emacs Lisp コードを記述する必要があります。*The Emacs Lisp Reference Manual* を参照してください。

33.1 Easy Customization インターフェース

Emacs には変更できる多くのセッティング *(settings)* があります。ほとんどのセッティングはカスタマイズ可能な変数 *(customizable variables。Section 33.2 [Variables], page 508* を参照してください*)* で、これらはユーザーオプション *(user options)* とも呼ばれます。非常にたくさんのカスタマイズ可能な変数があり、それらは Emacs の振る舞いを数々の側面から制御します。このマニュアルにドキュメントされている変数は、[Variable Index], page 679 にリストされています。セッティングの別のクラスにはフェイス *(faces)* があり、これはフォント、カラー、その他のテキスト属性を決定します (Section 11.8 [Faces], page 82 を参照してください)。

セッティング (変数およびフェイスの両方) を閲覧したり変更するには、`M-x customize`とタイプします。これは論理的に組織化されたセッティングのリストの操作、値の編集とセット、永続的な保存を行なうことができる、カスタマイズバッファー *(customization buffer)* を作成します。

33.1.1 カスタマイズグループ

カスタマイズセッティングは、カスタマイズグループ *(customization groups)* に組織化されています。これらのグループはより大きなグループに集められ、最終的に `Emacs`と呼ばれるマスターグループに集約されます。

`M-x customize`は、トップレベルの `Emacs`グループを表示するカスタマイズバッファーを作成します。これは、部分的には以下のようなものです:

```
For help using this buffer, see [Easy Customization] in the [Emacs manual].

_____________________________________ [ Search ]

 Operate on all settings in this buffer:
 [ Revert... ] [ Apply ] [ Apply and Save ]


Emacs group: Customization of the One True Editor.
      [State]: visible group members are all at standard values.
      See also [Manual].

[Editing]      Basic text editing facilities.
[Convenience]  Convenience features for faster editing.

...more second-level groups...
```

このバッファーも表示されている主要な部分は '`Emacs`'カスタマイズグループで、これはいくつかの他のグループ ('`Editing`'、'`Convenience`'など) を含みます。これらのグループの内容はここではリストされず、それぞれにたいして 1 行のドキュメントだけが表示されています。

グループの *state(*ステート、状態*)* には、そのグループ内のセッティングが、編集されているか (edited)、セットされているか (set)、保存されているか (saved) が示されます。Section 33.1.3 [Changing a Variable], page 500 を参照してください。

カスタマイズバッファーのほとんどは読み取り専用ですが、編集できるいくつかの編集可能フィールド *(editable fields)* が含まれています。たとえばカスタマイズバッファーの最上部にある編集可能フィールドは、セッティングを検索するためのものです (Section 33.1.2 [Browsing Custom], page 500 を参照してください)。マウスでクリック、またはポイントをそこに移動して RETをタイプすることによりアクティブにできる、ボタン *(buttons)* やリンク *(links)* もあります。たとえば‘[Editing]’のようなグループ名はリンクで、これらのリンクをアクティブにすることにより、そのグループにたいするカスタマイズバッファーが立ち上がります。

カスタマイズバッファーでは、TAB (widget-forward) とタイプすると、次のボタンまたは編集可能フィールドに前方へ移動します。S-TAB (widget-backward) は、前のボタンまたは編集可能フィールドに後方へ移動します。

33.1.2 セッティングのブラウズと検索

M-x customizeにより作成されたトップレベルのカスタマイズバッファーから、カスタマイズグループ‘Emacs’のサブグループへのリンクをフォローできます。これらのサブグループは、カスタマイズするためのセッティングを含んでいるでしょう。また、これらのサブグループには、Emacs のより特化したサブシステムを扱うサブグループが、さらに含まれているかもしれません。カスタマイズグループの階層を移動していけば、カスタマイズしたい、いくつかのセッティングが見つかるでしょう。

特定のセッティングまたはカスタマイズグループのカスタマイズに興味がある場合は、コマンド M-x customize-option、M-x customize-face、M-x customize-group で直接移動することもできます。Section 33.1.6 [Specific Customization], page 505 を参照してください。

どのグループまたはセッティングをカスタマイズしたいか確信がもてない場合、各カスタマイズバッファーの上部にある、編集可能なサーチフィールドを使用して、それらを検索できます。このフィールドで検索条件— 1 つの単語またはスペースで区切られた複数の単語、または正規表現 (Section 12.6 [Regexps], page 115 を参照してください) — をタイプできます。それからそのフィールドで RET をタイプするか、となりの‘Search’ボタンをアクティブにすることにより、その条件にマッチするグループとセッティングを含むカスタマイズバッファーに切り替わります。しかし、この機能はカレント Emacs セッションにロードされたグループ、またはセッティングだけを探すことに注意してください。

カスタマイズバッファーにサーチフィールドを表示したくない場合は、変数 custom-search-fieldを nilに変更してください。

コマンド M-x customize-aproposは、同じようにサーチフィールドを使用しますが、これはミニバッファーを使用して検索条件を読み取ります。Section 33.1.6 [Specific Customization], page 505 を参照してください。

M-x customize-browseは、利用可能なセッティングをブラウズする別の方法です。このコマンドは、グループまたはセッティングの名前だけを、構造化されたレイアウトで表示する、特別なカスタマイズバッファーを作成します。グループ名のとなりの‘[+]’ボタンを呼び出すことにより、同じバッファーでグループの内容を表示できます。グループの内容が表示されている場合、ボタンは‘[-]’に変化し、それを呼び出すことにより、再びグループ内容を隠すことができます。このバッファーのグループまたはセッティングには、それぞれ‘[Group]’、‘[Option]’、‘[Face]’というリンクがあります。このリンクを呼び出すことにより、そのグループ、オプション、フェイスだけを表示する、通常のカスタマイズバッファーが作成されます。M-x customize-browseでは、この方法によりセッティングを変更します。

33.1.3 変数の変更

以下は変数またはユーザーオプションが、カスタマイズバッファーではどのように表示されるかの例です:

```
[Hide] Kill Ring Max: 60
```

```
[State]: STANDARD.
Maximum length of kill ring before oldest elements are thrown away.
```

最初の行には、この変数の名前が`kill-ring-max`であることが、見やすいよう‘`Kill Ring Max`’のようにフォーマットされてに表示されています。この変数の値は‘`120`’です。‘`[Hide]`’というラベルのボタンは、アクティブにした場合は、この変数の値とステートを隠します。これは、変数がもし非常に長い値をもつ場合、カスタマイズバッファーが見にくくなるのを避けるために便利です (この理由により、非常に長い値をもつ変数は、最初は隠されています)。‘`[Hide]`’ボタンを使用すると、ボタンは‘`[Show Value]`’に変化し、これをアクティブにすると値とステートが表示されます。グラフィカルなディスプレイでは、‘`[Hide]`’と‘`[Show Value]`’ボタンは、下向きまたは右向きのグラフィカルな三角形で置き換えられます。

変数名の次の行は、変数のカスタマイズ状態 *(customization state)* を示します。この例では‘`STANDARD`’で、これは変数を変更していないので、値はデフォルトのままだということを意味します。‘`[State]`’ボタンは、変数をカスタマイズするためのオペレーションメニューを提供します。

カスタマイズのステートの下は、変数のドキュメントです。これは`C-h v`コマンド (Section 33.2.1 [Examining], page 508 を参照してください) で表示されるのと同じドキュメントです。ドキュメントが複数行の場合、1 行だけが表示されます。この場合、その行の最後に‘`[More]`’ボタンが表示されるので、これをアクティブにすれば完全なドキュメントを表示できます。

‘`Kill Ring Max`’に新しい値を入力するには、値にポイントを移動してそれを編集するだけです。たとえば`M-d`とタイプして‘`60`’を削除して、別の値をタイプします。テキストの変更を開始すると、‘`[State]`’行が変化します:

```
[State]: EDITED, shown value does not take effect until you
         set or save it.
```

値を編集してもすぐに変更は反映されません。変更を反映するには、‘`[State]`’をアクティブにして、‘`Set for Current Session`’を選択することにより、変数をセット *(set)* しなければなりません。すると変数のステートは以下のようになります:

```
[State]: SET for current session only.
```

無効な値を指定してしまうことを心配する必要はありません。‘`Set for Current Session`’オペレーションは正当性をチェックして、不当な値はインストールしません。

ファイル名、ディレクトリー名、Emacs コマンドのようなタイプの値を編集するときは、`C-M-i` (`widget-complete`)、または等価なキー `M-TAB`、`ESC TAB`で補完を行なうことができます。これはミニバッファーでの補完と同じように振る舞います (Section 5.4 [Completion], page 31 を参照してください)。

編集可能な値フィールドで`RET`とタイプすることにより、`TAB`のように、次のフィールドまたはボタンに移動できます。したがってフィールドの編集を終えたら`RET`とタイプして、次のボタンまたはフィールドに移動できます。編集可能なフィールドに改行を挿入するには、`C-o`または`C-q C-j`を使用します。

あらかじめ決められた値しかセットできず、値を直接編集することができない変数もいくつかあります。そのような変数の値の前には、かわりに‘`[Value Menu]`’ボタンが表示されます。このボタンをアクティブにすると、値の選択肢が表示されます。“on か off” のブーリーン値にたいしては、‘`[Toggle]`’ボタンが表示され、このボタンにより値のオンとオフを切り替えることができます。‘`[Value Menu]`’ボタンや‘`[Toggle]`’ボタンを使用した後は、変数をセットして、選択した値を反映するために、再度値をセットしなければなりません。

複雑な構造の値をもつ変数もいくつか存在します。たとえば、`minibuffer-frame-alist`の値は連想配列 (association list、alist) です。これはカスタマイズバッファーでは、以下のように表示されます:

```
[Hide] Minibuffer Frame Alist:
```

```
[INS] [DEL] Parameter: width
            Value: 80
[INS] [DEL] Parameter: height
            Value: 2
[INS]
   [ State ]: STANDARD.
   Alist of parameters for the initial minibuffer frame. [Hide]
   [...more lines of documentation...]
```

この場合、リストの各 association 要素は 2 つのアイテムからなり、1 つは‘Parameter’というラベルがつき、もう 1 つは‘Value’というラベルがつき、両方とも編集可能フィールドです。となりにある‘[DEL]’ボタンでリストから association を削除できます。association を追加するには、挿入したい位置の‘[INS]’ボタンを使用します。一番最後の‘[INS]’ボタンはリストの最後に挿入します。

変数をセットした場合、新しい値はカレント Emacs セッションでだけ効果があります。将来のセッションのために値を保存 *(save)* するには、‘[State]’ボタンを使用して、‘Save for Future Sessions’オペレーションを選択します。Section 33.1.4 [Saving Customizations], page 503 を参照してください。

‘[State]’ボタンを使用して‘Erase Customization’オペレーションを選択することにより、変数の値をその変数の標準値に復元することもできます。実際には 4 つのリセットオペレーションがあります:

‘Undo Edits’
: 値を変更したが、まだ変数をセットしていない場合は、実際の値にマッチするようにバッファーのテキストを復元します。

‘Revert This Session's Customizations’
: これは、変更された変数がある場合は、変数の値を最後に保存された値に復元し、それ以外は標準の値に復元します。テキストも復元された変数に合わせて更新します。

‘Erase Customization’
: これは変数をその変数の標準値にセットします。保存した値も削除します。

‘Set to Backup Value’
: これはこのセッションでカスタマイズバッファーでセットされる前の値に、変数をリセットします。変数をカスタマイズしてからリセットすると、これはカスタマイズした値を破棄するので、このオペレーションにより、破棄した値に戻すことができます。

特定のカスタマイズにたいして、コメントを記録できれば便利なこともあります。コメントを入力するフィールドを作成するには、‘[State]’メニューの‘Add Comment’アイテムを使用します。

カスタマイズバッファーの上部には 2 行のボタン行があります:

```
Operate on all settings in this buffer:
[Revert...] [Apply] [Apply and Save]
```

‘[Revert...]’ボタンは、上述で説明したうち、最初の 3 つのリセット操作をドロップダウンします。‘[Apply]’ボタンは、カレントセッションにセッティングを適用します。‘[Apply and Save]’ボタンはセッティングを適用して、将来のセッションのためにそれらのセッティングを保存します。このボタンは、Emacs が-qまたは-Qのオプションで開始された場合は表示されません (Section C.2 [Initial Options], page 576 を参照)。

コマンド C-c C-c (Custom-set) は、‘[Set for Current Session]’ボタンを使用するのと等価です。コマンド C-x C-s (Custom-save) は、‘[Save for Future Sessions]’ボタンを使用するのと同様です。

‘[Exit]’ボタンはカスタマイズバッファーを、バッファーリストの最後のバッファーに隠し (bury) ます。カスタマイズバッファーを kill させるようにするには、変数 custom-buffer-done-killをtに変更します。

33.1.4 カスタマイズの保存

カスタマイズバッファーでは、カスタマイズしたセッティングの‘[State]’ボタンで‘Save for Future Sessions’を選択することにより、それを保存 *(save)* できます。C-x C-s (Custom-save) コマンド、またはカスタマイズバッファーのトップにある‘[Apply and Save]’ボタンで、そのバッファー内で適用可能なすべてのセッティングが保存されます。

ファイル (通常は初期化ファイル。Section 33.4 [Init File], page 528 を参照してください) にコードを書き込むことにより保存は機能します。将来の Emacs セッションは、開始時に自動的にこのファイルを読み込んで、カスタマイズを再びセットします。

初期化ファイル以外の他のファイルにカスタマイズを保存する選択もできます。これが機能するには、変数 custom-fileに保存したいファイル名をセットして、そのファイルをロードするコード行を追加しなければなりません。たとえば:

```
(setq custom-file "~/.config/emacs-custom.el")
(load custom-file)
```

以下のようにして、Emacs のバージョンごとに違うカスタマイズファイルを指定することさえ可能です:

```
(cond ((< emacs-major-version 28)
       ;; Emacs 27 用のカスタマイズ
       (setq custom-file "~/.config/custom-27.el"))
      ((and (= emacs-major-version 26)
            (< emacs-minor-version 3))
       ;; バージョン 26.3 以前の Emacs 26 用カスタマイズ
       (setq custom-file "~/.config/custom-26.el"))
      (t
       ;; Emacs バージョン 28.1 以降
       (setq custom-file "~/.config/emacs-custom.el")))

(load custom-file)
```

Emacs が-qまたは--no-init-fileオプションで呼び出されたときは、カスタマイズを初期化ファイルに保存しません。なぜならそのようなセッションからカスタマイズを保存することにより、初期化ファイルに記述されていた他のすべてのカスタマイズが消されてしまうからです。

将来のセッションのために保存することを選択しなかった場合、そのカスタマイズは Emacs の終了とともに失われてしまうことに注意してください。終了時に保存されていないカスタマイズにたいするメッセージを表示させたい場合は、初期化ファイルに以下を追加してください:

```
(add-hook 'kill-emacs-query-functions
          'custom-prompt-customize-unsaved-options)
```

33.1.5 フェイスのカスタマイズ

フェイス (Section 11.8 [Faces], page 82 を参照してください) をカスタマイズできます。フェイスは、異なる種類のテキストを Emacs がどのように表示するか決定します。カスタマイズグループは、変数とフェイスの両方を含むことができます。

たとえばプログラミング言語のモードでは、ソースコードのコメントはフェイス font-lock-comment-faceで表示されます (Section 11.13 [Font Lock], page 89 を参照してください)。カスタマイズバッファーでは、‘[Show All Attributes]’リンクをクリックした後は、このフェイスについて以下のように表示されます:

```
[Hide] Font Lock Comment Face:[sample]
   [State] : STANDARD.
   Font Lock mode face used to highlight comments.
   [ ] Font Family: --
   [ ] Font Foundry: --
   [ ] Width: --
   [ ] Height: --
   [ ] Weight: --
   [ ] Slant: --
   [ ] Underline: --
   [ ] Overline: --
   [ ] Strike-through: --
   [ ] Box around text: --
   [ ] Inverse-video: --
   [X] Foreground: Firebrick     [Choose]  (sample)
   [ ] Background: --
   [ ] Stipple: --
   [ ] Inherit: --
   [Hide Unused Attributes]
```

最初の 3 行にはフェイス名、‘[State]’ボタン、そのフェイスにたいするドキュメントが表示されます。その下は、フェイス属性 *(face attributes)* のリストです。それぞれの属性の前にはチェックボックスがあります。チェックされているチェックボックスは‘[X]’と表示され、このフェイスがその属性に値を指定していることを意味します。空のチェックボックスは‘[ ]’と表示され、このフェイスがその属性に特に値を指定していないことを意味します。チェックボックスをアクティブにすることにより、その属性を指定または未指定にできます。

フェイスにすべての属性を指定する必要はありません。実際のところ、ほとんどのフェイスは少しの属性しか指定していません。上記の例では、font-lock-comment-faceはフォアグラウンドカラーだけを指定しています。未指定の属性にたいしては、すべての属性が指定された特別なフェイスdefaultの属性が使用されます。defaultフェイスは、明示的にフェイスが割り当てられていない任意のテキストを表示するために使用されるフェイスです。さらに、このフェイスのバックグラウンドカラー属性には、フレームのバックグラウンドカラーが使用されます。

属性リストの最後にある‘[Hide Unused Attributes]’ボタンは、このフェイスの未指定の属性を隠します。隠された属性があるとき、ボタンは‘[Show All Attributes]’に変化し、これはすべての属性リストを表示します。カスタマイズバッファーは、インターフェースが見にくくなるのを避けるため、未指定の属性が隠された状態で開始されるでしょう。

属性を指定するときは、通常の方法で値を変更できます。

フォアグラウンドカラーとバックグラウンドカラーは、カラーネームと RGB トリプレットの両方を使用して指定できます (Section 11.9 [Colors], page 83 を参照してください)。カラーネームのリストに切り替えるために、‘[Choose]’ボタンも使用できます。そのバッファーで RETでカラーを選択すると、値フィールドにそのカラーネームが入ります。

フェイスのセット・保存。リセットは、変数にたいする操作と同様に機能します (Section 33.1.3 [Changing a Variable], page 500 を参照してください)。

フェイスは、異なるタイプのディスプレイにたいして、違う外観を指定できます。たとえば、カラーディスプレイではテキストを赤にして、モノクロディスプレイでは太字フォントを使うようにフェ

イスを設定できます。フェイスにたいして複数の外観を指定するには、‘[State]’で呼び出されるメニューで ‘For All Kinds of Displays’を選択してください。

33.1.6 特定のアイテムのカスタマイズ

M-x customize-option RET *option* RET
M-x customize-variable RET *option* RET
: 1 つのユーザーオプション *option*にたいするカスタマイズバッファーをセットアップします。

M-x customize-face RET *face* RET
: 1 つのフェイス *face*にたいするカスタマイズバッファーをセットアップします。

M-x customize-icon RET *face* RET
: 1 つのアイコン *icon*にたいするカスタマイズバッファーをセットアップします。

M-x customize-group RET *group* RET
: 1 つのグループ *group*にたいするカスタマイズバッファーをセットアップします。

M-x customize-apropos RET *regexp* RET
: *regexp*にマッチする、すべてのセッティングとグループにたいするカスタマイズバッファーをセットアップします。

M-x customize-changed RET *version* RET
: Emacs のバージョン *version*以降に意味が変更された (または追加された)、すべてのユーザーオプション、フェイスとグループでカスタマイズバッファーをセットアップします。

M-x customize-saved
: カスタマイズバッファーを使って保存された、すべてのセッティングを含むカスタマイズバッファーをセットアップします。

M-x customize-unsaved
: セットしたが保存していない、すべてのセッティングを含むカスタマイズバッファーをセットアップします。

特定のユーザーオプションをカスタマイズしたい場合は、M-x customize-optionとタイプします。これは変数名を読み取り、そのユーザーオプション 1 つだけのためのカスタマイズバッファーをセットアップします。ミニバッファーから変数名を入力するときは、補完が利用可能ですが、Emacs にロードされた変数名だけが補完されます。

同様に M-x customize-faceを使用して、特定のフェイスをカスタマイズできます。M-x customize-groupを使用して、特定のカスタマイズグループにたいするカスタマイズバッファーをセットアップできます。

M-x customize-aproposは検索条件 — 1 つの単語か、スペースで区切られた複数の単語、または正規表現 — の入力を求め、名前がそれにマッチする、ロードされたすべてのセッティングとグループにたいするカスタマイズバッファーをセットアップします。これはカスタマイズバッファーのトップにあるサーチフィールドを使用するのと同様です (Section 33.1.1 [Customization Groups], page 499 を参照してください)。

新しいバージョンの Emacs にアップグレードしたとき、新しいセッティングをカスタマイズしたり、意味やデフォルト値が変更されたものをセッティングしたいと思うかもしれません。これを行なうには M-x customize-changedを使用して、ミニバッファーから以前の Emacs のバージョンを指

定します。これは指定されたバージョンから変更されたすべてのセッティングとグループを表示するカスタマイズバッファーを作成し、必要ならそれらをロードします。

セッティングを変更した後、その変更が間違いだと気づいたときは、変更を戻すために 2 つのコマンドを使用できます。保存されたカスタマイズのセッティングには、`M-x customize-saved`を使用します。セットしたが保存していないカスタマイズのセッティングには、`M-x customize-unsaved`を使用します。

33.1.7 カスタムテーマ

カスタムテーマ *(Custom themes)* は、1 つの単位として有効または無効にできる、セッティングのコレクションです。カスタムテーマを使用して、さまざまなセッティングコレクション間を簡単に切り替えることができ、あるコンピューターから別のコンピューターへそのようなコレクションを持ち運ぶことができます。

カスタムテーマは、Emacs Lisp ソースファイルとして保存されています。カスタムテーマの名前が *name*なら、そのテーマのファイル名は *name*`-theme.el`です。テーマファイルのフォーマットと、それを作成する方法については、Section 33.1.8 [Creating Custom Themes], page 507 を参照してください。

`M-x customize-themes`とタイプすると、Emacs が認識するカスタムテーマをリストする、`*Custom Themes*`という名前のバッファーに切り替わります。デフォルトでは、Emacs は 2 つの場所からテーマファイルを探します。1 つは `custom-theme-directory`により指定されるディレクトリー (デフォルトは`~/.emacs.d/`)) で、もう 1 つは Emacs がインストールされた場所 (変数 `data-directory`を参照してください) の `etc/themes`というディレクトリーです。後者には Emacs と共に配布されるいくつかのカスタムテーマが含まれており、これらはさまざまなカラースキーム (color schemes) に適合するように、Emacs フェイスをカスタマイズします (しかし、カスタムテーマの目的はこれだけに制限される必要はなく、変数をカスタマイズするのにも使用できることに注意してください)。

Emacs に他の場所からカスタムテーマを探させたい場合は、リスト変数 `custom-theme-load-path`にディレクトリーを追加します。この変数のデフォルト値は `(custom-theme-directory t)` です。ここでシンボル `custom-theme-directory`は、変数 `custom-theme-directory`の値を指定するという特別な意味をもち、`t`はビルトインのテーマディレクトリー `etc/themes`を意味します。`custom-theme-load-path`で指定されるディレクトリーにあるテーマが、`*Custom Themes*`バッファーにリストされます。

`*Custom Themes*`バッファーでは、カスタムテーマの隣のチェックボックスをアクティブにすることにより、カレント Emacs セッションで、そのテーマを有効または無効にできます。カスタムテーマが有効な場合、そのテーマのすべてのセッティング (変数とフェイス) が Emacs セッションで効果をもちます。選択したテーマを将来の Emacs セッションに適用するには、`C-x C-s` (`custom-theme-save`) とタイプするか、‘`[Save Theme Settings]`’ボタンを使用してください。

最初にカスタムテーマを有効にするとき、Emacs はテーマファイルの内容を表示して、本当にロードするか確認を求めます。これはカスタムテーマのロードにより不定な Lisp コードが実行されるからで、テーマが安全だと判っているときだけ yes と答えるべきです。この場合、Emacs は将来のセッションのために、そのテーマが安全だということを記憶するか尋ねます (これは変数 `custom-safe-themes`にテーマファイルの SHA-256 ハッシュ値を保存することにより行なわれます)。すべてのテーマを安全なものとして扱いたい場合は、変数の値を `t`) に変更します。(ディレクトリー `etc/themes` の)Emacs と共に配布されるテーマは、このチェックから除外されていて、常に安全だと判断されます。

カスタムテーマのセッティングと保存は、変数 `custom-enabled-themes`をカスタマイズすることにより機能します。この変数の値は、カスタムテーマ名 (`tango`のような Lisp シンボル) のリスト

です。custom-enabled-themesのセットに*Custom Themes*バッファーを使用するかわりに、たとえばM-x customize-optionのような通常のカスタマイズインターフェースを使用して、変数をカスタマイズできます。カスタムテーマ自身では、custom-enabled-themesをセットできないことに注意してください。

カスタマイズバッファーを通じて行なう任意のカスタマイズは、テーマのセッティングより優先されます。これによりテーマのセッティングを簡単にオーバーライドできます。2 つの異なるテーマのセッティングがオーバーラップする場合には、custom-enabled-themesで先に指定されたテーマが優先されます。カスタマイズバッファーでは、カスタムテーマによりセッティングがデフォルトから変更されているときは、‘State’には‘STANDARD’ではなく‘THEMED’が表示されます。

M-x load-themeとタイプすることにより、カレント Emacs セッションで特定のカスタムテーマを有効にできます。これはテーマ名の入力を求め、テーマファイルからテーマをロードし、それを有効にします。すでにテーマファイルがロードされているときは、M-x enable-themeとタイプすることにより、ファイルをロードせずにテーマを有効にできます。カスタムテーマを無効にするには、M-x disable-themeとタイプしてください。

カスタムテーマの説明を見るには、*Custom Themes*バッファーのその行で、?とタイプするか、Emacs の任意のバッファーで M-x describe-themeとタイプしてテーマ名を入力してください。

一部のテーマには変種 (light と dark の 2 つのみの場合が多い) があります。M-x theme-choose-variantを使えば他の変種に切り替えることができます。カレントでアクティブなテーマの変種が 1 種類しかなければ、そのテーマが選択されます。2 つ以上の変種がある場合には、どれに切り替えるか入力を求めます。

theme-choose-variantはアクティブなテーマが単一の場合のみ機能することに注意してください。

33.1.8 カスタムテーマの作成

M-x customize-create-themeとタイプすることにより、カスタマイズバッファーと似たインターフェースを使用して、カスタムテーマを定義できます。これは*Custom Theme*という名前のバッファーに切り替えます。これは、一般的な Emacs フェイスをそのテーマに挿入するかも尋ねます (カスタムテーマは、フェイスをカスタマイズするのに使用される場合があるので便利です)。これに no と答えると、そのテーマには最初は何もセッティングが含まれません。

*Custom Theme*バッファーの上部には、テーマ名と説明を入力できる、編集可能フィールドがあります。‘user’を除く任意の名前を指定できます。説明は、テーマにたいして M-x describe-themeを呼び出したときに表示される文です。最初の行は 1 センテンスの概要であるべきです。M-x customize-themesにより作成されたバッファーでは、このセンテンスがテーマ名のとなりに表示されます。

テーマに新しいセッティングを追加するには、‘[Insert Additional Face]’ボタンか、‘[Insert Additional Variable]’ボタンを使用します。これらのボタンはミニバッファーを使用して、補完つきでフェイス名または変数名を読み取り、そのフェイスまたは変数にたいするカスタマイズエントリーを挿入します。通常のカスタマイズバッファーと同じ方法で、変数の値またはフェイスの属性を編集できます。テーマからフェイスまたは変数を削除するには、名前の横のチェックボックスのチェックを外してください。

カスタムテーマのフェイスや変数を指定した後は、C-x C-s (custom-theme-write) とタイプするか、そのバッファーの‘[Save Theme]’ボタンを使用します。これは custom-theme-directory のディレクトリーに、*name*-theme.el(*name*はテーマ名) という名前で、テーマファイルを保存します。

*Custom Theme*バッファーから、‘[Visit Theme]’ボタンをアクティブにしてテーマ名を指定することにより、既存のカスタムテーマの閲覧と編集ができます。‘[Merge Theme]’ボタンを使用して、他のテーマのセッティングをバッファーに追加することもできます。‘[Merge Theme]’ボタンを使用して、‘user’という名前の特別なテーマ名を指定することにより、非テーマセッティングをカスタムテーマにインポートできます。

テーマファイルは単なる Emacs Lisp ソースファイルで、カスタムテーマのロードは Lisp ファイルをロードすることにより機能します。したがって*Custom Theme*バッファーを使用するかわりに、テーマファイルを直接編集することもできます。詳細は、Section “Custom Themes” in *The Emacs Lisp Reference Manual* を参照してください。

33.2 変数

変数 *(variable)* とは、値をもつ Lip シンボルです。このようなシンボルの名前は、変数名 *(variable name)* とも呼ばれます。変数名には、ファイルに記述できる任意の文字を含めることもできますが、ほとんどの変数名は通常の単語をハイフンで区切って構成されます。

変数の名前には、その変数の役割を簡単に説明する役目があります。ほとんどの変数はドキュメント文字列 *(documentation string)* ももっていて、これは変数の目的、どのような種類の値をもつべきか、値がどのように使用されるかを説明します。ヘルプコマンド C-h v (describe-variable) を使用して、このドキュメントを閲覧できます。Section 33.2.1 [Examining], page 508 を参照してください。

Emacs は内部の記録維持のために多くの Lisp 変数を使用しますが、非プログラマーに一番興味があるのはユーザーが変更することを意図した Lisp 変数であり、これらはカスタマイズ可能変数 *(customizable variables)* やユーザーオプション *(user options)* と呼ばれます (Section 33.1 [Easy Customization], page 499 を参照してください)。以下のセクションでは、カスタマイズのためのインターフェース以外から変数をセットする方法など、他の観点から Emacs 変数を説明します。

(少数の例外を除き)Emacs Lisp では、任意の変数は任意のタイプの値をもつことができます。しかし多くの変数は、特定のタイプの値を割り当てられた場合だけ意味をもちます。たとえば kill リングの最大長さを指定する kill-ring-maxの値としては、数字だけが意味をもちます。kill-ring-maxの値として文字列を与えた場合、C-y (yank) のようなコマンドはエラーをシグナルするでしょう。一方、タイプを気にしない変数もあります。たとえば、変数の値が nilのときはある効果をもたらし、非 nilのときは別の効果をもたらす場合、シンボル nil以外の任意の値は、そのタイプに関わらず 2 番目の効果をもたらします (慣例により、非 nil値を指定するために、通常は値 t — これは “true” が由来です — を使用します)。カスタマイズバッファーを使用して変数をセットする場合、無効なタイプを与えてしまう心配はありません。カスタマイズバッファーでは通常、意味のある値しか入力できないからです。判別がつかないときは、その変数が期待するどの種類の値かを見るために、C-h v (describe-variable) を使用して、変数のドキュメント文字列をチェックしてください (Section 33.2.1 [Examining], page 508 を参照してください)。

33.2.1 変数の確認とセット

C-h v *var* RET
: 変数 *var*の値とドキュメントを表示します (describe-variable)。

M-x set-variable RET *var* RET *value* RET
: 変数 *var*の値を *value*に変更します。

変数の値を調べるには、C-h v (describe-variable) を使用します。これはミニバッファーを使用して補完つきで変数名を読み取り、、変数の値とドキュメントの両方を表示します。たとえば、

```
C-h v fill-column RET
```

これは以下のような出力を表示します:

```
fill-column is a variable defined in ‘C source code’.
Its value is 70

  Automatically becomes buffer-local when set.
  This variable is safe as a file local variable if its value
  satisfies the predicate ‘integerp’.
  Probably introduced at or before Emacs version 18.

Documentation:
Column beyond which automatic line-wrapping should happen.
Interactively, you can set the buffer local value using C-x f.

You can customize this variable.
```

‘You can customize the variable’の行は、この変数がユーザーオプションであることを示します。C-h vはユーザーオプションだけに制限されません。これはカスタマイズ可能でない変数にも使用できます。

特定のカスタマイズ可能な変数をセットする一番簡単な方法は、M-x set-variableです。これはミニバッファーで変数名を読み取り (補完つき)、次にミニバッファーを使用して新しい値にたいする Lisp 式を読み取ります (M-nを使用してミニバッファーで編集するために、古い値を挿入することができます)。たとえば、

```
M-x set-variable RET fill-column RET 75 RET
```

これは fill-columnを 75 にセットします。

M-x set-variableはカスタマイズ可能な変数に制限されていますが、以下のような Lisp 式で任意の変数をセットできます:

```
(setq fill-column 75)
```

このような式を実行するには、M-: (eval-expression) とタイプして、ミニバッファーで式を入力します (Section 24.9 [Lisp Eval], page 329 を参照してください)。かわりに*scratch*バッファーに移動して、式をタイプしてから C-jとタイプすることもできます (Section 24.10 [Lisp Interaction], page 330 を参照してください)。

変数のセットは、Emacs のカスタマイズと同様、特に明記しない限りは、カレント Emacs セッションだけに影響します。将来のセッションのために変数を変更する唯一の方法は、初期化ファイルにそれを記述することです (Section 33.4 [Init File], page 528 を参照してください)。

init ファイルでカスタマイズ可能な変数をセットしていて Customize インターフェイスを使いたくない場合には、たとえば以下のように setoptを使うことができます:

```
(setopt fill-column 75)
```

これは setqと同じように機能しますが、その変数に特別なセッター関数 (setter function) があれば、setoptは自動的にそちらを実行します。一方カスタマイズ可能ではない変数に setoptを使うこともできますが、setqを使うより効率性に劣ります。

33.2.2 フック

フック *(hook)* とは、Emacs をカスタマイズするための重要な仕組みです。フックは関数のリストを保持する Lisp 変数で、これらの関数は、ある定められたタイミングで呼び出されます (これは、フッ

クを実行する *(running the hook)*)、と呼ばれます)。リストの中の個別の関数は、そのフックのフック関数 *(hook functions)* と呼ばれます。たとえばフック `kill-emacs-hook`は、Emacs を終了する直前に実行されます (Section 3.2 [Exiting], page 16 を参照してください)。

ほとんどのフックはノーマルフック *(normal hooks)* です。これは、Emacs がフックを実行するとき、フック関数が引数なしで順に呼び出します。わたしたちは、ほとんどのフックをノーマルフックに保つために努力しているので、あなたはこれらのフックを一貫した方法で使用することができます。変数名の最後が '`-hook`'の変数は、ノーマルフックです。

多くはありませんが、アブノーマルフック *(abnormal hooks)* もあります。アブノーマルフックは、名前の最後が '`-hook`'ではなく '`-functions`'です (古いコードの中には時代遅れのサフィックス '`-hooks`'を使うものもあります)。これらのフックがアブノーマルな理由は、関数が呼び出される方法にあります— もしかしたら引数が与えられているかもしれず、ことによると関数が返す値が何かに使用されるかもしれません。たとえば `find-file-not-found-functions`はアブノーマルです。なぜならフック関数のうちの 1 つが非 `nil`値を返した場合、残りの関数は呼び出されないからです (Section 15.2 [Visiting], page 146 を参照してください)。アブノーマルフック変数のドキュメントには、フック関数がどのように使用されるかの説明があります。

他の Lisp 変数と同じように、`setq`でフック変数をセットすることもできますが、フック (ノーマルとアブノーマルの両方) に関数を追加するための推奨される方法は、以下の例で示されるような、`add-hook`を使う方法です。詳細は、Section "Hooks" in *The Emacs Lisp Reference Manual* を参照してください。

ほとんどのメジャーモードは初期化の最終ステップで、1 つ以上のモードフック *(mode hooks)* を実行します。モードフックは個々のモードの振る舞いをカスタマイズするための便利な方法で、常にノーマルフックです。たとえば、以下は Text モードと、Text モードを基礎とする他のモードで、Auto Fill モードをオンにするフックをセットアップする方法です:

```
(add-hook 'text-mode-hook 'auto-fill-mode)
```

これは、引数を与えられない場合にマイナーモードを有効にする `auto-fill-mode`を呼び出すことにより機能します (Section 20.2 [Minor Modes], page 241 を参照してください)。次に、Text モードを基礎とする LaTeX モードでは Auto Fill モードをオンにしたくない場合、以下の行を追加してこれを行なうことができます:

```
(add-hook 'latex-mode-hook (lambda () (auto-fill-mode -1)))
```

ここでは、無名関数 (anonymous function。Section "Lambda Expressions" in *The Emacs Lisp Reference Manual* を参照してください) を構築するために、特別なマクロ `lambda`を使用しており、`auto-fill-mode`に`-1`を与えて呼び出すことにより、マイナーモードを無効にしています。LaTeX モードは、`text-mode-hook`を実行した後に、`latex-mode-hook`モードを実行するので、その結果 Auto Fill モードが無効になります。

以下はもっと複雑な例で、C コードのインデントをカスタマイズするのにフックを使う方法です:

```
(setq my-c-style
  '((c-comment-only-line-offset . 4)
    (c-cleanup-list . (scope-operator
                       empty-defun-braces
                       defun-close-semi))))

(add-hook 'c-mode-common-hook
  (lambda () (c-add-style "my-style" my-c-style t)))
```

メジャーモードフックは、それを元のモードとして派生された *(derived)* 他のメジャーモードにも適用されます (Section "Derived Modes" in *The Emacs Lisp Reference Manual* を参照して

ください)。たとえば HTML モード (Section 22.12 [HTML Mode], page 273 を参照してください) は Text モードから派生しており、HTML モードが有効になるときは、`html-mode-hook`を実行する前に `text-mode-hook`が実行されます。これは 1 つのフックを複数の関連するモードに作用させるための便利な方法を提供します。特に任意のプログラミング言語にたいしてフック関数を適用したい場合は、それを `prog-mode-hook`モードに追加します。Prog モードは、それを継承する他のメジャーモードと比較すると、ほとんど何も行なわないメジャーモードで、まさにこの目的のために存在します。

実行される順番に依存しないようにフック関数をデザインするのがベストです。実行順への依存はトラブルを招きます。しかし実行順は予測可能です。フック関数はフックに登録された順に実行されます。

何度も `add-hook`を呼び出すことにより、さまざまな異なるバージョンのフック関数を追加した場合、追加されたすべてのバージョンのフック関数がフック変数に残ることを忘れないでください。`remove-hook`を呼び出すことにより関数を個別にクリアーするか、`(setq` *`hook-variable`* `nil)`ですべてのフック関数を削除できます。

フック変数がバッファーローカルな場合、グローバル変数のかわりにバッファーローカル変数が使用されます。しかしバッファーローカル変数が要素 `t`を含む場合は、グローバル変数も同様に実行されます。

33.2.3 ローカル変数

`M-x make-local-variable RET` *`var`* `RET`
: 変数 *var*が、カレントバッファーでローカル値をもつようにします。

`M-x kill-local-variable RET` *`var`* `RET`
: 変数 *var*が、カレントバッファーでグローバル値を使うようにします。

`M-x make-variable-buffer-local RET` *`var`* `RET`
: 変数 *var*がセットされた時点で、カレントバッファーにたいしてローカルになるようマークします。

ほとんどの変数は、特定の Emacs バッファーにたいしてローカル *(local)* にすることができます。これは、そのバッファーでの変数の値が、他のバッファーでの変数の値とは、独立していることを意味します。多くはありませんが、常にバッファーごとにローカルな変数もあります。他のすべての Emacs 変数は、バッファーで変数をローカルにしていないかぎりは、すべてのバッファーに効果を及ぼすグローバル *(global)* な値をもちます。

`M-x make-local-variable`は変数名を読み取り、それをカレントバッファーにたいしてローカルにします。その後、このバッファーで変数の値を変更しても他のバッファーには影響せず、変数のグローバル値を変更してもこのバッファーには影響しなくなります。

`M-x make-variable-buffer-local`は、変数がセットされたとき自動的にローカルになるように、変数をマークします。より正確には、1 度この方法で変数がマークされると、通常の方法による変数のセットは、最初に自動的に `make-local-variable`を呼び出します。このような変数をパーバッファー *(per-buffer:* バッファーごと*)* 変数と呼びます。Emacs の多くの変数は、通常はパーバッファーです。変数のドキュメント文字列には、いつこれを行なうかが記述されています。パーバッファー変数のグローバル値は、通常は任意のバッファーには影響しませんが、それでもまだ意味があります。グローバル値は、新しいバッファーにたいする、この変数の初期値として使用されます。

メジャーモード (Section 20.1 [Major Modes], page 240 を参照してください) は常に変数をセットする前に、変数をローカルにします。あるバッファーでメジャーモードを変更しても、他のバッファーに影響がないのは、これが理由です。マイナーモードは変数をセットすることにより機能しま

す— 通常、各マイナーモードは 1 つの制御変数 (controlling variable) をもっていて、この変数が非 nilの場合はモードが有効になります (Section 20.2 [Minor Modes], page 241 を参照してください)。多くのマイナーモードにたいして制御変数はパーバッファーであり、したがって常にバッファーローカルです。そうでない場合、他の変数と同様に特定のバッファーで変数をローカルにできます。

多くはありませんが、バッファーでローカルにできない (かわりに各ディスプレイにたいして常にローカル。Section 18.10 [Multiple Displays], page 205 を参照してください) 変数も存在します。そのような変数をバッファーローカルにしようとすると、エラーメッセージが表示されます。

M-x kill-local-variableは、指定された変数が、カレントバッファーにたいしてローカルであることを終了させます。その後は、そのバッファーにたいして、その変数のグローバル値が効力をもちます。メジャーモードのセットにより、数少ないパーマネントローカル *(permanent locals:* 永久にローカル*)* な変数を除いて、そのバッファーのすべてのローカル変数は kill されます。

変数がカレントバッファーでローカル値をもつかに関わらず、変数にグローバル値をセットするには、Lisp コンストラクト setq-defaultを使用することができます。このコンストラクトは setqと同じように使用されますが、(もしあれば) ローカル値のかわりにグローバル値をセットします。カレントバッファーがローカル値をもつ場合、新しいグローバル値は他のバッファーに切り替えるまで見えないでしょう。以下は例です:

```
(setq-default fill-column 75)
```

setq-defaultは、make-variable-buffer-localでマークされた変数のグローバル値をセットする唯一の方法です。

Lisp プログラムは変数のデフォルト値を得るために、default-valueを使用することができます。この関数はシンボルを引数として受け取り、それのデフォルト値を返します。引数は評価されるので、通常は明示的にクォートする必要があります。たとえば、以下は fill-columnのデフォルト値を得る方法です:

```
(default-value 'fill-column)
```

33.2.4 ファイル内のローカル変数

ファイルに、Emacs でそのファイルを編集するときに使用するローカル変数の値を指定できます。ファイルを visit するか、メジャーモードをセットすることにより、Emacs はローカル変数指定をチェックします。これは自動的にこれらの変数をバッファーにたいしてローカルにし、ファイルで指定された値にセットします。

あるファイルのディレクトリーにたいしてディレクトリーローカル変数 (Section 33.2.5 [Directory Variables], page 516 を参照) が指定されている場合、ファイルローカル変数はそれをオーバーライドします。

33.2.4.1 ファイル変数の指定

ファイルローカル変数を指定するには 2 つの方法があります。1 つは最初の行に記述する方法で、もう 1 つはローカル変数リストを使用する方法です。以下は最初の行でこれらを指定する方法の例です:

```
-*- mode: modename; var: value; ... -*-
```

この方法により、任意の数の変数/値 (variable/value) ペアーを指定できます。各ペアーはコロンとセミコロンで区切ります。特別な変数/値ペアー mode: *modename*;が与えられた場合にはメジャーモードを指定します (サフィックス “-mode” なし)。*value*は文字列として使用されて、評価はされません。

手作業でエントリーを追加するかわりに、M-x add-file-local-variable-prop-lineを使用することができます。このコマンドは変数と値の入力を求め、適切な方法で最初の行にこれらを追加します。M-x delete-file-local-variable-prop-lineは変数の入力を求め、最初の行から

変数のエントリーを削除します。コマンド M-x copy-dir-locals-to-file-locals-prop-line は、カレントのディレクトリーローカル変数を最初の行にコピーします (Section 33.2.5 [Directory Variables], page 516 を参照してください)。

以下は、最初の行で Lisp モードを指定して、2 つの変数に数値をセットする例です:

```
;; -*- mode: Lisp; fill-column: 75; comment-column: 50; -*-
```

modeの他に、ファイル変数として特別な意味をもつキーワードは coding、unibyte、evalです。これらは以下で説明します。

シェルスクリプトでは、最初の行はスクリプトのインタープリターの識別に使用されるので、ローカル変数をそこに置くことはできません。これに対処するために、Emacs は最初の行がインタープリターを指定しているときは、*2* 行目からローカル変数指定を探します。man pages にも同じことが言えます。man pages は troff プリプロセッサーのリストを指定するマジック文字列 '\"'で始まるからです (しかし、すべてがこれを行なう訳ではありません)。

'-*-'行を使用するのではなく、ファイルの終端付近でローカル変数リスト *(local variables list)* を使用することにより、ファイルローカル変数を定義することもできます。ローカル変数リストは、ファイル終端から 3000 文字以内で開始され、ファイルがページに分かれているときは最後のページになければなりません。

ファイルにローカル変数リストと '-*-'の両方がある場合、Emacs は最初に '-*-'行のすべてを処理してから、ローカル変数リストのすべてを処理します。例外はメジャーモード指定です。Emacs はメジャーモード指定がどこにあろうと、まずそれを適用します。なぜならほとんどのメジャーモードは、初期化部分ですべてのローカル変数を kill するからです。

ローカル変数リストは、文字列 'Local Variables:'を含む行で開始され、文字列 'End:'を含む行で終了します。この間には、以下のように 1 行に変数名と値のペアーが記述されます:

```
/* Local Variables:  */
/* mode: c           */
/* comment-column: 0 */
/* End:              */
```

この例では、各行はプレフィクス '/*'で始まり、サフィックス '*/'で終了します。Emacs は、リストの最初の行のマジック文字列 'Local Variables:'を囲む文字列から、プレフィクスとサフィックスを識別します。その後はリストの他の行で自動的にこれらを破棄します。プレフィクスおよび/またはサフィックスを使用する通常の理由は、そのファイルが意図する他のプログラムが混乱しないように、ローカル変数をコメントに埋め込むためです。上記は、コメントが '/*'で始まり '*/'で終わる C プログラミング言語での例です。

Emacs のローカル変数リストではないが、そのように見えるテキストがある場合は、そのテキストの後にフォームフィード文字 (ページ区切りです。Section 22.4 [Pages], page 254 を参照してください) を挿入して、それを取り消すことができます。Emacs はファイルの最後のページ (つまり最後のページ区切りの後) にあるファイルローカル変数だけを調べます。

ローカル変数を直接タイプするかわりに、コマンド M-x add-file-local-variableを使用することができます。これは変数と値の入力を求め、それらをリストに追加し、'Local Variables:'と、必要なら開始・終了マーカーも追加します。コマンド M-x delete-file-local-variableは、リストから変数を削除します。M-x copy-dir-locals-to-file-localsは、ディレクトリーローカル変数をリストにコピーします (Section 33.2.5 [Directory Variables], page 516 を参照してください)。

'-*-'行と同じように、ローカル変数リストの変数は文字列として使用され、最初に評価されることはありません。長い文字列値をファイル内で複数行に分割したい場合、改行とバックスラッシュを

使用できます (Lisp 文字列定数では無視されます)。各行には、プレフィクスとサフィックスを記述するべきです。たとえ行がその文字列で開始または終了していても、それらはリストを処理するとき取り除かれます。以下は例です:

```
# Local Variables:
# compile-command: "cc foo.c -Dfoo=bar -Dhack=whatever \
#   -Dmumble=blaah"
# End:
```

いくつかの名前は、ローカル変数リスト内で特別な意味をもちます:

- `mode`は、指定されたメジャーモードを有効にします。
- `eval`は、指定された Lisp 式を評価します (式が返す値は無視されます)。
- `coding`は、このファイルでの文字コード変換にたいするコーディングシステムを指定します。Section 19.5 [Coding Systems], page 223 を参照してください。
- `unibyte`の値が `t`の場合、Emacs Lisp のロードとコンパイルを unibyte モードで行ないます。Section “Disabling Multibyte Characters” in *GNU Emacs Lisp Reference Manual* を参照してください。

これら 4 つのキーワードは、実際には変数ではありません。他のコンテキストでこれらをセットしても、特別な意味はありません。

Emacs の複数のバージョン間にまたがってファイルを編集するような場合に、新しいバージョンの Emacs ではそのファイルを処理する新たなモードが導入されたとします。新しいバージョンの Emacs では新たなモード (たとえば `my-new-mode`)、そして古いバージョンの Emacs では古いモード (たとえば `my-old-mode`) にフォールバックするという具合に、`mode`エントリーを複数使うことができます。1 行目でモードを有効にしている場合には、以下のように記述します:

```
-*- mode: my-old; mode: my-new -*-
```

Emacs は見つかったモードの中から最後に定義されたモードを使用するので、古いバージョンの Emacs では `my-new-mode`は無視されますが、`my-new-mode`が定義されているバージョンの Emacs では `my-old-mode`が無視されます。同様にして、ファイル終端のローカル変数ブロックでは以下のように記述します:

```
Local variables:
mode: my-old
mode: my-new
```

マイナーモードにたいして `mode`キーワードを使用しないでください。ローカル変数リストでマイナーモードを有効または無効にするには、`eval`でモードコマンドを実行する Lisp 式を指定します (Section 20.2 [Minor Modes], page 241 を参照)。たとえば以下のローカル変数リストは、引数なし (引数に 1 を指定しても同じことを行なう) で `eldoc-mode`を呼び出すことにより、ElDoc モード (Section 23.6.3 [Programming Language Doc], page 299 を参照) を有効にし、引数 −1 で `font-lock-mode`(Section 11.13 [Font Lock], page 89 を参照) を呼び出すことにより、Font Lock モードを無効にする例です。

```
;; Local Variables:
;; eval: (eldoc-mode)
;; eval: (font-lock-mode -1)
;; End:
```

しかしこの方法でマイナーモードを指定するのは、間違っている場合もあることに注意してください。マイナーモードは個人の好みを表しており、そのファイルを編集するユーザーにあなたの好みを強制

するのは、不適切かもしれません。状況に依存して自動的にマイナーモードを有効または無効にしたい場合は、たいていメジャーモードフックでこれを行なう方がよいのです (Section 33.2.2 [Hooks], page 509 を参照してください)。

ローカル変数と、ファイル名とファイル内容にしたがったバッファーのメジャーモード (もしあればローカル変数リストも) をリセットするには、コマンド `M-x normal-mode`を使用します。Section 20.3 [Choosing Modes], page 244 を参照してください。

33.2.4.2 安全なファイル変数

ファイルローカル変数が危険な場合もあります。他の誰かのファイルを visit するとき、そのファイルのローカル変数リストが Emacs に何を行なうか、告げるものはありません。`eval` "variable" や、その他の `load-path`などにたいする不正な値は、実行する意図がない Lisp コードを実行するかもしれません。

したがって、安全と判っていないファイルローカル変数を発見した場合、Emacs はファイルのローカル変数リスト全体を表示して、それらをセットする前に継続するか尋ねます。`y`または`SPC`をタイプすると、ローカル変数リストは効果をもち、`n`の場合は無視します。Emacs がバッチモード (Section C.2 [Initial Options], page 576 を参照してください) で実行されている場合、Emacs は確認することができないので、`n`と応えられたとみなします。

Emacs は通常、特定の変数/値ペアーが安全だと認識できます。たとえば `comment-column`や`fill-column`には、任意の整数値を与えても安全です。ファイルが安全だと判っている変数/値ペアーだけを指定する場合、Emacs はそれらをセットする前に確認を求めません。そうでない場合、確認プロンプトで`!`とタイプすることにより、このファイル内のすべての変数/値ペアーが安全なことを記録するよう Emacs に指示できます。その後、Emacs が同じファイルまたは別のファイルで、これらの変数/値ペアーに出会うと、これらを安全だとみなします。

同意を求めるプロンプトで `i`とタイプすることによって、ファイル内のすべての変数/値ペアーを永続的に無視するよう Emacs に指示することもできます。それ以降は、これらのペアーはそのファイルと他のすべてのファイルで無視されるようになります。

`load-path`のようないくつかの変数は、特に危険 *(risky)* だと判断されます。これらをローカル変数として指定すべき理由はほとんどなく、それらを変更するのは危険です。ファイルに危険なローカル変数だけが含まれる場合、Emacs は確認プロンプトで`!`の選択肢を提示することも、それを受け入れることもしません。ファイル内のいくつかのローカル変数が危険で、いくつかの変数は潜在的に安全ではない場合は、プロンプトで`!`を入力できます。これはすべての変数に適用されますが、危険ではない変数だけを将来のセッションのために安全とマークします。もし危険な変数を安全な値として記録したいと本当に望むなら、'`safe-local-variable-values`'をカスタマイズすることによりこれを行ないます (Section 33.1 [Easy Customization], page 499 を参照)。同様に永続的に無視されるべき危険な変数の値を記録したければ、`ignored-local-variable-values`をカスタマイズしてください。

変数 `enable-local-variables`により、Emacs がローカル変数を処理する方法を変更できます。デフォルト値は `t`で、これは上述の振る舞いを指定します。`nil`の場合、Emacs は単にすべてのファイルローカル変数を無視します。`:safe`は安全な値だけを使用して、残りは無視します。`:all`は Emacs にたいして値が安全か否かに関わらず、すべてのファイルローカル変数をセットするよう指示します (恒常的に使用しないよう推奨)。他の値の場合、安全と判っている値かどうかの決定を試みずに、ローカル変数をもつ各ファイルごとに尋ねます。

変数 `enable-local-eval`は、Emacs が `eval`変数を処理するかどうかを制御します。`enable-local-variables`のように、変数に対する可能な値は 3 つで、`t`、`nil`、およびそれ以外

です。デフォルトは、tやnilではないmaybeで、通常 Emacs は eval変数を処理するときに確認を求めます。

例外として、評価する任意の eval形式が、変数 safe-local-eval-formsで指定された形式の場合、Emacs は確認を求めません。

33.2.5 ディレクトリーごとのローカル変数

大きなソフトウェアプロジェクトでのディレクトリーツリーのような、特定のディレクトリーや、それのサブディレクトリーのすべてのファイルにたいして、同じローカル変数を定義したいことがあるかもしれません。これはディレクトリーローカル変数 *(directory-local variables)* で行なうことができます。ファイルローカル変数はディレクトリーローカル変数をオーバーライドするので、あるディレクトリー内のファイルに特別なセッティングが必要な場合は、ディレクトリー変数でそのディレクトリー内の大多数にたいするセッティングを指定してから、ファイルローカル変数により、少数のファイルにたいしてオーバーライドを要する一般的なセッティングを定義できます。

ディレクトリーローカル変数を定義する通常の方法は、そのディレクトリーに.dir-locals.el[1] を配す方法です。そのディレクトリー、またはそれのサブディレクトリーの任意のファイルを Emacs が visit するとき、.dir-locals.elで指定されたディレクトリーローカル変数が、あたかもそのファイルのファイルローカル変数 (Section 33.2.4 [File Variables], page 512 を参照してください) として定義されたかのように、ファイルに適用されます。Emacs は visit されたファイルのディレクトリーから、ディレクトリーツリーを上に移動しながら.dir-locals.elを検索します。スローダウンを避けるために、検索はリモートファイルをスキップします。必要なら、変数 enable-remote-dir-localsを tにセットして、検索範囲をリモートファイルに広げることができます。

Emacs が追加でロードする.dir-locals-2.elが存在する場合は、それを使用することもできます。これは.dir-locals.elがバージョンコントロールの共有ディレクトリー配下にあり、個人的なカスタマイズに使用できないときに有用です。

.dir-locals.elは、特別な構成のリストをもちます。これはモード名 (シンボルで指定) を alist(Association Lists: 連想リスト。Section “Association Lists” in *The Emacs Lisp Reference Manual* を参照してください) にマップします。各 alist エントリーは、変数名と、指定されたメジャーモードが有効なときに、その変数に割り当てるディレクトリーローカル値からなります。モード名のかわりに ‘nil’を指定でき、これは alist が任意のモードで適用されることを意味します。サブディレクトリー (文字列で指定) を指定することもできます。この場合、そのサブディレクトリーのすべてのファイルに alist が適用されます。

以下は、.dir-locals.elファイルの例です:

```
((nil . ((indent-tabs-mode . t)
         (fill-column . 80)
         (mode . auto-fill)))
 (c-mode . ((c-file-style . "BSD")
            (subdirs . nil)))
 ("src/imported"
  . ((nil . ((change-log-default-name
              . "ChangeLog.local"))))))
```

これはディレクトリーツリーの任意のファイルにたいして変数 ‘indent-tabs-mode’および fill-columnをセットして、任意の C ソースファイルにたいしてインデントスタイルをセットしま

[1] MS-DOS では、DOS ファイルシステムの制限により、このファイルの名前は_dir-locals.elになります。ファイルシステムによりファイル名が 8+3 に制限されている場合、OS によりファイル名が_dir-loc.elに切り詰められるでしょう。

す。特別な要素 modeは有効にするマイナーモードを指定します。したがって (mode . auto-fill) えはマイナーモード auto-fill-modeの有効化が必要なことを指定します。特別な要素 subdirsは変数ではありません。これは特別なキーワードで、C モードのセッティングがカレントディレクトリーだけに適用され、任意のサブディレクトリーには適用されないことを示します。最後に、これはsrc/importedサブディレクトリー内の任意のファイルにたいして、違う ChangeLogファイル名を指定します。

異なるモード名やディレクトリーを使用する変数にたいして、.dir-locals.elファイル内に複数の値が含まれる場合には、汎用的なモードと比較してより特化したモードにたいする値が優先して適用されるでしょう。更にディレクトリー下に指定された値は、より高い優先度をもちます。たとえば:

```
((nil . ((fill-column . 40)))
 (c-mode . ((fill-column . 50)))
 (prog-mode . ((fill-column . 60)))
 ("narrow-files" . ((nil . ((fill-column . 20))))))
```

c-modeは prog-modeから派生しているので、c-modeを使用するファイルは prog-modeにもマッチします。しかし C のファイルは prog-modeより特化したモード名なので、fill-columnに使用される値は 50になるでしょう。prog-modeから派生する他のモードを使用するファイルは、60を使用します。narrow-filesディレクトリー配下のファイルでは、モードエントリーよりディレクトリーエントリーが優先されるので、たとえ c-modeを使用するファイルでも、値 20が使用されることになります。

.dir-locals.el内では mode、eval、unibyteを指定できます。これらの変数は、ファイルローカル変数のときとオ同じ意味をもちます。codingは、ディレクトリーローカル変数としては指定できません。Section 33.2.4 [File Variables], page 512 を参照してください。

.dir-locals.el内のスペシャルキー auto-mode-alistで、ファイルのメジャーモードをセットできます。これは変数 auto-mode-alistと同様に機能します (Section 20.3 [Choosing Modes], page 244 を参照)。たとえば以下はそのディレクトリー内の.defソースファイルを C モードにするよう Emacs に指示する例です:

```
((auto-mode-alist . (("\\.def\\'" . c-mode))))
```

.dir-locals.elファイルを手で編集するかわりに、コマンド M-x add-dir-local-variable を使用できます。これはモード名またはサブディレクトリー、および変数名と値の入力を求め、ディレクトリーローカル変数を定義するエントリーを追加します。M-x delete-dir-local-variable は、エントリーを削除します。M-x copy-file-locals-to-dir-localsは、カレントファイル内のファイルローカル変数を、.dir-locals.elにコピーします。

ディレクトリーローカル変数を指定する他の方法は、dir-locals-set-class-variables関数を使用して、ディレクトリークラス *(directory class)* の中に、変数/値ペアーのグループを定義する方法です。その後、dir-locals-set-directory-class関数を使用して、そのクラスに対応するディレクトリーを Emacs に指示します。これらの関数呼び出しは通常、初期化ファイルで行なわれます (Section 33.4 [Init File], page 528 を参照してください)。この方法は、何らかの理由でディレクトリーに.dir-locals.elを置けないときに便利です。たとえば、この方法で書き込み不可なディレクトリーにセッティングを適用できます:

```
(dir-locals-set-class-variables 'unwritable-directory
   '((nil . ((some-useful-setting . value)))))

(dir-locals-set-directory-class
   "/usr/include/" 'unwritable-directory)
```

変数にたいしてディレクトリーローカル値とファイルローカル値の両方が指定された場合、ファイルローカル値が効果をもちます。安全ではないディレクトリーローカル値は、安全でないファイルローカル値と同じ方法で扱われます (Section 33.2.4.2 [Safe File Variables], page 515 を参照してください)。

ディレクトリーローカル変数は、Dired バッファー (Chapter 27 [Dired], page 380 を参照してください) のような、ファイルを直接 visit していないが、ディレクトリーで処理を行なうバッファーにたいしても効果があります。

33.2.6 接続ごとのローカル変数

ほとんどの変数はローカルマシンの状況を反映します。リモートのデフォルトディレクトリーをもつバッファー操作時には、それらの変数が異なる値を使用しなければならないときが時折あります。`shell`を呼び出す際の挙動について考えてみましょう。ローカルのマシンでは`/bin/bash`を使用して termcap を用いるかもしれませんが、リモートのマシンでは`/bin/ksh`と terminfo かもしれないのです。

これは接続ローカル変数 *(connection-local variables)* で実現することができます。ディレクトリーローカル変数とファイルローカル変数は接続ローカル変数をオーバーライドします。安全ではない接続ローカル値は、安全でないファイルローカル値と同じ方法で扱われます (Section 33.2.4.2 [Safe File Variables], page 515 を参照してください)。

接続ローカル変数は `connection-local-set-profile-variables`関数を使用して変数/値ペアのグループとしてプロファイル *(profile)* 内に宣言されます。関数 `connection-local-set-profiles`はリモートマシンを識別する与えられた条件にたいしてプロファイルをアクティブにします。

```
(connection-local-set-profile-variables 'remote-terminfo
   '((system-uses-terminfo . t)
     (comint-terminfo-terminal . "dumb-emacs-ansi")))

(connection-local-set-profile-variables 'remote-ksh
   '((shell-file-name . "/bin/ksh")
     (shell-command-switch . "-c")))

(connection-local-set-profile-variables 'remote-bash
   '((shell-file-name . "/bin/bash")
     (shell-command-switch . "-c")))

(connection-local-set-profiles
   '(:application tramp :machine "remotemachine")
   'remote-terminfo 'remote-ksh)
```

このコードでは`remote-terminfo`、`remote-ksh`そして`remote-bash`という 3 つの異なるプロファイルを宣言しています。プロファイル`remote-terminfo`と`remote-ksh`は、ホスト名が regexp の`"remotemachine"`にマッチするリモートのデフォルトディレクトリーをもつすべてのバッファーに適用されます。このような条件はプロパティ`:protocol` (Tramp の手法) や`:user` (リモートユーザー名) も区別できます。`nil`という条件はリモートデフォルトディレクトリーをもつすべてのバッファーにマッチします。

33.3 キーバインディングのカスタマイズ

このセクションでは、キーをコマンドにマップするキーバインド *(key bindings)* と、そのキーバインドを記録するキーマップ *(keymaps)* を説明します。それに init ファイルを編集して、キーバインドをカスタマイズする方法も説明します (Section 33.3.6 [Init Rebinding], page 522 を参照してください)。

ほとんどのモードは自身のキーバインディングを定義するので、モードをアクティブにすることにより、あなたのカスタムキーバインディングがオーバーライドされるかもしれません。いくつかのキーはユーザー定義バインディングのために予約されており、モードはそれらを使用してはならないので、これらのキーはこの問題にたいして安全です。予約済みのキーは、`C-c`と英字 (大文字と小文字の両方)、修飾キーなし (Section 33.3.7 [Modifier Keys], page 523 を参照) のファンクションキーの `F5`から `F9`です。

33.3.1 キーマップ

Section 2.4 [Commands], page 14 で説明されているように、各 Emacs コマンドは、対話的に使用することを条件として定義された Lisp 関数です。すべての Lisp 関数と同様に、コマンドは小文字とハイフンからなる関数名をもちます。

キーシーケンス *(key sequence)* — 短くはキー *(key)* — とは、1 つの単位として意味をもつ、連続する入力イベント *(input events)* のことです。入力イベントとは文字、ファンクションキー、マウスボタン — つまりコンピューターに送ることができるすべての入力のことです。キーシーケンスは、それが何のコマンドを実行するかを指示するバインディング *(binding)* により、意味をもちます。

キーシーケンスとコマンド関数との間のバインディングは、*keymaps(*キーマップ*)* と呼ばれるデータ構造に記録されます。Emacs には多くの keymaps があり、それぞれが特別の機会に使用されます。

一番重要なキーマップは、グローバルキーマップ *(global keymap)* です。なぜならグローバルキーマップは常に効果があるからです。グローバルキーマップは Fundamental モードにたいしてキーを定義します (Section 20.1 [Major Modes], page 240 を参照してください)。これらの定義のほとんどは、、ほとんどすべてのメジャーモードでは一般的です。メジャーモードまたはマイナーモードは、いくつかのキーにたいするグローバル定義をオーバーライドするために、それぞれ独自の keymap をもつことができます。

たとえば `g`のような自己挿入文字 (self-inserting character) は、グローバルキーマップがそれをコマンド command `self-insert-command`にバインドするので、自己挿入を行なうのです。`C-a` のような標準的な Emacs の編集文字もグローバルキーマップから、それらの標準的な意味を取得します。`M-x keymap-global-set`のようなキーをリバインドするコマンドは、新しいバインディングをグローバルマップの適切な位置に保存することにより機能します (Section 33.3.5 [Rebinding], page 521 を参照)。カレントのキーバインディングを閲覧するには `C-h b`コマンドを使用してください。

ほとんどの現代的なキーボードは、文字キーと同じようにファンクションキーをもちます。ファンクションキーは文字キーが行なうように入力イベントを送り、キーマップはファンクションキーにたいするバインディングをもつことができます。キーシーケンスにはファンクションキーと文字をミックスすることもできます。たとえば、キーボードにファンクションキー `Home`がある場合、Emacs は `C-x Home`のようなキーシーケンスを認識できます。`S-down-mouse-1`のように、マウスイベントとキーボードイベントをミックスすることさえ可能です。

テキスト端末では、ファンクションキーをタイプすることにより、文字シーケンスがコンピューターに送られます。シーケンスの正確な詳細は、ファンクションキーと端末タイプに依存します (シーケンスが `ESC [`で始まることもしばしばあります)。Emacs が端末タイプを理解する場合、自動的にそのようなシーケンスを 1 つの入力イベントとして処理します。

後続に 1 文字 (ASCIIと非 ASCIIの大文字小文字) を伴う C-cから構成されるキーシーケンスはユーザー用に予約済みです。これらのキーシーケンスを Emacs 自身がバインドすることは決してなく、Emacs 拡張はこれらのキーシーケンスのバインディングを避ける必要があります。言い換えると、ユーザーは C-c aや C-c çのようなキーシーケンスをバインドでき、それは Emacs のバインディングによりバインディングがシャドーされることは決してないということに基づいています。

33.3.2 プレフィクスキーマップ

内部的には、Emacs は各キーマップの 1 つのイベントだけを記録します。複数イベントのキーシーケンスの解釈は、キーマップの連鎖を生じます。最初のイベントにたいして最初のキーマップが定義を与え、シーケンス内の 2 番目のイベントを探すのに他のキーマップが使用され... と連鎖していきます。したがって C-xや ESCなどのプレフィクスキーは独自のキーマップをもち、それらはプレフィクスの直後のイベントにたいする定義を保持します。

プレフィクスキーの定義は通常、それに続くイベントを探すのに使用するキーマップです。プレフィクスキーの定義として、関数定義がキーマップであるような Lisp シンボルを指定することもできます。効果は同じですが、そのプレフィクスキーが何のためなのか説明するためのコマンド名を提供します。たとえば、C-xのバインディングはシンボル Control-X-prefixで、このシンボルの関数定義は、C-xコマンドにたいするキーマップです。プレフィクスキーとしての C-c、C-x、C-h、ESCは、グローバルキーマップに定義されているので、これらのプレフィクスキーは常に利用できます。

通常のプレフィクスキー以外に、“架空のプレフィクスキー (fictitious prefix key)” もあり、これらはメニューバーを表します。メニューバーのキーバインディングについての特別な情報は、Section “Menu Bar” in *The Emacs Lisp Reference Manual* を参照してください。ポップアップメニューを呼び出すマウスボタンイベントもプレフィクスキーです。詳細については、Section “Menu Keymaps” in *The Emacs Lisp Reference Manual* を参照してください。

いくつかのキーマップは、名前のついた変数に格納されています:

- ctl-x-mapは、C-xの後の文字に使用されるマップにたいする変数名です。
- help-mapは。C-hの後の文字のためのマップです。
- esc-mapは、ESCの後の文字のためのマップです。したがって、すべてのメタ文字がこのマップで定義されています。
- ctl-x-4-mapは、C-x 4の後の文字のためのマップです。
- mode-specific-mapは、C-cの後の文字のためのマップです。
- project-prefix-mapはプロジェクト関連のコマンドに使用される、C-x pの後の文字のためのマップです (Section 25.2 [Projects], page 352 を参照)。

33.3.3 ローカルキーマップ

ここまではグローバルマップの詳細を説明してきました。メジャーモードは、ローカルキーマップ *(local keymaps)* で独自のキーバインディングを提供することにより、Emacs をカスタマイズします。たとえば C モードは、C 言語のためにカレント行をインデントするために、TABをオーバーライドします。マイナーモードもローカルキーマップをもつことができます。マイナーモードが効力をもつとき、マイナーモードのキーマップの定義は、メジャーモードのローカルキーマップとグローバルキーマップの両方をオーバーライドします。それに加えて、バッファーの一部のテキストに、他のすべてのキーマップをオーバーライドする独自のキーマップを指定できます。

ローカルキーマップは、あるキーをプレフィクスキーマップとして定義することにより、そのキーをプレフィクスキーとして再定義できます。そのキーがグローバルでもプレフィクスとして定義されている場合、そのキーのグローバルおよびローカルの定義 (両方のキーマップ) が、相乗して効果を

もちます。つまりプレフィクスキーに続くイベントを探すのに、両方の定義が使用されます。たとえばローカルキーマップが C-cをプレフィクスキーマップとして定義し、そのキーマップが C-zをコマンドとして定義する場合、これは C-c C-zにローカルな意味を提供します。これは C-cで始まる他のシーケンスには影響を与えません。これらのシーケンスが独自のローカルバインディングをもたない場合、グローバルバインディングが効果をもちます。

これを別の方法で考えると、Emacs はキーシーケンス全体のバインディングにたいして、複数のキーマップを 1 つずつ探して、複数イベントキーシーケンスを処理すると考えることができます。最初にマイナーモードが有効な場合はマイナーモードのキーマップをチェックして、次にメジャーモードのキーマップをチェックして、それからグローバルキーマップをチェックするのです。これはキーの照合が機能する正確な方法ではありませんが、通常の場面における結果を理解するには充分です。

33.3.4 ミニバッファーキーマップ

ミニバッファーは独自のローカルキーマップのセットをもちます。これにはさまざまな補完や exit コマンドが含まれます。

- `minibuffer-local-map`は、通常の入力 (補完なし) に使用されます。
- `minibuffer-local-ns-map`は同様ですが、SPCで RETと同じように exit します。
- `minibuffer-local-completion-map`は、寛大な補完 (permissive completion) のためのキーマップです。
- `minibuffer-local-must-match-map`は、強い補完 (strict completion) と慎重な補完 (cautious completion) のためのキーマップです。
- `minibuffer-local-filename-completion-map`は前の 2 つと同様ですが、特にファイル名補完のためのキーマップです。これは SPCをバインドしません。

`minibuffer-local-completion-map`ではデフォルトでは TAB、SPC、?は補完を行います。日常的にスペースやクエスチョンマークのような文字をもつコレクションにたいして補完を行うようなら、以下を init ファイルに記述してそれらのキーでの置換を無効にすると便利かもしれません:

```
(keymap-set minibuffer-local-completion-map "SPC" 'self-insert-command)
(keymap-set minibuffer-local-completion-map "?" 'self-insert-command)
```

33.3.5 対話的なキーバインディングの変更

Emacs がキーを再定義する方法は、キーマップのそのキーのエントリーを変更する方法です。グローバルキーマップを変更できます。この場合すべてのメジャーモードで変更が効果をもちます (ただし同じキーにたいしてそれをオーバーライドする独自のローカルバインディングをもつ場合を除きます)。ローカルキーマップを変更することもできます。これは同じメジャーモードを使用するすべてのバッファーに効果があります。

このセクションでは、現在の Emacs セッションでキーをリバインドする方法を説明します。将来の Emacs セッションで効果をもつようにキーをリバインドする方法については、Section 33.3.6 [Init Rebinding], page 522 を参照してください。

M-x keymap-global-set RET *key cmd* RET
: *cmd*を実行する *key*をグローバルに定義します。

M-x keymap-local-set RET *key cmd* RET
: *cmd*を実行する *key*を、(そのとき効力をもつメジャーモードで) ローカルに定義します。

M-x keymap-global-unset RET *key*
: グローバルマップで *key*を未定義にします。

`M-x keymap-local-unset RET` *key*
(そのとき効力をもつメジャーモードで) ローカルに *key*を未定義にします。

たとえば以下は、通常の `C-z`にたいするグローバルな定義を置き換えて、`C-z`を `shell`コマンド (Section 31.5.2 [Interactive Shell], page 459 を参照してください) にバインドします:

```
M-x keymap-global-set RET C-z shell RET
```

`keymap-global-set`コマンドは、キーの後にコマンド名を読み取ります。キーを押した後に以下のようなメッセージが表示されるので、そのキーにバインドしたいコマンドを入力できます:

```
Set key C-z to command:
```

ファンクションキーとマウスイベントも同じ方法で再定義できます。リバインドするキーを指定するときに、ファンクションキーをタイプするか、マウスをクリックするだけです。

複数のイベントを含むキーも、同じ方法で再定義できます。Emacs は、(プレフィクスキーではない) 完了キーまで、リバインドするキーの読み取りを続けます。したがって *key*に `C-f`をタイプすると、それで完了です。これによりミニバッファーに入って、すぐに *cmd*を読み取ります。しかし `C-x`をタイプした場合、これはプレフィクスなので、他の文字を読み取ります。それが `4`の場合、これもプレフィクス文字なので、さらに文字を読み取ります。たとえば、

```
M-x keymap-global-set RET C-x 4 $ spell-other-window RET
```

これは、(架空のコマンド)`spell-other-window`を実行するように、`C-x 4 $`を再定義します。

`keymap-global-unset`で、キーのグローバルな定義を削除できます。これはそのキーを未定義 *(undefined)* にします。その後このキーをタイプしても、Emacs はビープ音を鳴らすだけです。同様に `keymap-local-unset`は、カレントメジャーモードのキーマップでキーを未定義にして、メジャーモードでそのキーにたいするグローバル定義 (またはグローバル定義に無い状態) が有効になります。

あるキーを再定義 (または未定義に) してから、後でその変更を取り消したくなった場合、キーを未定義にしても上手くいきません — そのキーを標準の定義に再定義する必要があります。そのキーの標準の定義の名前を見つけるには、フレッシュな Emacs の Fundamental モードで、`C-h c`を使用します。このマニュアルのキーのドキュメントにも、それらのコマンド名がリストされています。

間違ってコマンドを呼び出すことから自分を守りたい場合、そのキーを未定義にするより、コマンドを無効にするほうがよいでしょう。無効にされたコマンドは、実際にそれを実行したくなったとき、少しの手間で呼び出すことができます。Section 33.3.11 [Disabling], page 527 を参照してください。

33.3.6 `init` ファイル内でのキーのリバインド

いつでも使いたいキーバインドがある場合、初期化ファイルに Lisp コードを記述することにより、それらを指定できます。初期化ファイルの説明については、Section 33.4 [Init File], page 528 を参照してください。

Lips を用いてキーバインディングを記述する際には、`keymap-global-set`または `keymap-set`のいずれかの関数を使用することをお薦めします。たとえば以下はグローバルキーマップにおいて、`C-z`に `shell`コマンド (Section 31.5.2 [Interactive Shell], page 459 を参照) をバインドする例です:

```
(keymap-global-set "C-z" 'shell)
```

`keymap-global-set`の 1 つ目の引数にはキーシーケンスを記述します。これはスペースで区切られた一連の文字から構成される文字列であり、それぞれの文字がキーに対応しています。修飾されたキーは、たとえば Control キーなら '`C-`'、Meta キーなら '`M-`'のように、修飾キーを前に追加することによって指定できます。`TAB`や `RET`のようなスペシャルキーの場合には、`TAB`や `RET`のように山形カッコ (angle brackets) の中に記述することによって指定することができます。

上記例でキーシーケンスにバインドされるコマンド名shellの前にあるシングルクォートによって、それを変数ではなくシンボル定数としてマークします。クォートを省略した場合、Emacs は shellを変数として評価しようとします。これはおそらくエラーを引き起こしますが、もちろんあなたはそれを望まないはずです。

以下に、ファンクションキーやマウスイベントなどを含めた、追加の例を示します:

```
(keymap-global-set "C-c y" 'clipboard-yank)
(keymap-global-set "C-M-q" 'query-replace)
(keymap-global-set "<f5>" 'flyspell-mode)
(keymap-global-set "C-<f5>" 'display-line-numbers-mode)
(keymap-global-set "C-<right>" 'forward-sentence)
(keymap-global-set "<mouse-2>" 'mouse-save-then-kill)
```

非ASCII文字にたいするキーバインディングは、言語とコーディングシステムに問題を起こすかもしれません。 Section 33.4.5 [Init Non-ASCII], page 534 を参照してください。

かわりに低レベル関数define-keyとglobal-set-keyを使うこともできます。たとえばこれらの低レベル関数を用いて、上記例のようにC-zにshellコマンドをバインドするには:

```
(global-set-key (kbd "C-z") 'shell)
```

キーシーケンスの指定にはさまざまな方法が存在しますが、上記例で示した関数kbdを使うのがもっともシンプルな方法です。kbdは単一の文字列引数としてキーシーケンスのテキスト表現を受け取り、global-set-keyのような低レベル関数に適した形式に変換します。Lips におけるキーのバインディングと使用に関する詳細については、Section “Keymaps” in *The Emacs Lisp Reference Manual* を参照してください。

Section 33.3.3 [Local Keymaps], page 520 で説明したように、メジャーモードとマイナーモードはローカルキーマップを定義できます。これらのキーマップは、セッションで最初にそのモードがロードされるときに構築されます。特定のキーマップに変更を施すには、関数keymap-setを使用できます。キーバインディングの削除にはkeymap-unsetを使用します。

モードのキーマップはモードがロードされるまで構築されないので、キーマップを変更するコードをモードフック *(mode hook)* に配置することにより実行を遅延しなければなりません (Section 33.2.2 [Hooks], page 509 を参照)。たとえば Texinfo モードは、フック texinfo-mode-hookを実行します。以下は Texinfo モードで C-c nと C-c pにローカルバインディングを追加、C-c C-x xのローカルバインディングを削除するために、どのようにフックを使用できるかの例です:

```
(add-hook 'texinfo-mode-hook
          (lambda ()
            (keymap-set texinfo-mode-map "C-c p"
                        'backward-paragraph)
            (keymap-set texinfo-mode-map "C-c n"
                        'forward-paragraph)))
            (keymap-set texinfo-mode-map "C-c C-x x" nil)
```

33.3.7 修飾キー

Emacs では、デフォルトのキーバインディングがセットアップされているので、修飾されたアルファベット文字は大文字小文字が区別されません。つまり C-Aは C-aと同じことを行い、M-Aは M-aと同じことを行ないます。これはアルファベット文字だけに当てはまり、他のキーのシフトキーが押された (shifted) バージョンには適用されません。たとえば、C-@は C-2と同じではありません。

Control修飾されたアルファベット文字は、一般的に大文字小文字が区別されません。Emacs は常に C-Aを C-a、C-Bを C-b、... として扱います。これは歴史的な理由によります。非グラフィカルな

環境ではこれらのキーストロークに差異はありません。しかし GUI フレームでは shift した Control のアルファベットキーストロークをバインドできます。

```
(keymap-global-set "C-S-n" #'previous-line)
```

他の修飾キーでは Emacs をカスタマイズするとき修飾されたアルファベットの大文字小文字を区別するようにできます (非グラフィカルなフレームでも可)。たとえば M-aと M-Aで別のコマンドを実行できます。

一般的に使用される修飾キーは Controlと Metaだけですが、Emacs は他の修飾キーもサポートします。これらは Super、Hyper、Altと呼ばれます。これらの修飾キーを使用する方法を提供する端末の数は多くありません。ほとんどのキーボードで Altとラベルされたキーは、通常は Altではなく Meta修飾を発行します。Emacs の標準のキーバインディングには Superと Hyperは含まれず、少数の標準キーバインディングだけが Altを使用します。これらのキーで修飾された文字は含まれません。しかしこれらの修飾を使用するキーバインディングに意味を割り当てるように Emacs をカスタマイズできます。修飾ビットはそれぞれ‘s-’、‘H-’、‘A-’になります。

これらの追加的な修飾キーがキーボードになくても、C-x @を使用して入力できます。C-x @ h は Hyper フラグ、C-x @ sは Super フラグ、C-x @ aは Alt フラグを次の文字に加えます。たとえば Hyper-Control-aを入力するには、C-x @ h C-aとタイプします (残念なことに同じ文字にたいして C-x @を使用して、2 つの修飾を追加する方法はありません。なぜなら最初の 1 つは 2 回目の C-xにたいして作用するからです)。同様に必要となることは稀ですが C-x S、C-x c、C-x mで Shift、Control、Meta といった修飾キーを入力できます。

33.3.8 ファンクションキーのリバインド

キーシーケンスには、通常の文字と同じようにファンクションキーを含めることができます。Lisp 文字 (実際は整数です) がキーボードの文字を表すように、Lisp シンボルはファンクションキーを表します。ファンクションキーのラベルに示された単語が、それにタイプする Lisp シンボルの名前になります。以下は一般的なファンクションキーにたいする、慣例的な Lisp 名です:

left、up、right、down
: カーソル矢印キーです。

begin、end、home、next、prior
: その他のカーソルを再配置するキーです。

select、print、execute、backtab
insert、undo、redo、clearline
insertline、deleteline、insertchar、deletechar
: その他のファンクションキーです。

f1、f2、...、f35
: (キーボード上部にある) 番号つきのファンクションキーです。

kp-add、kp-subtract、kp-multiply、kp-divide
kp-backtab、kp-space、kp-tab、kp-enter
kp-separator、kp-decimal、kp-equal
kp-prior、kp-next、kp-end、kp-home
kp-left、kp-up、kp-right、kp-down
kp-insert 、kp-delete
: (標準的なキーボードでは右側にある) キーパッドの名前や句読点のキーです。

kp-0、kp-1、...、kp-9
: キーパッドの数字キーです。

`kp-f1`、`kp-f2`、`kp-f3`、`kp-f4`
: キーパッドの PF キーです。

これらの名前は便利ですが、いくつかのシステム (特に X を使用するシステム) では、異なる名前を使用するかもしれません。端末のファンクションキーにたいして、どのシンボルが使用されているか確認するには、`C-h c`とタイプして、その後にそのファンクションキーを入力してください。

ファンクションキーにバインドする例については、Section 33.3.6 [Init Rebinding], page 522 を参照してください。

多くのキーボードの右手側には、テンキーボード (numeric keypad) があります。キーパッドのテンキーは‘`Num Lock`’とラベルされたキーで切り替えるにことにより、カーソル移動キーにもなります。デフォルトでは、Emacs はこれらのキーを、メインのキーボードの対応するキーに変換します。たとえば‘`Num Lock`’がオンの場合、テンキーの labeled ‘8’のラベルがついたキーは`kp-8`を生成し、これは`8`に変換されます。また‘`Num Lock`’がオフの場合、このキーは`kp-up`を生成し、これは`UP`に変換されます。`8`や`UP`のようなキーをリバインドした場合、それはキーパッドの対応するキーにも影響します。しかし直接‘`kp-`’をリバインドした場合、これはメインのキーボードの等価なキーに影響を与えません。修飾されたキーは変換されないことに注意してください。たとえば`Meta`キーを押したまま、テンキーの ‘8’を押すと、これは`M-kp-8`を生成します。

Emacs は変数 `keypad-setup`、`keypad-numlock-setup`、`keypad-shifted-setup`、`keypad-numlock-shifted-setup`を使用することにより、テンキーのキーをバインドするための便利な方法を提供します。これらの変数は‘`keyboard`’カスタマイズグループで見つけることができます (Section 33.1 [Easy Customization], page 499 を参照してください)。キーをリバインドして、数引数を発行するなど、他のタスクを行なうことができます。

33.3.9 名前のある **ASCII**コントロール文字

当初`TAB`、`RET`、`BS`、`LFD`、`ESC`、`DEL`は、特定の ASCIIコントロール文字の名前として使用され、多用されるために自身の特別なキーをもつようになりました。たとえば`TAB`は`C-i`の別の名前です。その後、ユーザーは Emacs でこれらのキーと、`Ctrl`キーと一緒にタイプするコントロール文字を区別できると便利なことに気づきました。したがってほとんどの現代的な端末では、これらは同じではありません。つまり`TAB`は`C-i`と異なります。

これら 2 種類の入力を、キーボードが区別するなら、Emacs も区別することができます。Emacs は特別なキーを`tab`、`return`、`backspace`、`linefeed`、`escape`、`delete`という名前のファンクションキーとして扱います。これらのファンクションキーは、そのキー自体に何もバインドされていない場合は、対応する ASCII文字に自動的に変換されます。結果として、ユーザーも Lisp プログラマーも、彼らがそうしたいと望まない限りは、これらの区別に注意を払う必要はありません。

(たとえば)`TAB`と`C-i`を区別したくない場合は、ASCII文字の`TAB`(8 進コード 011) だけにたいしてバインディングを 1 つ指定します。これらを区別したいときは、ASCII文字にたいして 1 つのバインディング、ファンクションキーの`tab`にたいして別のバインディングを指定します。

通常の ASCII端末では、`TAB`と`C-i`(および同じような他のペアー) を区別する方法はありません。なぜなら端末はどちらの場合も同じ文字を送るからです。

33.3.10 マウスボタンのリバインド

Emacs はマウスボタンを表すためにも Lisp シンボルを使用します。Emacs で通常のマウスイベントは、クリック *(click)* イベントです。これはボタンを押して、マウスを移動せずにボタンを離すと発生します。ドラッグ *(drag)* イベントも取得できます。これはボタンを押したままマウスを移動したとき発生します。ドラッグイベントは、最後にボタンを離したときにも発生します。

基本的なクリックイベントにたいするシンボルは、一番左のボタンが mouse-1、次が mouse-2、. . . となります。以下は、カレントウィンドウを 2 番目のマウスボタンで分割するように再定義する方法です:

```
(keymap-global-set "<mouse-2>" 'split-window-below)
```

ドラッグイベントにたいするシンボルも同様ですが、単語 'mouse'の前にプレフィクス 'drag-'がつきます。たとえば左ボタンでのドラッグは drag-mouse-1イベントを生成します。

マウスボタンが押されたときに発生するイベントにたいして、バインディングを定義することもできます。これらのイベントは 'drag-'ではなく 'down-'で始まります。このようなイベントは、それらにキーがバインドされているときだけ生成されます。ボタンダウンイベントを受け取った場合、その後に常にそれに対応するクリックまたはドラッグイベントが続きます。

もし望むならシングルクリック、ダブルクリック、トリプルクリックを区別することもできます。ダブルクリックとは、マウスボタンをほぼ同じ場所で 2 回クリックすることを意味します。最初のクリックは通常のクリックイベントを生成します。2 回目のクリックが充分早ければ、かわりにダブルクリックイベントを生成します。ダブルクリックイベントにたいするイベントタイプは、たとえば double-mouse-3のように、'double-'で始まります。

これは同じ場所での 2 回目のクリックに特別な意味を与えることができることを意味しますが、それは最初のクリックを受け取ったときに実行される、通常のシングルクリックにたいする定義も実行されることを前提にしなければなりません。

これはダブルクリックで行なえることを制限しますが、ユーザーインターフェースデザイナーはこの制限は任意のケースにおいて従うべき制限だと言います。ダブルクリックは、シングルクリックで行なう何かを、よりもっと行なうためのものであるべきです。ダブルクリックイベントにたいするコマンドは、ダブルクリックにたいして追加の作業を処理するべきです。

ダブルクリックイベントにバインディングがない場合、これは対応するシングルクリックイベントに変化します。したがって、特にダブルクリックイベントを定義していない場合、これはシングルクリックコマンドを 2 回実行します。

Emacs はトリプルクリックイベントもサポートし、それらの名前は 'triple-'で始まります。Emacs はクワドループルクリック (quadruple clicks: 4 連クリック) をイベントタイプとして区別しません。3 回目以降のクリックは、追加のトリプルクリックイベントを生成します。しかしクリックされた数はすべてイベントリストに記録されるので、Emacs Lisp を知っていて、本当にそれを使いたい場合はそれらを区別できます (Section "Click Events" in *The Emacs Lisp Reference Manual* を参照してください)。わたしたちは 3 連クリックを超えるクリックに明確な意味を与えるのは推奨しませんが、連続するクリックが同じ 3 つの意味のセットを巡回する— たとえば 4 連クリックは 1 クリックに等しく、5 連クリックは 2 連クリックに等しく、6 連クリックは 3 連クリックに等しい、とするのが便利なときがあるかもしれません。

Emacs はドラッグおよびボタンダウンイベントで、複数回ボタンが押されたことも記録します。たとえば、ボタンを 2 回押して、それからボタンを押したままマウスを移動した場合、Emacs は 'double-drag-'イベントを受け取ります。2 回目にボタンを押した瞬間、Emacs は 'double-down-'イベントを受け取ります (そしてすべてのボタンダウンイベントと同様に、なにもバインドされていなければ無視されます)。

変数 double-click-timeは、複数回のクリックをグループ化するのに、クリックの間にどれだけの時間経過を許すかを指定します。変数の値の単位はミリ秒です。値が nilの場合、ダブルクリックは検知されません。値が tの場合、時間の制限はありません。デフォルトは 500 です。

変数 double-click-fuzzは、複数回のクリックをグループ化するのに、クリックの間にどれだけマウスが移動できるかを指定します。変数の値はウィンドウ化されたディスプレイではピクセル単位で、テキストモード端末では文字セルの 1/8 を単位とし、デフォルトは 3 です。

マウスイベントにたいするシンボルは、修飾キーの状態も示し、‘C-’、‘M-’、‘H-’、‘s-’、‘A-’、‘S-’のプレフィクスが通常つきます。‘double-’や‘triple-’は常に‘drag-’や‘down-’の前にきますが、これらのプレフィクスは常にそれより前にきます。

フレームにはバッファーのテキストを表示しない、モードラインやスクロールバーのような領域が含まれます。スクリーンの特別な領域でマウスボタンが押されたかどうかは、ダミーのプレフィクスキーで知ることができます。たとえばモードラインでマウスをクリックした場合、通常のマウスボタンシンボルの前にプレフィクスキー mode-lineを受け取ります。したがって、以下はモードラインで左ボタンをクリックしたときに scroll-up-commandを実行する方法です:

```
(keymap-global-set "<mode-line> <mouse-1>" 'scroll-up-command)
```

以下はダミーのプレフィクスキーと、その意味の完全なリストです:

mode-line
: マウスはウィンドウのモードラインにあります。

vertical-line
: マウスは横に並んだウィンドウを分ける垂直ラインにあります (スクロールバーを使用している場合は、垂直ラインに表示されます)。

vertical-scroll-bar
: マウスは垂直スクロールバーにあります (これは Emacs が現在サポートしているスクロールバーにたいしてだけです)。

menu-bar
: マウスはメニューバーにあります。

tab-bar
: マウスはタブバーにあります。

tab-line
: マウスはタブラインにあります。

header-line
: マウスはヘッダーラインにあります。

キーシーケンスにマウスボタンを複数配することもできますが、これは通常行なわれません。

33.3.11 コマンドの無効化

コマンドを無効にするとは、そのコマンドを対話的に呼び出しユーザーに確認を求めることを意味します。コマンドを無効にする目的は、ユーザーが間違ってコマンドを実行するのを防ぐためです。わたしたちは初心者を混乱させるようなコマンドにたいして、これを行なっています。

無効なコマンドを呼び出そうとすると、Emacs はコマンド名、コマンドのドキュメント、すぐに何を行なうかの手引きを対話的に表示します。その後、Emacs はコマンドを要求されたとおり実行するか、そのコマンドを有効にしてから実行するか、キャンセルするか入力を求めます。コマンドを有効にすると決めた場合は、他の質問— 永続的に有効にするか、それともカレントセッションでだけ有効にするか — にも応えなければなりません (永続的に有効にする場合、これは自動的に初期化ファイルを編集することにより機能します)。!とタイプして、カレントセッションだけにたいして、すべてのコマンドを有効にすることもできます。

コマンド無効化の直接的なメカニズムは、コマンドにたいする Lisp シンボルの disabledプロパティに非 nilを put することです。以下はこれを行なう Lisp プログラムです:

```
(put 'delete-region 'disabled t)
```

disabledプロパティの値が文字列の場合、その文字列はコマンドが使用されたときに表示されるメッセージに含まれます。

```
(put 'delete-region 'disabled
```

```
        "It's better to use `kill-region' instead.\n")
```

無下にコマンドを無効にするのではなく、コマンドを実行する前に問い合わせを行いたい場合もあるかもしれません。たとえば `M->` (`end-of-buffer`) というコマンドを実行する前に問い合わせを行うには、init ファイルに以下のように記述してください:

```
(command-query
 'end-of-buffer
 "Do you really want to go to the end of the buffer?")
```

デフォルトでは `y/n`による問い合わせを行いますが、3 つ目のオプション引数に非 `nil`を与えれば、かわりに `yes`による問い合わせを行います。

初期化ファイルを直接編集するか、初期化ファイルを編集する `M-x disable-command`コマンドにより、コマンドを無効にできます。同様に `M-x enable-command`はコマンドを永続的に有効にするために、初期化ファイルを編集します。Section 33.4 [Init File], page 528 を参照してください。

Emacs が`-q`または`--no-init-file`オプション (Section C.2 [Initial Options], page 576 を参照してください) で呼び出された場合、これらのコマンドは初期化ファイルを編集しません。Emacs は初期化ファイルを読み込んでいないので、これを行なうと情報が失われるかもしれないからです。

コマンドが無効にされているかどうかは、それを呼び出すのに使用されるキーとは独立しています。`M-x`を使用してコマンドを呼び出しても、無効化は適用されます。しかし Lisp プログラムから関数として呼び出す場合、コマンドの無効化は効力をもちません。

33.4 Emacs 初期化ファイル

Emacs を開始したとき、Emacs は通常、初期化ファイル *(initialization file)*、短くは *init* ファイルから、Lisp プログラムのロードを試みます。このファイルは、もし存在する場合は、Emacs をどのように初期化するかを指定します。Emacs が`~/.emacs.el`、`~/.emacs.d/init.el`、`~/.config/emacs/init.el`、またはその他の場所を探すにしても、伝統的に`~/.emacs`は init ファイルとして扱われます。Section 33.4.4 [Find Init], page 533 を参照してください。

Emacs のすべての設定が 1 つのディレクトリーにあると便利だと思うかもしれません。その場合には`~/.emacs.d/init.el`、または XDG 互換の`~/.config/emacs/init.el`を使用する必要があります。

コマンドラインスイッチ '`-q`'により、init ファイルのロードを抑止でき、'`-u`' (または '`--user`') で、別のユーザーの init ファイルを指定できます (Section C.2 [Initial Options], page 576 を参照してください)。

デフォルト *init* ファイルが存在する場合もあります。これは `default.el`という名前のライブラリーで、ライブラリーにたいする標準の検索パスから探されます。Emacs ディストリビューションには、そのようなライブラリーは含まれていませんが、あなたのサイトは、ローカルなカスタマイズのためにこれを作成しているかもしれません。このライブラリーが存在する場合、Emacs を開始したときは常にこれがロードされます (ただし '`-q`'を指定した場合は除きます)。しかし init ファイルがあれば、それが最初にロードされるので、そこで `inhibit-default-init`に非 `nil`をセットすれば、デフォルト `init` ファイルはロードされません。

あなたのサイトにはサイトスタートアップファイル *(site startup file)* もあるかもしれません。もし存在する場合、これは `site-start.el`という名前です。`default.el`と同様に、Emacs はこのファイルを、Lisp ライブラリーにたいする標準の検索パスから探します。Emacs はこのライブラリーを init ファイルの前にロードします。このライブラリーのロードを抑止するには、オプション '`--no-site-file`'を使用します。Section C.2 [Initial Options], page 576 を参照してください。わたしたちは何かを変更する場合、`site-start.el`の使用を推奨しません。これを好まないユーザー

もいるからです。変更を `default.el`に記述すれば、ユーザーはもっと簡単にそれをオーバーライドできます。

`default.el`や `site-start.el`は、Emacs が Lisp ライブラリーを検索する任意のディレクトリーに配置できます。変数 `load-path` (Section 24.8 [Lisp Libraries], page 327 を参照してください) は、これらのディレクトリーを指定します。多くのサイトはこれらのファイルを、Emacs のインストールディレクトリーの中の、`site-lisp`(たとえば`/usr/local/share/emacs/site-lisp`)に配置します。

init ファイルにたいするバイトコンパイル (Section “Byte Compilation” in *the Emacs Lisp Reference Manual* を参照してください) は推奨されていません。一般的にこれは開始時のスピードの大幅な改善はせず、ファイルをリコンパイルするのを忘れたときに問題を起こすことが多いのです。よりよい解決策は、Emacs サーバー (Section 31.6 [Emacs Server], page 469 を参照してください) を使用して、Emacs を開始する回数を減らすことです。init ファイルで多くの関数を定義している場合、これらを (バイトコンパイルされた) 別のファイルに移動して、それを init ファイルでロードします。

マイナーなカスタマイズを超えるような、実際の Emacs Lisp プログラムを記述するつもりなら、*Emacs Lisp Reference Manual* を読むべきでしょう。

33.4.1 **init** ファイルの構文

init ファイルには、1 つ以上の Lisp 式が含まれています。式のそれぞれは引数をともなう関数名で、それらはすべてカッコで括られています。たとえば `(setq fill-column 60)`は、変数 `fill-column`(Section 22.6 [Filling], page 256 を参照してください) を 60 にセットするために、関数 `setq`を呼び出します。

`setq`で任意の Lisp 変数をセットできますが、init ファイルの特定の変数にたいして、`setq`は多分あなたの望むとおりには動作しないでしょう。いくつかの変数は `setq`でセットしたとき、自動的にバッファーローカルになります。あなたが望むのは、init ファイルでデフォルト値をセットすることなので、`setq-default`を使用します (以下のセクションには両方の方法の例あり)。

マイナーモードのカスタマイズ可能な変数のいくつかは Customize でセットするとモードを有効にするために特別なことを行ないますが、通常の `setq`ではそれを行ないません。init ファイルでモードを有効にするには、マイナーモードコマンドを呼び出します。最後に複雑な方法で初期化される少数のカスタマイズ可能なユーザーオプションは Customize インターフェイス (Chapter 33 [Customization], page 499 を参照)、あるいは `customize-set-variable`か `setopt` (Section 33.2.1 [Examining], page 508 を参照) を使用してセットする必要があります。

`setq`の 2 番目の引数は、変数にたいする新しい値の式です。これには、定数、変数、関数呼び出し式を指定できます。init ファイルでは、定数が使用される場合がほとんどです。これは以下のとおりです:

数字: 数字は 10 進で記述され、オプションで最初にマイナス記号がある場合があります。

文字列: Lisp の文字列構文は、少数の例外を除き、C の文字列構文と同じです。文字列定数の開始と終了にはダブルクォートを使用します。

文字列には改行を含む、特別なリテラル文字を含めることができます。しかし、それらにたいして、バックスラッシュシーケンスを使う方が明確になる場合が多くあります。改行は ‘`\n`’、バックスペースは ‘`\b`’、キャリッジリターンは ‘`\r`’、タブは ‘`\t`’、フォームフィード (control-L) は ‘`\f`’、エスケープは ‘`\e`’、バックスラッシュは ‘`\\`’、ダブルクォートは ‘`\"`’、そして 8 進コードが *ooo*の文字は ‘`\ooo`’です。バックスラッシュとダブルクォートだけは、バックスラッシュシーケンスが必須な文字です。

‘\C-’はコントロール文字のプレフィクスとして使用でき、‘\C-s’は ASCIIの control-S です。‘\M-’はメタ文字のプレフィクスとして使用でき、‘\M-a’は Meta-Aで、‘\M-\C-a’は Ctrl-Meta-Aです。

init ファイルに非 ASCII文字を含めるための情報は、Section 33.4.5 [Init Non-ASCII], page 534 を参照してください。

文字: Lisp の文字定数の構文は、たとえば?x、?\n、?\"、?\)のように、文字 ‘?’と、その後ろに文字または ‘\’で始まるエスケープシーケンスからなります。Lisp では、文字列と文字は置き換え可能ではないことに注意してください。あるコンテキストでは一方が、他のコンテキストでは他方が要求されます。

非 ASCII文字を送るキーにコマンドをバインドする情報については、Section 33.4.5 [Init Non-ASCII], page 534 を参照してください。

True: tは “true(真)” という意味です。

False: nilは “false(偽)” という意味です。

その他の Lisp オブジェクト:
シングルクォートに続けて Lisp オブジェクトを記述します。

Emacs Lisp 構文についての詳細な情報は Section “Introduction” in *The Emacs Lisp Reference Manual* を参照してください。

33.4.2 init ファイルの例

以下は Lisp 式で一般的に行ないたいような事柄の例です:

- 変数 load-pathにディレクトリーを追加します。その後、Emacs に含まれていない Lisp ライブラリーをそのディレクトリーに配置すれば、M-x load-libraryでそれらをロードすることができます。Section 24.8 [Lisp Libraries], page 327 を参照してください。

```
(add-to-list 'load-path "/path/to/lisp/libraries")
```

- C モードで、ポイントが行の途中にある場合、TABが単にタブを挿入するようにします。

```
(setq c-tab-always-indent nil)
```

個の例では、変数の通常の値が t、つまり “true” の変数の値を、nilつまり “false” にしています。

- (この設定をオーバーライドしないすべてのバッファーで) 検索のデフォルトを、大文字小文字を区別させるようにします。

```
(setq-default case-fold-search nil)
```

これはデフォルト値をセットし、この変数にたいするローカル値 (Section 33.2.3 [Locals], page 511 を参照してください) をもたないすべてのバッファーに効果を及ぼします。case-fold-searchを setqでセットした場合は、カレントバッファーだけに効果があり、それは多分あなたが init ファイルで行ないたいことではないはずです。

- Emacs があなたのメールアドレスを正しく推測できない場合、メールアドレスを指定します。

```
(setq user-mail-address "cheney@torture.gov")
```

Message モードのような、さまざまな Emacs パッケージは、メールアドレスを知る必要がある場合に、user-mail-addressを参照します。Section 29.2 [Mail Headers], page 421 を参照してください。

- 新しいバッファーにたいするデフォルトのモードを Text モードにします。

```
(setq-default major-mode 'text-mode)
```

  Text モードに入るコマンドに text-modeが使用されていることに注意してください。前のシングルクォートはシンボルを定数にしています。そうしない場合、text-modeは変数名として扱われます。
- 西ヨーロッパのほとんどの言語をサポートする Latin-1 文字を、デフォルトにセットアップします。

```
(set-language-environment "Latin-1")
```

- グローバルなマイナーモードの Line Number モードをオフに切り替えます。

```
(line-number-mode 0)
```

- Text モードと関連するモードで、自動的に Auto Fill モードをオンに切り替えます (Section 33.2.2 [Hooks], page 509 を参照してください)。

```
(add-hook 'text-mode-hook 'auto-fill-mode)
```

- クリップボード使用時のコーディングシステムを変更します (Section 19.10 [Communication Coding], page 229 を参照)。

```
(setopt selection-coding-system 'utf-8)
```

- インストールされた fooという名前のライブラリー (実際には標準の Emacs ディレクトリーの foo.elcまたは foo.elというファイル) をロードします。

```
(load "foo")
```

  loadの引数が、‘/’や‘~’で始まらない相対ファイル名の場合、loadは load-pathのディレクトリーを検索します (Section 24.8 [Lisp Libraries], page 327 を参照してください)。
- ホームディレクトリーの、コンパイルされた Lisp ファイル foo.elcをロードします。

```
(load "~/foo.elc")
```

  ここでは完全なファイル名が使用されているので、検索は行なわれません。
- mypackageという名前の Lisp ライブラリー (たとえば mypackage.elcや mypackage.elというファイル) をロードすることにより、関数 myfunctionの定義を探すよう Emacs に指示します。

```
(autoload 'myfunction "mypackage" "Do what I say." t)
```

  ここで文字列"Do what I say."は、この関数のドキュメント文字列です。これを autoload定義の中で指定することにより、そのパッケージがロードされていなくてもヘルプコマンドで利用可能になります。最後の引数 tは、この関数が interactive(対話的) であることを示します。つまり、この関数は M-x myfunction RETとタイプするか、キーにバインドすることにより、対話的に呼び出すことができます。関数が interactive でない場合は、tを省略するか、nilを使用します。
- 関数 make-symbolic-linkを実行するように、キー C-x lをリバインドします (Section 33.3.6 [Init Rebinding], page 522 を参照してください)。

```
(keymap-global-set "C-x l" 'make-symbolic-link)
```

  または

```
(keymap-set global-map "C-x l" 'make-symbolic-link)
```

  繰り返しになりますが、シングルクォートは make-symbolic-linkを変数として値を参照するのではなく、シンボルとして参照するために使用されることに注意してください。

- Lisp モードだけで、同じことを行ないます。

```
(keymap-set lisp-mode-map "C-x l" 'make-symbolic-link)
```

- Fundamental モードで next-lineを実行するすべてのキーにたいして、かわりにforward-lineを実行するように再定義します。

```
(keymap-substitute global-map 'next-line 'forward-line)
```

- C-x C-vを未定義にします。

```
(keymap-global-unset "C-x C-v")
```

  キーを未定義にする 1 つの理由は、それをプレフィクスにできることです。単に C-x C-v *anything*を定義すると、C-x C-vはプレフィクスになりますが、最初に通常の C-x C-vにたいする非プレフィクス定義を開放しなければなりません。

- Text モードで '$'に句読点構文をもたせます。'$'にたいする文字定数の使い方に注意してください。

```
(modify-syntax-entry ?\$ "." text-mode-syntax-table)
```

- 確認なしでの、コマンド narrow-to-regionの使用を有効にします。

```
(put 'narrow-to-region 'disabled nil)
```

- さまざまなプラットフォームと Emacs バージョンにたいして、設定を調整します。

  ユーザーは通常、すべてのシステムで Emacs が同じように振る舞うことを期待するので、すべてのプラットフォームで同じ init ファイルを使用するのが妥当です。しかし Emacs をカスタマイズするのに使用する関数が、他のプラットフォームや Emacs バージョンで利用できないということも発生します。この状況に対処するには、以下のように、ある関数または機能が利用可能かテストする条件文の内部に、カスタマイズを配置します:

```
(if (fboundp 'blink-cursor-mode)
    (blink-cursor-mode 0))

(if (boundp 'coding-category-utf-8)
    (set-coding-priority '(coding-category-utf-8)))
```

  関数が定義されていない場合に発生するエラーを、単に無視することもできます。

```
(ignore-errors (set-face-background 'region "grey75"))
```

  存在しない変数への setqは一般的に無害なので、これらを条件文の中に置く必要はありません。

- 自動的にパッケージのロードと構成を行うためには use-packageを使用します。

```
(use-package hi-lock
  :defer t
  :init (add-hook 'some-hook 'hi-lock-mode)
  :config (use-package my-hi-lock)
  :bind (("M-o l" . highlight-lines-matching-regexp)
         ("M-o r" . highlight-regexp)
         ("M-o w" . highlight-phrase)))
```

  これは初めて hi-lockにコマンドか変数が使用された際に hi-lockをロードするとともに、そのロード後に追加で my-hi-lockパッケージ (hi-lockを更にカスタマイズしたパッケージかもしれない) をロードします。use-packageの機能については、それ自身のマニュアルにおいて完全に文書化されています。*use-package User manual* を参照してください。

33.4.3 端末固有の初期化

各端末タイプは、Emacs をその端末タイプで実行するとき Emacs にロードされる Lisp ライブラリーをもつことができます。*termtype*という名前の端末タイプにたいして、そのライブラリーは `term/`*termtype*と呼ばれます (`term-file-aliases`の連想配列 (association list) の中に (*termtype* `.` *alias*)という形式のエントリーがある場合、Emacs は *termtype*のところに *alias*を使用します)。このライブラリーは通常のようにディレクトリー `load-path`を検索することにより見つけられ、サフィックスは‘`.elc`’と‘`.el`’です。通常はほとんどの Emacs ライブラリーがあるディレクトリーの、サブディレクトリー `term`にあります。

端末固有ライブラリーの通常の目的は、`input-decode-map`を使用して、その端末のファンクションキーで使用されるエスケープシーケンスを、より意味のある名前にマップすることです。これがどのように行なわれるかの例は、ファイル `term/lk201.el`を参照してください。多くのファンクションキーは、Termcap データベースの情報にしたがい、自動的にマップされます。端末固有ライブラリーは、Termcap が指定しないファンクションキーだけをマップすればよいのです。

端末タイプがハイフンを含む場合、最初のハイフンの前の部分だけが、ライブラリーの選択で意味をもちます。したがって端末タイプ‘`aaa-48`’と‘`aaa-30-rv`’は、両方ともライブラリー `term/aaa`を使用します。ライブラリー内のコードは、`(getenv "TERM")`を使用して、完全なタイプ名を取得できます。

ライブラリーの名前は、変数 `term-file-prefix`の値と、端末タイプを結合することにより構築されます。`.emacs`で `term-file-prefix`を `nil`にセットすることにより、端末固有ライブラリーのロードを抑止できます。

Emacs は初期化の最後、`.emacs`と端末固有ライブラリーの両方が読み込まれた後に、フック `tty-setup-hook`を実行します。端末固有ライブラリーの任意の部分をオーバーライドしたい場合や、ライブラリーをもたない端末の初期化を定義したい場合は、このフックにフック関数を追加します。Section 33.2.2 [Hooks], page 509 を参照してください。

33.4.4 Emacs が init ファイルを探す方法

Emacs は通常はホームディレクトリー配下で init ファイルを探します。Section 33.4 [Init File], page 528 を参照してください。

Emacs は`~/.emacs.el`、`~/.emacs`、`~/.emacs.d/init.el`というファイル名で、この順序で init ファイルを探します。これらの名前のどれを使用するか選択できます (ホームディレクトリー直下の場所だけが場所のベース名として先頭にドットをもつことに注意)。

Emacs は XDG 互換の場所でも `init.el`を探します。デフォルトはディレクトリー `~/.config/emacs`です。これは環境変数 `XDG_CONFIG_HOME`をセットしてオーバーライドできます。この値はデフォルトの XDG init ファイルの名前の`~/.config`を置き換えます。しかし`~/.emacs.d`、`~/.emacs`、`~/.emacs.el`が存在する場合には常に優先されます。これは XDG ロケーションを使用するためには、それらの削除やリネームを行わなければならないことを意味します。

XDG ロケーションと`~/.emacs.d`がいずれも存在しなければ Emacs が`~/.emacs.d`を作成 (したがって以降の呼び出しの間もそれが使用される) することにも注意してください。

Emacs は使用を決定したディレクトリーを `user-emacs-directory`んセットします。

これがたとえ古いバージョンの Emacs にたいする後方互換性のためだとしても、モダンな POSIX プラットフォームは初期化ファイルを`~/.config`の配下に配置することを優先するので、誤った init ファイルに起因する問題のトラブルシューティングや init ファイルコレクションのアーカイブはそのディレクトリーのリネームにより行うことができます。古いバージョンの Emacs がそれらのカレン

トのデフォルト位置で設定ファイルを見つけるのを助けるために、以下の Emacs Lisp コードを実行することができます:

```
(make-symbolic-link ".config/emacs" "~/.emacs.d")
```

しかし suで開始されたシェルから Emacs を実行して環境変数 XDG_CONFIG_HOMEが未セットなら、Emacs は現在の見かけのユーザーではなく、あなた自身の初期化ファイルを探すことを試みます。このアイデアは、たとえスーパーユーザーとして実行しているときでも、自分のエディターカスタマイズを取得するべきだという考えです。

より正確には、最初に Emacs はどのユーザーの init ファイルを使用するか決定します。Emacs は環境変数 LOGNAMEと USERからユーザー名を取得します。どちらも存在しない場合、実効ユーザー ID を使用します。ユーザー名が実ユーザー ID とマッチしたとき、Emacs は HOMEを使用します。そうでない場合、Emacs はシステムのユーザーデータベースの、そのユーザー名に対応するホームディレクトリーを探します。

簡略化のために Emacs ドキュエントの残りの部分では、一般的には init ファイルにたいして単にカレントデフォルト位置~/.emacs.d/init.elを使用します。

33.4.5 init ファイル内の非 ASCII文字

init ファイルの文字列やキーバインディングに、アクセントつき文字などの非 ASCII文字が含まれる場合、それは言語やコーディングシステムに問題を起こすかもしれません。

init ファイルで非 ASCII文字を使用したい場合、init ファイルの最初の行に '`-*-coding: coding-system-*-`' タグを配して、問題となる文字をサポートするコーディングシステムを指定するべきです。Section 19.6 [Recognize Coding], page 225 を参照してください。なぜなら、非 ASCIIテキストのでコーディングにたいするにたいするデフォルトのコーディングシステムは、Emacs が init ファイルでそのような文字列を使用する個所を読み込むまでに、セットアップされていないかもしれないので、Emacs がその文字列を間違ってデコードする可能性があるからです。'`-*-coding: coding-system-*-`' を記述した場合、set-language-environment を呼び出す等、他の方法でコーディングシステムを変更する Emacs Lisp コードの追加は避けるべきです。

非 ASCII文字を直接使用せずに、Section "General Escape Syntax" in *The Emacs Lisp Reference Manual* で述べた文字エスケープ構文のいずれかを使う選択肢もあります。これらはすべて ASCII文字だけですべての Unicode コードポイントを指定できます。

非 ASCIIキーをバインドするには、ベクターを使用しなければなりません (Section 33.3.6 [Init Rebinding], page 522 を参照してください)。非 ASCII文字はメタキーとして解釈されるので、文字列構文は使用できません。たとえば:

```
(global-set-key [?char] 'some-function)
```

*char*を挿入するには、C-qとタイプしてからバインドしたいキーをタイプします。

33.4.6 早期初期化ファイル

Emacs のほとんどのカスタマイズは通常の init ファイルに配置するべきです。Section 33.4 [Init File], page 528 を参照してください。しかし Emacs のスタートアップで通常の init ファイルを処理するより早い段階で効果をもつカスタマイズが必要な場合があります。そのようなカスタマイゼーションは早期 init ファイル~/.config/emacs/early-init.elまたは~/.emacs.d/early-init.el に配置することができます。このファイルはパッケージシステムと GUI の初期化前にロードされるので package-enable-at-startup、package-load-list、package-user-dirのようなパッケージ初期化プロセスに影響を与える変数をカスタマイズできます。新たなパッケージのインストールだけに影響がある package-archivesのような変数やインストール済みパッケージを利用可能にするプ

ロセス以外は正規の init ファイルでカスタマイズできます。Section 32.3 [Package Installation], page 493 を参照してください。

通常の init ファイルに残すことができるカスタマイゼーションを `early-init.el`に移動することは推奨できません。早期 init ファイルは GUI の初期化前に読み込まれるので。GUI 機能に関連するカスタマイゼーションは `early-init.el`では信頼性をもって機能しないからです。対照的に通常の init ファイルは GUI の初期化後に読み込まれます。GUI 機能に依存するカスタマイズを早期 init ファイルで行わなければならない場合には、Emacs がスタートアップ時に提供する `window-setup-hook` や `tty-setup-hook`のようなフックを外して実行してください。Section 33.2.2 [Hooks], page 509 を参照してください。

早期 init ファイルの詳細な情報は Section “Init File” in *The Emacs Lisp Reference Manual* を参照してください。

33.5 永続的に認証情報を保つ

他のサービスに接続する Emacs パッケージには、たとえば *The Gnus Manual* や *The Tramp Manual* で見られるように、認証 (Section 5.7 [Passwords], page 38 を参照) を要求するものがいくつかあります。同じユーザー名とパスワードを何度も提供するのは煩わしいかもしれないので、Emacs は `auth-source`ライブラリーを通じて、認証情報を永続的に保つことを提案します。

デフォルトでは、認証情報はファイル`~/.authinfo`、`~/.authinfo.gpg`、または`~/.netrc`から取得されます。これらのファイルは以下のような、ftp プログラム由来として知られる、netrc ファイルと同様の構文をもちます:

```
machine mymachine login myloginname password mypassword port myport
```

同様に `auth-source`ライブラリーは現在のところ、クラシックな netrc バックエンド、JSON ファイル、Secret Service API、および標準的な Unix パスワードマネージャーの pass という、複数のストレージバックエンドをサポートしています。

これらの選択肢はすべて、ユーザーオプション `auth-sources`を通じてカスタマイズできます。Section “Help for users” in *Emacs auth-source* を参照してください。

設定されたバックエンドで見つからないようなパスワードがインタラクティブに入力された場合、いくつかのバックエンドはそれの永続的な保存を提案します。これはユーザーオプション `auth-source-save-behavior`をカスタマイズして、変更することができます。

34 一般的な問題への対処

意図しないコマンドをタイプした場合、その結果は不可解なことが多くあります。このチャプターでは、間違いをキャンセルしたり、不可解な状況から復帰するために何ができるかを示します。Emacs のバグとシステムクラッシュについても考察します。

34.1 中止と中断

`C-g`
`C-Break` (MS-DOS のみ)
: quit(中止): コマンドの実行、または途中までタイプしたコマンドをキャンセルします。

`C-]`
: 最内の再帰編集レベル (recursive editing level) を abort(中断) して、それを呼び出したコマンドをキャンセルします (`abort-recursive-edit`)。

`ESC ESC ESC`
: quit または abort のどちらか、意味のあるほうを行ないます (`keyboard-escape-quit`)。

`M-x top-level`
: 現在実行中のすべての再帰編集レベルを abort します。

`C-/`
`C-x u`
`C-_`
: バッファー内容にたいする直前の変更をキャンセルします (`undo`)。

完了する前のコマンドをキャンセルする方法は 2 つあります。それは `C-g`による *quit* と、`C-]`や `M-x top-level`による *abort* です。quit は途中までタイプしたコマンド、または実行中のコマンドをキャンセルします。abort は再帰編集レベルを抜けて、再帰編集を呼び出したコマンドをキャンセルします (Section 31.11 [Recursive Edit], page 484 を参照してください)。

`C-g`による quit は、途中までタイプしたコマンドから抜けたり、望まない数引数から抜け出す方法です。さらに、あるコマンドが実行中の場合、`C-g`は比較的安全にコマンドを停止します。たとえば、長い時間がかかる kill コマンドを quit した場合、すべてのテキストがバッファーに残るか、またはすべてのテキストが kill リングに残るか、もしかしたらその両方かもしれません。リージョンがアクティブの場合、Transient Mark モードがオフでなければ、`C-g`はマークを非アクティブにします (Section 8.7 [Disabled Transient Mark], page 57 を参照してください)。インクリメンタル検索の途中では、`C-g`は特別に振る舞います。検索を抜けるには 2 回連続して `C-g`をタイプします。詳細は、Section 12.1 [Incremental Search], page 105 を参照してください。

ミニバッファー内で `C-g`をタイプすると、ミニバッファーをオープンしたコマンドを終了して、ミニバッファーをクローズします。そのミニバッファーがもっとも最近オープンされたのでなければ (これは `minibuffer-follows-selected-frame`が `nil`の際に発生し得る。Section 5.1 [Basic Minibuffer], page 28 を参照)、`C-g`は同意を求めた後にそれより最近にオープンされたミニバッファーもクローズして、それらに関連するコマンドを終了します。

MS-DOS では、文字 `C-Break`が `C-g`のような文字の役割をします。MS-DOS ではユーザーとの相互作用を行なうとき以外に、実行中のコマンドで `C-g`を認識できないのが理由です。それとは対照的に、`C-Break`は常に認識できます。Section “MS-DOS Keyboard” in *Specialized Emacs Features* を参照してください。

`C-g`をタイプした瞬間に変数 `quit-flag`を `t`にセットすることにより、`C-g`は機能します。Emacs Lisp はこの変数を頻繁にチェックして、これが非 `nil`のときは quit します。Emacs が入力待ちのと

きにタイプしたときだけ、C-gは実際にコマンドとして実行されます。この場合に実行されるコマンドは、keyboard-quitです。

テキスト端末では、最初の C-gが認識される前に 2 回目の C-gで quit した場合は、emergency escape(緊急エスケープ) 機能がアクティブになり、シェルに戻ります。Section 34.2.7 [Emergency Escape], page 541 を参照してください。

quit できない状況もいくつか存在します。Emacs がオペレーティングシステムが何か行なうのを待つような場合、待ちが発生する箇所で特定のシステムコールにたいして、Emacs が特別な対処をしない場合、quit は不可能です。ユーザーが quit したいと望むようなシステムコールにたいして、わたしたちはこれを行なっていなすが、それでも処理できないケースに出会う場合もあります。とても一般的なケースの 1 つは、NFS を使用したファイルへの入出力待ちです。Emacs 自体は quit する方法を知っていますが、多くの NFS 実装は、NFS サーバーがハングしたとき、ユーザープログラムが NFS を待つのを止めることを、単に許していません。

C-] (abort-recursive-edit) による abort は、再帰編集レベルを抜けて、それを呼び出したコマンドをキャンセルするのに使用されます。C-gによる quit はこれを行なわず、行なうこともできません。なぜならこれは再帰編集レベルの中で、途中までタイプされたコマンドをキャンセルするからです。どちらの操作も有用です。たとえば再帰編集中に、数引数を入力するために C-u 8とタイプした場合、C-gでその引数をキャンセルして、その再帰編集レベルに留まることができます。

シーケンス ESC ESC ESC (keyboard-escape-quit) は、quit か abort のどちらかを呼び出します (多くの PC プログラムで ESCは “抜け出す” ことを意味するので、このような定義にしました)。これは C-gのように、プレフィクス引数のキャンセル、選択されたリージョンのクリアー、また問い合わせつき置換から抜け出すこともできます。また C-]のように、ミニバッファーから抜け出したり、再帰編集から抜け出すこともできます。これは C-x 1のように、フレームの複数ウィンドウ分割から抜け出すこともできます。これが行なうことができないのは、実行中のコマンドの停止です。これは通常のコマンドとして実行されるので、Emacs が次のコマンドのために準備ができるまで、これを認識しないからです。

コマンド M-x top-levelは、現在の再帰編集レベルからすべての再帰編集レベルを抜けるための、充分な回数の C-]コマンドと等価です。ミニバッファーがアクティブなときは、ミニバッファーも抜けます。C-]は、1 度に 1 レベル再帰編集レベルを抜けますが、M-x top-levelは、1 度ですべての再帰編集レベルを抜けます。C-]と M-x top-levelの両方とも、他のすべてのコマンドと同様 (そして C-gとは異なり)、Emacs がコマンドにたいして準備ができているときだけ効果があります。C-]は通常のキーで、このキーが意味をもつのは、それがキーマップでバインドされているときだけです。Section 31.11 [Recursive Edit], page 484 を参照してください。

厳密に言えば C-/ (undo) はキャンセルコマンドではありませんが、すでに実行を終えたコマンドをキャンセルすると考えることができます。undo 機能についての詳細は、Section 13.1 [Undo], page 131 を参照してください。

34.2 Emacs のトラブルへの対処

このセクションでは、キーボードコードのミクスアップ (mixup)、文字化け、メモリー不足、クラッシュやハングなど、Emacs が期待したとおりに動作しない状況の認識と対処法について説明します。

Emacs でバグを見つけたと思ったら何をすればよいかについては、Section 34.3 [Bugs], page 542 を参照してください。

34.2.1 再帰編集レベル

再帰編集レベルは、Emacs の重要かつ便利な機能ですが、それを理解していない場合は、うまく機能していないように見えるかもしれません。

モードラインの、メジャーモードやマイナーモードを囲む丸カッコ (parentheses) の周囲に、角カッコ (square brackets)‘[...]’がある場合、それは再帰編集レベルにいることを意味します。もしこれが目的でない場合、またはその意味を理解していない場合は、すぐに再帰編集レベルを抜けるべきです。これを行なうには、M-x top-levelとタイプします。Section 31.11 [Recursive Edit], page 484 を参照してください。

34.2.2 スクリーン上のゴミ

テキスト端末でテキストが間違って見える場合、まず行なうことはバッファーのテキストが間違っていないか確かめることです。画面全体を再描画するために、C-l (recenter-top-bottom) とタイプしてください。この後でスクリーンが正常に表示される場合、問題は前のスクリーンの更新にあります (そうでない場合は、以下のセクションを参照してください)。

ディスプレイ更新の問題は、使用している端末にたいする間違った terminfo エントリーの結果であることがしばしばあります。Emacs ディストリビューションのファイル etc/TERMSは、この種の既知の問題にたいする解決を与えます。INSTALLのセクションの中の 1 つは、これらの問題にたいする一般的なアドバイスを含みます。正しい terminfo エントリーを使用しているようなら、それは terminfo エントリーにバグがあるか、特定の端末タイプで発生する Emacs のバグである可能性があります。

34.2.3 テキスト内のゴミ

C-lがそのテキストが間違っていることを示す場合、最初に実際の結果を生成するのに何のコマンドをタイプしたか見るために、C-h l (view-lossage) とタイプします。それから C-x u (undo) を使用して、正しいと思える状態まで、1 つずつ変更を undo します。

バッファーの先頭または最後の大量のテキストが失われているように見える場合は、モードラインに単語 ‘Narrow’が表示されていないかチェックします。もしこれが表示されている場合、表示されていないテキストはまだ存在しますが、一時的に制限されています。これに再びアクセスできるようにするには、C-x n w (widen) とタイプします。Section 11.5 [Narrowing], page 81 を参照してください。

34.2.4 メモリー不足

‘Virtual memory exceeded’というエラーメッセージが表示された場合は、C-x s (save-some-buffers) で変更されたバッファーを保存してください。この方法は、バッファーを保存するのに最小限の追加メモリーを必要とします。Emacs はこのエラーが発生したときでも利用可能な予備メモリーを保持しており、それは C-x sが処理を完了するのに充分なはずです。予備メモリーを使用したとき、モードラインの先頭に ‘!MEM FULL!’が表示された場合、それは予備メモリーも使い切ったことを意味します。

変更されたバッファーを変更したら、この Emacs セッションを終了して別のセッションを開始するか、M-x kill-some-buffersを使用して、カレント Emacs ジョブのスペースを開放できます。これにより充分なスペースが開放された場合、予備メモリーは再充填され、モードラインから ‘!MEM FULL!’の表示が消えます。これは同じ Emacs セッションで、安全に編集を継続できることを意味します。

メモリー不足のときは、バッファーの保存や kill に M-x buffer-menuを使用しないでください。Buffer Menu はかなりの量のメモリーを必要とするので、予備メモリーの供給では不充分でしょう。

GNU/Linux システムでは、Emacs がメモリー不足 (out-of-memory) という状況の通知を受け取ることは通常はありません。かわりに Emacs プロセスがメモリー不足になると、OS が Emacs を kill するかもしれません。これは *out-of-memory killer*、あるいは *OOM killer* という名前で知られ

ている機能です。この動作が効力をもつ際にはメモリー不足という状況を Emacs がすぐに検知することはできず、上述のようなバッファーの保存をあなたに促すことはできないでしょう。ただし OS のこの動作をオフに切り替えることは可能であり、そうすれば Emacs が kill されてしまう前に、メモリー不足という状況をもっと有益なやり方で処理するチャンスが生まれます。これを行うにはスーパーユーザーになり、以下の行を`/etc/sysctl.conf`に含めてから、シェルプロンプトでコマンド `sysctl -p` を呼び出してください:

```
vm.overcommit_memory=2
vm.overcommit_ratio=0
```

上記のセッティングはシステム上の全プロセスに影響を与えること、そしてメモリー負荷が高いシステムでは、Emacs プロセスに限定されない一般的な動作であることに注意してください。

34.2.5 Emacs がクラッシュしたとき

Emacs はクラッシュを前提としていませんが、もしクラッシュした場合、exit する前にクラッシュレポート *(crash report)* を生成します。クラッシュレポートは標準エラーストリームにプリントされます。Emacs が GNU システムまたは Unix システムでグラフィカルなデスクトップから開始された場合、標準エラーストリームは一般的に`~/.xsession-errors`のようなファイルにリダイレクトされるので、そこでクラッシュレポートを探すことができます。MS-Windows では、クラッシュレポートは標準エラーストリームに加え、Emacs プロセスのカレントとディレクトリーの、`emacs_backtrace.txt` という名前のファイルに書き込まれます。

クラッシュレポートのフォーマットは、プラットフォームに依存します。GNU C ライブラリーを使用するいくつかのプラットフォームでは、クラッシュレポートには、クラッシュ前の実行状態を説明する *backtrace* が含まれ、これはクラッシュをデバッグする助けとなります。以下は GNU システムの例です:

```
Fatal error 11: Segmentation fault
Backtrace:
emacs[0x5094e4]
emacs[0x4ed3e6]
emacs[0x4ed504]
/lib64/libpthread.so.0[0x375220efe0]
/lib64/libpthread.so.0(read+0xe)[0x375220e08e]
emacs[0x509af6]
emacs[0x5acc26]
...
```

数字 '`11`' はクラッシュにたいするシステムのシグナル番号 — このケースでは segmentation fault — です。16 進数字はプログラムのアドレスで、これによりデバッグツールを使用して、ソースコード行に関連付けることができます。たとえば GDB コマンド '`list *0x509af6`'は、'`emacs[0x509af6]`' エントリーにたいするソースコード行をプリントします。システムに `addr2line`ユーティリティーがある場合、以下のシェルコマンドはソースコードの行番号とともに、backtrace を出力します:

```
sed -n 's/.*\[\(.*\)]$/\1/p' backtrace |
  addr2line -C -f -i -p -e bindir/emacs-binary
```

On MS-Windows, the backtrace looks somewhat differently, for example:< MS-Windows ではバックトレースの外観は若干異なります。以下は例です:

```
Backtrace:
00007ff61166a12e
00007ff611538be1
```

```
00007ff611559601
00007ff6116ce84a
00007ff9b7977ff0
...
```

したがって sedを通じたフィルタリングであり、ソースコードの行番号を表示するコマンドは以下のようになります

```
addr2line -C -f -i -p -e bindir/emacs-binary < backtrace
```

ここで *backtrace*は backtrace のコピーを含むテキストファイル名、*bindir*は Emacs 実行可能ファイルを含むディレクトリー名 (MS-Windows では Emacs を起動したディレクトリーにある `emacs_backtrace.txt`)、*emacs-binary*は Emacs 実行可能ファイル (GNU および Unix システムでは通常は `emacs`、MS-Windows および MS-DOS では `emacs.exe`) です。`-p`オプションがない古いバージョンの `addr2line`では、このオプションを省略してください。

core ファイルをサポートするシステムでは、Emacs はオプションでコアダンプ *(core dump)* を生成します。コアダンプはクラッシュ前のプログラムの状態に関する多くのデータを含むファイルで、通常 GDB のようなデバッガーにロードして調べられます。多くのプラットフォームでは、コアダンプはデフォルトで無効になっているので、(たとえばシェルのスタートアップスクリプトで) シェルコマンド '`ulimit -c unlimited`'を実行して、明示的に有効にしなければなりません。

34.2.6 クラッシュ後のリカバリー

Emacs、またはコンピューターがクラッシュした場合、クラッシュしたとき編集していたファイルを、自動保存ファイルからリカバリーすることができます。これを行なうには、再び Emacs を開始して、コマンド `M-x recover-session`とタイプしてください。

このコマンドは m 最初に中断されたセッションのファイルを、ファイルの日付とともにリストするバッファーを表示します。そこからリカバリーするファイルを、選択しなければなりません。通常リカバリーしたいファイルは、一番最近のセッションでしょう。選択したファイルにポイントを移動して、`C-c C-c`とタイプしてください。

その後 `recover-session`は、そのセッション中に編集していた各ファイルについて、検討を行ないます。そのようなファイルそれぞれについて、そのファイルをリカバリーするか尋ねるのです。あるファイルにたいして `y`と応えると、コマンドはファイルとファイルの自動保存ファイルの日付を表示して、再度そのファイルをリカバリーするか尋ねます。この 2 回目の質問にたいして同意するには、`yes`と応えなければなりません。`yes`と応えた場合、Emacs はそのファイルを visit しますが、テキストは自動保存ファイルから取得します。

`recover-session`が終了すると、リカバリーを選択したファイルが Emacs バッファーに表示されます。そこでファイルを保存する必要があります。それらを保存することだけが、そのファイル自身を更新するのです。

ファイルに関連付けられていないバッファーをリカバリーしたいときや、自動保存が重要な更新を記録するほど最新でなかった場合、最後の手段として — コアダンプが保存されていて、Emacs の実行ファイルからデバッグシンボルがストリップされていないという条件の元に — コアダンプからそれらを取得するために、GDB(GNU デバッガー) で `etc/emacs-buffer.gdb`スクリプトを使用することができます。

コアダンプを入手したら、すぐに `core.emacs`のような別の名前にリネームします。これにより、他のクラッシュによるコアダンプの上書きを防ぎます。

このスクリプトを使用するには、Emacs 実行ファイル名とコアダンプのファイル名を、'`gdb /usr/bin/emacs core.emacs`'のように指定します。`(gdb)`プロンプトで、'`source`

/usr/src/emacs/etc/emacs-buffer.gdb'としてリカバリースクリプトをロードします。それから利用可能なバッファーを見るために、コマンド ybuffer-listとタイプします。これは各バッファーにたいして、バッファー番号をリストします。バッファーを保存するには、ysave-bufferを使用します。ここでバッファー番号とそのバッファーを書き込むファイル名を指定します。すでに存在するファイル名を使用するべきではありません。ファイルがすでに存在する場合、このスクリプトはそのファイルの古い内容のバックアップを作成しません。

34.2.7 緊急エスケープ

テキスト端末では、1 回目の C-gにたいして Emacs が実際に反応して quit する前に、2 回目の C-gをタイプすると、緊急エスケープ *(emergency escape)* が、Emacs を即座にサスペンドします。これにより、どんなにひどくハングしていても、常に GNU Emacs を抜け出すことができます。物事が正しく処理されている場合、Emacs は最初の C-gを素早く認識・処理するので。2 回目の C-gは緊急エスケープを引き起こしません。しかし何らかの問題が、Emacs が最初の C-gを処理するのを妨げる場合、2 回目の C-gでシェルに戻ります。

緊急エスケープによるサスペンドから Emacs を再開する場合、サスペンド前に行なっていた何かに戻る前に、Emacs は回復の報告と、以下の 2 つの質問をします:

```
Emacs is resuming after an emergency escape.
Auto-save? (y or n)
Abort (and dump core)? (y or n)
```

質問に応えるには、それぞれにたいして yまたは nの後に、RETをタイプします。

'Auto-save?'にたいして yと応えると、自動保存が有効なすべての編集されたバッファーの自動保存を、即座に行ないます。nと応えると、これをスキップします。この質問は、Emacs が安全に自動保存を行えないような、何らかの状況にある場合は省略されます。

'Abort (and dump core)?'にたいして yと応えると、Emacs はクラッシュしてコアダンプします。これは専門家 (wizard) が、なぜ Emacs が最初の C-gで quit しなかったかを見つけ出すことを可能にします。コアダンプの後、実行は継続されません。

この質問に nと応えた場合、Emacs は実行を再開します。運がよければ、Emacs は最終的に quit 要求を行なうでしょう。そうでない場合、連続して C-gをタイプして、緊急エスケープを再度呼び出します。

実際は Emacs がハングしているのではなく、遅いだけの場合、本当に意味するところを意図せずに、2 連 C-gの機能を呼び出してしまうかもしれません。この場合は、2 つの質問の両方に nを応えれば、前の状態に戻ることができます。やがて要求した quit が行なわれるでしょう。

緊急エスケープはテキスト端末だけでアクティブになっています。グラフィカルなディスプレイでは、マウスを使用して Emacs を kill したり、他のプログラムに切り替えることができます。

MS-DOS では緊急エスケープを発生させるために、C-Breakを 2 回タイプしなければなりません — しかしシステムコールがハングしたり、Emacs が C コードのタイトなループにハマっているときは機能しないケースがあります。

34.2.8 DELで削除できない場合

すべてのキーボードには多くのキーがありますが、通常 BACKSPACEとラベルされたキーは、最後にタイプした文字を削除するのに使用されます。Emacs では、このキーは DELに等しいと想定されています。

グラフィカルなディスプレイで Emacs を開始したとき、Emacs はどのキーが DELなのか自動的に決定します。いくつかの特殊なケースでは、Emacs がシステムから間違った情報を取得して、BACKSPACEが後方ではなく前方に削除する場合があります。

Deleteというキーをもつキーボードもあります。これは通常、前方に削除するために使用されます。Emacs でこのキーが後方に削除を行なう場合も、Emacs が間違った情報— ただし反対の意味の— を受け取ったことを意味します。

テキスト端末で、BACKSPACEが文字を削除するかわりに、Control-hのようなヘルプコマンドのプロンプトを表示する場合、それはこのキーが実際は'BS'文字を送っていることを意味します。Emacs は BSを DELと扱うべきですが、そうしていないのです。

これらのケースのすべてにおいて、直ちに改善できることは同じでで、それはコマンド M-x normal-erase-is-backspace-modeを使用する方法です。これは Emacs が DELの処理をサポートする 2 つのモードを切り替えるので、もし Emacs が間違ったモードで開始された場合、正しいモードに切り替えることができます。テキスト端末では、BSが DELとして扱われる場合にヘルプを見たいときは、C-hのかわりに F1を使用します。C-?が文字コード 127 を送る場合は、このキーも機能するでしょう。

すべての Emacs セッションで問題を解決するには、初期化ファイル (Section 33.4 [Init File], page 528 を参照してください) に以下の行の 1 つを記述します。上記の最初のケースでは、BACKSPACE が後方ではなく前方に削除を行なうので、BACKSPACEが DELとして動作するように、以下の行を使用します:

```
(normal-erase-is-backspace-mode 0)
```

他の 2 つのケースでは、以下の行を使用します:

```
(normal-erase-is-backspace-mode 1)
```

すべての Emacs セッションで問題を解決する別の方法は、変数 normal-erase-is-backspace をカスタマイズする方法です。BSまたは BACKSPACEが DELとなるようにモードを指定するには値 t、他のモードにたいしては nilを指定します。Section 33.1 [Easy Customization], page 499 を参照してください。

34.3 バグの報告

Emacs でバグを見つけたと思ったときは、それを報告してください。それを fix することは約束できませんし、それがバグであると常に認める訳ではありませんが、もちろんそれについて知りたいのです。追加したいと考える機能についても、同じことが言えます。このセクションはあなたが見つけたのがバグかどうかを判断したり、バグの場合に有効なバグレポートを作成する助けとなるでしょう。

バグの可能性がある何かを発見した際の一般的な手順は以下のようになります:

- あなたが発見したものが既知の問題なのか、あるいはすでに報告および/または訂正されているバグなのかを確認してください。既知の問題やバグを調べる方法については、Section 34.3.1 [Known Problems], page 543 で答えが得られるでしょう。
- あなたが見た挙動がバグなのか確信がもてなければ、Section 34.3.2 [Bug Criteria], page 544 を参照のしてください。そこには Emacs において明確なバグだとわたしたちがみなすものについて記述されています。
- バグを見つけたと判断したら Section 34.3.3 [Understanding Bug Reporting], page 544 を参照してください。わたしたちが問題の再現と調査を容易に行えるように、あなたが何を見つけたかをもっとも効率的に記述する助けとなる筈です。
- 次に Section 34.3.4 [Checklist], page 545 を参照してください。そこにはバグレポートを送信する方法、および含める情報についての詳細が記述されています。簡単に述べると、電子メールを通じてバグレポートを送信する助けとなるコマンド report-emacs-bugを通じてそれらを行うことになります。バグレポートの送信によってバグの調査および訂正のプロセスが始まり、そ

のバグに関して議論する電子メールのコピーを受け取ることになるでしょう。更に詳細な情報の提供や、見込みのある訂正のテスト等を求められるかもしれません。

- 最後にバグの修整、新機能の追加、ドキュメントの改善など Emacs にたいして特定の変更を提案したい場合には、どうか Section 34.3.5 [Sending Patches], page 550 を参照してください。そこにはその類いの変更を送信する際に関する詳細が記述されています。

34.3.1 既存のバグレポートの既知の問題を読む

バグを報告する前に、少しでも可能ならわたしたちが既に把握済みのバグではないか確認してください。実際にはそれがもっと後の Emacs リリースや、Emacs の開発バージョンですでに fix されているかもしれません。以下は既知の問題について読むことができる主な場所のリストです:

- `etc/PROBLEMS`ファイル。`C-h C-p`とタイプして読むことができます。このファイルには Emacs をコンパイル、インストール、実行するときに出会うであろう既知の問題について、特に他のソフトウェアに起因する問題のため Emacs では解決が困難な問題に重点を置いた詳細リストが含まれています。次善策や解決策の提案も、たくさんあります。

- `https://debbugs.gnu.org`の GNU Bug Tracker。Emacs のバグや問題は‘`emacs`’パッケージの下のトラッカーにファイルされています。トラッカーには各バグの状態、最初のバグレポート、バグ報告者、およびバグに関する議論や訂正を共有する Emacs 開発者によるフォローアップメッセージについての情報が記録されています。subject、severity、その他の条件でバグを検索できます。

  ウェブページでバグトラッカーを閲覧するかわりに、`debbugs`パッケージを使用して、それを Emacs から閲覧できます。このパッケージはパッケージメニュー (Chapter 32 [Packages], page 490 を参照してください) を通じてダウンロードできます。このパッケージは、バグをリストするコマンド `M-x debbugs-gnu`、特定のバグを検索する `M-x debbugs-gnu-search`を提供します。Emacs メンテナーにより適用されるユーザータグは、`M-x debbugs-gnu-usertags`で表示されます。

- ‘`bug-gnu-emacs`’メーリングリスト (ニュースグループ‘`gnu.emacs.bug`’も利用可能)。リストのアーカイブは`https://lists.gnu.org/mailman/listinfo/bug-gnu-emacs`で見ることができます。このリストはバグトラッカーに送られた Emacs バグレポートとドローアップメッセージの、mirror として機能します。これにはバグトラッカーが導入される前 (2008 年以前) の古いバグレポートも含まれています。

  もし望むなら、メーリングリストに登録できます。このリストの目的は Emacs メンテナーにバグと機能リクエストの情報を提供するためのもので、報告には大量のデータが含まれるかもしれないことに注意してください。購読者はこれについて不満を言うべきではありません。

- ‘`emacs-pretest-bug`’メーリングリスト。このリストは今は使用されておらず、主に歴史的な興味のためのものです。一時は (たとえばまだリリースされていない)Emacs 開発バージョンのために使用されていました。2003 年から 2007 年中頃までのアーカイブは、`https://lists.gnu.org/r/emacs-pretest-bug/`で見ることができます。このリストに送信された電子メールメッセージは、現在では‘`bug-gnu-emacs`’にリダイレクトされています。

- ‘`emacs-devel`’メーリングリスト。このメーリングリストにバグを報告する人がときどきいます。しかしこのリストの主な目的は違うので、バグレポートはバグリストに送るほうがよいでしょう。バグを報告する前に、この一覧を読んだことに感謝を感じることはありません。

34.3.2 バグがあったとき

Emacs が不正なメモリー位置にアクセスする場合 (“segmentation fault” と呼ばれます)、または (“disk full” のようなメッセージではなく) プログラムに問題があることを示す、オペレーティングシステムのエラーメッセージとともに終了する場合、それは確実にバグです。

Emacs がバッファーの内容を正しく対応して表示しないとき、それはバグです。しかしバッファーのナローイング (Section 11.5 [Narrowing], page 81 を参照してください) チェックするべきです。これはバッファーの一部を隠して、表示される方法を変更できるので、バグではありません。

コマンドが永久に完了しないなら、それはバグですが、本当に Emacs のせいか確認しなくてはなりません。コマンドの中には単に長時間かかるものがいくつかあります。`C-g` (MS-DOS では `C-Break`) をタイプしてから、`C-h l`で、Emacs が受け取った入力が、あなたがタイプしようと意図したものなのか確認します。その入力が、あなたが素早く処理されるべきだとわかっているものだった場合は、バグを報告してください。そのコマンドが長時間かかるものか判らない場合は、マニュアルを調べるか、協力してくれる人に尋ねてください。

あなたの親しんでいるコマンドが、コマンドの通常の定義が正当なのに、Emacs のエラーメッセージを表示する場合、それはおそらくバグです。

コマンドが間違ったことを行なうなら、それはバグです。しかしそのコマンドが何を行なうべきか確実に知っているか確認してください。そのコマンドに詳しくない場合、コマンドは実際は正しく動いているのでしょう。疑うなら、コマンドのドキュメント (Section 7.2 [Name Help], page 45 を参照してください) を参照してください。

あるコマンドの意図された定義が、それを編集するための最良の定義ではないこともあります。これはとても重要な問題の一種ですが、判断の問題でもあります。いくつかの既存の機能にたいする無知から、そのような決定を行なうのは簡単でもあります。通常の方法でドキュメントをチェックして、それを理解したと確信し、あなたがやりたいことが不可能だと確実に判るまでは、そのような問題にたいして不満を言わないのが、おそらく最良です。他の Emacs ユーザーにも尋ねてみましょう。マニュアルを注意深く読んだ後でも、そのコマンドが何を想定しているか確信がもてないときは、不明解な単語を index(索引) や glossary(用語集) でチェックしましょう。

注意深くマニュアルを読んだ後でも、そのコマンドが何を行なうべきか判らないとき、それは報告すべきマニュアルのバグであることを示します。Emacs のエクスパートでない人— あなたを含めて — にたいして、すべてを明解にするのがマニュアルの役目です。プログラムのバグと同様に、ドキュメントのバグレポートは重要です。

関数や変数のビルトインドキュメントがマニュアルと異なる場合、どちらか一方が間違っていなければならないので、これはバグです。

Emacs の一部ではないパッケージについての問題は、そのパッケージの開発者にそれらを報告することにより開始したほうが良いでしょう。

34.3.3 バグレポートの理解

バグがあると判断したときはそれを報告すること、そして有用な方法で報告することが重要です。もっとも有用なのは、Emacs を起動するシェルコマンドから、問題が発生するまで、何のコマンドをタイプしたか、そしてそれらのコマンドをタイプしたことにより生成された効果に関する正確な記述です。

バグレポートのもっとも重要な原理は、事実を報告することです。仮定や口頭の説明は、詳細な生データの代替にはなりません。事実の報告は簡単ですが、多くの人は事実のかわりに仮定の説明をしようと懸命に努め、それを報告するのです。その説明が Emacs が実装されている方法にたいする仮定にもとづく場合、それらは役に立たないかもしれません。その一方で事実の欠落により、わたしたちはバグについての実際の情報を得られないでしょう。実際に問題をデバッグして、推定を超える説明を報告したい場合、それは有用です— しかし、どうか生の事実も同様に含めてください。

たとえば、C-x C-f /glorp/baz.ugh RETとタイプして、ファイルを visit したとき、そのファイルが偶然大きい (とあなたは知っている) ファイルで、Emacs が 'I feel pretty today'と表示したとします。バグレポートにはすべての情報が必要になります。あなたは問題がファイルのサイズにあると仮定して、"大きなファイルを visit したら、Emacs が 'I feel pretty today'と表示します"、などと報告すべきではありません。これはわたしたちが "推測説明 (guessing explanations)" と呼ぶものです。ファイル名に 'z'があるという事実が、問題の原因かもしれません。もしそうなら、あなたの報告を受け取ったとき、わたしたちは大きなファイルで問題の再現を試み、それらのファイル名にはおそらく 'z'が含まれておらず、問題を確認できないでしょう。名前に 'z'が含まれるファイルを visit してみるべきだと、推測できる方法はありません。

C-x C-fはおろか、"ファイルを visit して" とさえ言うべきではありません。なぜならファイルが visit される方法は複数あり、それらの方法すべてにおいて問題が再現されるか確証がないからです。同様にテキストを入力する方法では、"その行に 3 文字あるとき" ではなく、もしもテキストを入力したのであれば "RET A B C RET C-pとタイプした後"、すなわちあなたの場合に問題が発生したテキストについて教えてください。

可能なら、すぐにバグを再現するために emacs -Q(Emacs は初期のカスタマイズなしで開始されます。Section C.2 [Initial Options], page 576 を参照してください) で Emacs を呼び出して、バグを発生させるステップを繰り返してみてください。この方法でバグを再現できたら、あなたの個人的なカスタマイズをバグから除外して、バグが容易に再現するようにしてください。バグレポートは、Emacs を emacs -Qで開始したことから始まり、バグを再現させる正確な一連のステップを続けるべきです。可能ならバグを再現するのに必要な、正確なファイル内容を報告してください。

emacs -Qでは再現できないバグもいくつかあります。結局は再現するのが難しいバグもあります。そのような場合、何を行なったかを報告すべきです — が、前述したように、どうか最初にバグを発生させた生の事実を固持してください。

報告したい複数の問題がある場合は、どうかそれらを個別のバグとしてそれぞれ報告してください。

34.3.4 バグレポートのためのチェックリスト

バグを報告する前に、まずその問題がすでに報告されていないか、確認を試みてください (Section 34.3.1 [Known Problems], page 543 を参照してください)。

もし可能なら、その問題がすでに fix されていないか、最新リリース版の Emacs も試してみてください。同様に、最新の開発版を試してみるのもよいでしょう。これがある人にとっては簡単でないことは認識しているので、バグを報告する前に、絶対にこれを行なわなければならないと思わないでください。

Emacs でバグレポートを書くベストな方法は、コマンド M-x report-emacs-bugを使用する方法です。これはメールバッファー (Chapter 29 [Sending Mail], page 420 を参照してください) をセットアップして、自動的にいくつかの重要な情報を挿入します。しかし、すべての必要な情報は提供できません。だから以下のガイドラインを読んで、それに従うべきです。そうすればメッセージを送る前に、他の重大な情報を手で入力できます。M-x report-emacs-bugによって挿入されたいくつかの情報は、適切ではないと感じるかもしれませんが、完全に確信があるのでなければそれを残してください。そうすれば開発者たちがそれを判断できます。

レポートを記述し終えたら、C-c C-cとタイプすると、それは Emacs メンテナー bug-gnu-emacs@gnu.orgに送られます Emacs の中からメールを送れなければ、バグレポートのテキストを普段使用しているメールクライアントにコピーして、そのアドレスに送信できます (システムがサポートしていれば C-c M-iで Emacs に行なわせることができる)。または、そのアドレスに問題を説明する簡単なメールを送ることもできます。以下に示す必要な情報を含めてください。

(問題の訂正や新機能の実装等で)Emacs にコードを提供したい場合には、Emacs の Issue Tracker に patch を送るのが一番簡単にこれを行う方法です。`M-x submit-emacs-patch`コマンドを使用してください。これはバグを報告する際と同じように機能します。Section 34.3.5 [Sending Patches], page 550 を参照してください。

どのような場合でもレポートは '`bug-gnu-emacs`'メーリングリストに送られて、`https://debbugs.gnu.org`の GNU Bug Tracker に保管されます。報告について、より詳細な情報を尋ねる必要がある場合のために、どうか有効な返信用アドレスを含めてください。提出されたレポートは調停されるので、Tracker で実際にレポートが見られるようになるまで遅れが生じることもあります。

バグを報告するために GNU Bug Tracker がどのように機能するか知る必要はありませんが、もし望むなら、トラッカーのオンラインドキュメント (`https://debbugs.gnu.org/Advanced.html`) で、使用できるさまざま機能を見ることができます。

'`bug-gnu-emacs`'メーリングリストに送られたすべてのメールは、'`gnu.emacs.bug`'ニュースグループにもゲートウェイされます。この逆も真ですが、バグレポート (または返信) をニュースグループにポストしないでください。これにより、さらに情報を尋ねるためにあなたに連絡するのが困難になるのと、それがバグトラッカーと充分に統合されていないからです。

データが 500,000 バイトを超える場合は、どうかそれを直接レポートに含めないでください。要求されたら送るという提案に留めるか、データをオンラインで利用可能にしてその場所を知らせてください。大きい添付ファイルは圧縮して送信するのが最善です。

GNU Bug Tracker はあなたのレポートにバグ番号を割り当てます。そのバグをフォローする議論では Tracker がバグの議論を追跡できるように、受信者リストにあるバグのアドレスを消さないでください。バグのアドレスは '*nnnnn*`@debbugs.gnu.org`'のようなアドレスです。ここで *nnnnn* はバグ番号です。

メンテナーがバグを詳細に調べられるように、レポートには以下の事項を含めるべきです:

- あなたが間違いと主張する動作の説明。たとえば "Emacs プロセスが致命的なシグナルを受け取った" とか "結果のテキストは以下だが、これは間違いだと思う" など。

  もちろん、Emacs が致命的なシグナルを受け取るというバグなら、見逃すことはないでしょう。しかし、そのバグが正しくないテキストの場合、メンテナーは何が間違っているか気づかないかもしれません。なぜそのような危険を放置するのですか?

  あなたが遭遇した問題が致命的なシグナルの場合でも、明示的にそれを告げるべきです。何か奇妙なこと — たとえばあなたのソースコピーの同期がとれていない、またはあなたのシステムの C ライブラリーにバグがある (これはあり得ます) — が起こっているとしましょう。あなたのコピーはクラッシュするかもしれませんが、私たちのコピーはクラッシュしないでしょう。あなたがクラッシュすることを告げていれば、わたしたちの Emacs はクラッシュしないので、バグはなかったとわたしたちは言うことができます。クラッシュすることを告げていない場合、わたしたちはバグがあるかどうか知ることができません— わたしたちの観察から、なんらかの結論を描くことも不可能です。

  問題の挙動とそれを再現する方法の記述には、通常だと以下に挙げる観点のうち 1 つ以上が必要です:

  – バグを再現するのに必要なファイルの完全なテキスト。

    ファイルを visit せずに問題を発生させる方法を説明できるなら、ぜひそうしてください。これによりデバッグがとても簡単になります。ファイルが必要な場合、わたしたちがファイルの正確な内容を見られるよう確実にしてください。たとえば、行末にスペースがあるか、バッファーの最終行の後に改行があるかが問題となる場合があります (最終行が終端されているかどうかを心配すべき理由はないかもしれませんが、それがバグだと報告することを試みてください)。

– バグを再現するためにタイプする必要がある、正確なコマンド。少しでも可能なら、‘-Q’オプション (Section C.2 [Initial Options], page 576 を参照してください) で Emacs を開始したときからの、完全なレシピを送ってください。このオプションはあなたの個人的なカスタマイズをバイパスします。

  Emacs への入力を正確に記録する方法の 1 つとして、それを dribble ファイルに書き込む方法があります。このファイルを開始するにはコマンド `M-x open-dribble-file` を使用します。このコマンドからその Emacs プロセスが kill されるまで、Emacs はすべての入力を指定された dribble ファイルにコピーします。機密情報 (パスワードなど) は、dribble ファイルへの記録を終了させることに注意してください。

– Emacs Manual や Emacs Lisp Reference Manual が、実際の Emacs の振る舞いを記述できていない、またはテキストが分かりにくいといったバグの場合、間違いだと思うテキストをマニュアルからコピーしてください。そのセクションが小さければ、セクション名だけで充分です。

– バグの徴候が Emacs のエラーメッセージの場合、エラーメッセージの正確なテキストと、Emacs の Lisp プログラムがどのようにしてエラーに至ったかを示す、backtrace を報告するのが重要です。

  エラーメッセージを正確に取得するには、それを`*Messages*`バッファーからバグレポートにコピーします。一部だけではなく、すべてをコピーしてください。

– Lisp の世界にロードしたプログラム (初期化ファイルを含みます) をチェックしてください。任意の変数にたいするセットは、Emacs の機能に影響を与えるかもしれません。初期化ファイルをロードせずに、フレッシュな状態で開始された Emacs でも、その問題が発生するか確認してください (`-Q`スイッチで Emacs を開始することにより init ファイルのロードを抑止できます)。それで問題が発生しなかったら、問題を発生させるために Lisp の世界にロードしなければならないプログラムの、正確な内容を報告しなければなりません。

– その問題が init ファイルや Emacs の標準システムの一部ではない他の Lisp プログラムに依存する場合、最初にそれらのプログラムのメンテナーに苦情を訴えて、それがバグでないことを確認すべきです。彼らが機能すると思われる方法で Emacs を使用してバグを確認した後、彼らがそのバグを報告すべきです。

– GNU Emacs のソース中の何かについて言及したい場合、数行のコンテキストとともにそのコードの行を示してください。行番号だけを示すのは止めてください。

  開発ソースの行番号と、あなたのソースの行番号は一致しません。行番号だけでは、あなたのバージョンでその行番号がどのコードをさすのか、メンテナーが判断するのに余分な作業を要しますし、それに確信をもつこともできません。

– テキスト端末上での表示のバグである可能性がある場合は、端末タイプ (環境変数 `TERM`の値)、その端末にたいする`/etc/termcap`の完全な termcap エントリー (このファイルはすべての機種で同じではありません)、Emacs が実際に端末に送った出力。

  端末の出力を収集するためには、Emacs を開始した直後にコマンド `M-x open-termscript` を呼び出すという方法があります。このコマンドは Emacs プロセスが kill されるまでの間に端末出力をどこに記録するかファイル名の入力を求めます。問題が Emacs のスタートアップ時に発生するようなら、

```
(open-termscript "~/termscript")
```

  という Lisp 式を Emacs の初期化ファイルに記述すれば、Emacs が最初にスクリーンを表示する際に termscript ファイルがオープンされます。

  警告: バグをシミュレートする端末タイプへのアクセスなしに、端末依存バグを fix するのは、しばしば困難で、不可能なときもあります。

- Emacs のバージョン番号。これがないと、GNU Emacs のカレントバージョンで, バグを探す意義があるかを知ることができません。

  この情報は、`M-x report-emacs-bug`により自動的にレポートに含まれますが、レポートでこのコマンドを使用しない場合は、`M-x emacs-version RET`とタイプして、バージョン番号を取得できます。このコマンドが機能しない場合、たぶんあなたは GNU Emacs ではない他の何かを使っているので、どこか他のところにそのバグを報告する必要があるでしょう。
- 使用している機種のタイプ、およびオペレーティングシステム名とバージョン番号 (繰り返しになりますが、これらは `M-x report-emacs-bug` で自動的にレポートに含まれます)。`M-x emacs-version RET` もこの情報を提供します。`*Messages*`バッファーからコマンドの出力をコピーするか、あるいはカレントバッファーにバージョン情報を挿入するために `C-u M-x emacs-version RET` を使用すれば、すべてを正確に取得できます。
- Emacs のビルド時に `configure`に与えたコマンドライン引数 (`M-x report-emacs-bug`により自動的にレポートに含まれます)。
- Emacs のソースにたいして行なった変更の完全なリスト (わたしたちには、変更された Emacs のバグを詳細に調べる時間はないでしょう。しかし変更を行なっていて、それをわたしたちに告げないとしたら、それはわたしたちを野性のダチョウ狩りに送り出すようなものです)。

  これらの変更について正確に記述してください。英語による説明では充分ではありません。それらにたいする統一コンテキスト diff(unified context diff) も送ってください。

  独自のファイル追加や、他の機種へのポートも、ソースの変更です。
- GNU Emacs の標準的なインストール手順からの、その他あらゆる逸脱の詳細。
- 非 ASCIIまたは国際化されたテキストと関連性がある場合は、Emacs を開始したときの locale。これは `M-x report-emacs-bug`によって自動的に含められます。GNU/Linux と Unix システム、または Bash のような POSIX スタイルのシェルを使用している場合には、かわりに以下のシェルコマンドを使用して関連する値を見ることができます:

```
echo LC_ALL=$LC_ALL LC_COLLATE=$LC_COLLATE LC_CTYPE=$LC_CTYPE \
  LC_MESSAGES=$LC_MESSAGES LC_TIME=$LC_TIME LANG=$LANG
```

  システムにあれば GNU/Linux および Unix コマンドを使用して、locale セッティングを表示することもできます。

以下はバグレポートには不要な事柄です:

- バグを取り巻く状況の説明 — これは再現可能なバグにたいしては必要ありません。

  バグに遭遇したとき人は、入力ファイルを変えてバグが発生しなくなるか、影響がないかなどを詳しく調べるのに、多くの時間を費やすことがしばしばです。

  これは大抵多くの時間がかかる割に、とても有用とは言えません。なぜならわたしたちがバグを探す方法は、ブレークポイントを設定したデバッガーの元で 1 つの例を実行することであり、一連の例から得られる推論ではないからです。追加の例を探すのを止めることにより、あなたも時間を節約できるでしょう。すぐにバグレポートを送って、編集作業に戻り、報告すべき他のバグを探す法がよいでしょう。

  もちろんオリジナルのかわりに、簡単な例を見つけることができたら、そちらのほうが便利です。出力中のエラーは簡単に見分けられますし、デバッガーでの実行も時間が短くなります

  しかし単純化は必須ではありません、これを行なうことができなかったり、試す時間がない場合は、どうかオリジナルのテストケースでバグを報告してください。
- コアダンプファイル。

  コアダンプによるデバッグは有用ですが、それはあなたの Emacs 実行ファイルと、あなたのマシンだけで行なうことができます。したがって Emacs メンテナーにコアダンプを送るのは、有

益ではないでしょう。何よりも、コアダンプをメールのバグレポートに含めないでください! そのような巨大なメッセージは、すこぶる迷惑です。

- Emacs を実行したときのシステムコールトレース。

  システムコールトレースは、ある特別な種類のデバッグにはとても有用ですが、有用な情報が少ない場合がほとんどです。したがって、クラッシュに関する情報を報告する一番の方法は、システムコールトレースを送ることだと考えているように思える人が多いのは奇妙です。おそらくソースコードやデバッグシンボルがないプログラムをデバッグする経験がもたらす、習慣的なやり方なのでしょう。

  ほとんどのプログラムでは、システムコールトレースより、backtrace のほうが、通常はずっとずっと参考になります。完全な情報を得るには、変数の値を表示して、それらを `pr`で Lisp オブジェクトとしてプリントすることにより backtrace を補完すべきであるとはいえ、シンプルな backtrace のほうが一般的により参考になります (上記参照)。

- バグにたいするパッチ。

  バグにたいするパッチは、それが良いものなら有用です。しかし、そのパッチで充分だと思い込んで、テストケースのような、バグレポートに必要な他の情報を省略しないでください。わたしたちはそのパッチに問題を見つけて別の方法で fix すると判断するかもしれないし、結局はそれを理解できないかもしれません。わたしたちが、あなたが fix しようと試みているバグを理解できなければ、そしてなぜそのパッチが改善なのかを理解できなければ、私たちはそれを採用できません。わたしたちにパッチを理解、インストールさせることを容易にするためのガイドラインについては、Section 34.3.5 [Sending Patches], page 550 を参照してください。

- バグが何か、何に依存するかについての推測。

  そのような推測は通常間違っています。エキスパートでさえ、事実を見つけるために最初にデバッガーを使用しなければ、そのような事柄を正しく推測できないのです。

以下に Emacs をデバッグしてバグに関する追加情報を提供したい場合に役に立つアドバイスのいくつかを挙げます:

- エラーにたいする backtrace を作成するには、エラーが発生する前に `M-x toggle-debug-on-error`を使用します (つまりこのコマンドを与えた後でバグを発生させなければならない)。これはエラーにより backtrace を表示する Lisp デバッガーを開始するためです。デバッガーの backtrace をバグレポートにコピーしてください (関連する`*.el`を見つけてロードする方法を知っていれば、エラーを発生させる前にそれらをロードしておけばより詳細なバックトレースが得られるのでそうしてほしい)。

  エラーのデバッグには Edebug の使用をお薦めします。Edebug パッケージによる Emacs Lisp プログラムのデバッグに関する情報については Section “Edebug” in *the Emacs Lisp Reference Manual* を参照してください。

  このデバッガーの使用は、バグを再現する方法を知っているときだけ利用可能です。バグを再現できなければ、最低でもエラーメッセージ全体をコピーしてください。

- Emacs が無限ループや、とても長い処理にハマっているように見えるとき、(変数 `debug-on-quit`が非 `nil`の場合は)`C-g`とタイプすると Lisp デバッガーを開始して、backtrace を表示します。この backtrace は、そのような長い loop のデバッグにたいして有用なので、backtrace を生成できたら、バグレポートにコピーしてください。

  (`inhibit-quit`がセットされている等で)、`C-g`に Emacs が応答しない場合、Emacs の外から `debug-on-event`で指定されたシグナル (デフォルトは SIGUSR2) を送ることにより、デバッガーに入ることができます。

- GDB のような C デバッガーからの追加情報は、そのマシンを利用できなくても問題を見つけることを可能にするかもしれません。GDB の使い方を知らなければ、どうか GDB マニュアルを読んでください— 非常に長いという訳ではありませんし、GDB は簡単に使用できます。GDB マニュアルを含む GDB ディストリビューションはオンライン形式で見つけることができ、ほとんどは Emacs ディストリビューションと同じ場所で見つけることができます。GDB の下で Emacs を実行するためには、Emacs をコンパイルした場所の `src`サブディレクトリーに移動してから `gdb ./emacs`とタイプする必要があります。GDB はカレントディレクトリーの `.gdbinit` を読み込むので、カレントディレクトリーが `src`であることが肝要です (GDB 内から `source ./.gdbinit`とタイプしてファイルを読み込むよう GDB に指示することも可)。

  しかし、何がバグを引き起こしたかを示すような追加の情報を収集するときは、考える必要があります。

  たとえば、多くの人は C レベルの backtrace だけを送ってきますが、これだけではとても有用とは言えません。引数つきのシンプルな backtrace は、GNU Emacs の中で何が起こっているかを少ししか伝えないことがしばしばです。なぜなら backtrace にリストされたほとんどの引数は、Lisp オブジェクトへのポインターだからです。これらのポインターの数値は、何であれ意味をもちません。問題となるのはポインターが指すオブジェクトの内容 (そして、その内容自身もポインターの場合がほとんどです) なのです。

  有用な情報を提供するためには、Lisp オブジェクトの値を Lisp 表記で示す必要があります。基底スタックの近傍のスタックフレームのいくつかで、Lisp オブジェクト変数それぞれにたいしてこれを行ないます。どの変数が Lisp オブジェクトであるかはソースを調べます。なぜならデバッガーはそれらを整数と判断するからです。

  Lisp 構文で変数の値を表示するには、最初にその値をプリントして、それから Lisp オブジェクトを Lisp 構文でプリントするために、ユーザー定義の GDB コマンド `pr`を使用します (他のデバッガーを使用しなければならない場合、そのオブジェクトを引数として、関数 `debug_print` を呼び出します)。`pr`コマンドはファイル `.gdbinit`で定義されていて、(コアダンプではなく) 実行中のプロセスにたいしてデバッグするときだけ機能します。

  Lisp エラーで Emacs を停止させて GDB に戻るには、`Fsignal`にブレークポイントを置きます。

  実行中の Lisp 関数のバックトレースを得るにはは、GDB コマンド `xbacktrace`をタイプします。

  ファイル `.gdbinit`では、データ型や Lisp オブジェクトの内容を調べるための、他のコマンドも定義されています。これらのコマンドの名前は 'x'で始まります。これらのコマンドは `pr`より低いレベルで動作するので、少し不便になりますが、コアダンプをデバッグしていたり、Emacs が致命的なシグナルを受け取ったときなど、`pr`が機能しないようなときでも、機能するでしょう。

  Emacs のデバッグにたいする、より詳細なアドバイスと、他の有用なテクニックは、Emacs ディストリビューションの、ファイル `etc/DEBUG`で利用可能です。そのファイルには、Emacs が応答しない問題を詳しく調べる手順も含まれています (多くの人はこれを Emacs が "ハング" したとみなしますが、実際はおそらく無限ループにハマっているのでしょう)。

  インストールした Emacs のファイル `etc/DEBUG`を探すには、変数 `data-directory`に保管されたディレクトリー名を使用します。

34.3.5 GNU Emacs へのパッチの送付

GNU Emacs を改善するためにバグ fix を書きたいなら、それはとても助けになります。変更を送るとき、メンテナーがそれらを使うのが簡単になるように、どうか以下のガイドラインにしたがってください。これらのガイドラインにしたがわない場合でも、あなたの情報はまだ有用でしょうが、それを使用するのに余分な作業が必要になります。GNU Emacs の保守は最良の状況でも多くの作業を要

すので、わたしたちを助けるのにあなたがベストをすくさなければ、わたしたちはそれを維持できないのです。

各パッチは、わたしたちがそれを正しく評価するために、簡単な情報をもたなければなりません。これらの情報については以下で説明します。

それらの情報がすべて揃ったら、`M-x submit-emacs-patch`コマンドを使ってパッチを送信してください。このコマンドはそのパッチの件名とパッチファイルの入力を求めます。それからパッチファイルが添付された Message モードのバッファーを作成、表示してそのパッチに関するより詳細な説明、それから以下で求めるようなその他の情報を追加できるようにします。それらが終わったら `C-c C-c` をタイプして、電子メールで開発者にパッチを送信してください。そのメールは`https://debbugs.gnu.org`にある、GNU Bug Tracker に送信されるでしょう。Tracker はバグレポートに行うように、あなたの投稿に番号を割り当てます。通常だと開発者が返信することになるので (さらなる詳細や追加の情報を求めるためかもしれません)、必ず有効な返信用のメールアドレスを含めるようにしてください。

パッチの投稿の一部としてわたしたちが提供して欲しいのは以下の情報です:

- そのパッチがどんな問題を fix するか、またはどんな改善をもたらすのかを説明してください。
  - 既存のバグにたいする fix については、'`bug-gnu-emacs`'リストの関連するディスカッションか、`https://debbugs.gnu.org`の GNU Bug Tracker のバグエントリーに返信するのが最善でしょう。その変更が、なぜバグを fix するのか説明してください。
  - 新しい機能については、その機能と実装についての説明を含めてください。
  - 新しいバグにたいしては、あなたが fix したと思っている問題にたいする、正しいバグレポートを含めてください (Section 34.3.4 [Checklist], page 545 を参照)。それを採用する前に、わたしたちはその変更が正しいと、私たち自身に納得させる必要があります。もしパッチが正しくても問題を再現する方法がなければ、それが fix しようと試みている問題を理解する妨げになるでしょう。
- コードの変更には将来このソースを読む人の理解を助けとなるように、なぜこの変更が必要なのかについて適切なコメントすべてを含めてください。
- 異なる理由にたいする変更を一緒に混ぜないでください。それらを個別に送ってください。

  異なる理由にたいして 2 つの変更を行なった場合、わたしたちをそれを一緒に採用したいとは思わないでしょう。1 つだけを採用したいと思うかもしれないし、それぞれを異なるバージョンの Emacs にインストールしたいと思うかもしれません。それらを合わせて 1 つの diff にして送った場合、それらを区別するために (変更のどの部分がどの目的のためかを理解するために)、余計な作業を行なう必要があります。それを行なう時間がない場合には、わたしたちはそのパッチの組み込みを長期間先延ばしする必要が生じるかもしれません。

  1 つの変更を記述したら、その変更の説明と一緒にそれをすぐに送れば、2 つの変更は混ざることはなくなり、それらを区別する余計な作業なしに、わたしたちはそれぞれを正しく判断することができます。
- 1 つの変更を終えたら、それをすぐに送ってください。ときどき人は多くの変更を累積して、すべて一緒に送るのがわたしたちの助けとなると考えます。上述したように、それは正にあなたにできる最悪のことです。

  変更は個別に送るべきなので、すぐに送ることができるでしょう。これは、その変更が重要なものなら、それをすぐに採用するオプションをわたしたちに与えます。
- パッチそのもの。これは以下のいずれかの方法によって生成できます:
  - Emacs レポジトリを使用している場合、あなたのコピーが (たとえば `git pull`などにより) 最新であることを確認してください。あなたの変更をプライベートのブランチにコミッ

トして、`git format-patch master`を使用することにより、マスターバージョンからパッチを生成できます (パッチの適用が容易になるので、これが推奨する方法)。または以下で述べるように変更をコミットせずに、`git diff`を使用することもできます。

– diff の作成には`diff -u`を使用してください。GNU diff がある場合、C コードの diff の作成には`diff -u -F'^[_a-zA-Z0-9$]\\+ *('`を使用してください。これは変更のある関数名を表示します。

diff を作成する際にはどちらが古いバージョンで、どちらが新しいバージョンか、あいまいになるのを避けてください。どうか diff の第 1 引数に古いバージョン、2 番目の引数に新しいバージョンを指定してください。そして一方のバージョンにたいして、それが古いバージョンなのか、変更した新しいバージョンなのかを示す名前をつけてください。

- あなたの変更にたいする、コミットログのエントリーを記述してください。それにより、わたしたちがそれを記述するために余計な作業をしなくてすみ、あなたが行なった変更をわたしたちが理解する助けにもなります。

  コミットログはその変更の根拠、あなたのパッチが修整を試みる問題を変更後のコードがどのように解決するかを説明して、更に人々がその変更をどこで見つけられるかを示すことが目的です。したがって変更した関数とその理由を具体的に記す必要があります。適切なコミットログメッセージにたいするわたしたちのスタイルと要件については、どうか Emacs ソースツリーにあるファイル`CONTRIBUTE`のセクション “Commit messages” を参照してください。

  どのような種類の情報を記述するかを見るために、最近のコミットにたいするコミットログエントリーも参考にして、わたしたちが使用しているスタイルを学んでください。他のプロジェクトとは異なり、たとえば Texinfo ファイルのようなドキュメントにたいするコミットログも必要です。Section 25.3 [Change Log], page 355、および Section “Change Log Concepts” in *GNU Coding Standards* を参照してください。

- fix を記述するときは、わたしたちが他のシステムを壊すような変更は採用できないということを、念頭に置いてください。あなたの変更が、ほかのタイプのシステムでコンパイルおよび/または使用された場合の影響についてどうか考慮してください。

  一般的には改善となるかもしれないが、そう確信するのは難しいような fix を送る人が、ときどきいます。そのような変更を採用するのは、わたしたちがそれをとても慎重に調べなければならないので、難しくなります。もちろん、その変更が正しい理由の説明は、わたしたちを納得させる助けになります。

  もっとも安全な変更は、特定の機種やシステムで使用されるファイルやファイルの一部にたいする変更です。これらの変更は、新しいバグを他の機種やシステムに作成しないので安全です。

  インストールの安全性が明確な形式でパッチをデザインして、わたしたちの作業量を、良い状態に保つ助けとなってください。

34.4 Emacs 開発への貢献

Emacs は共同制作によるプロジェクトであり、わたしたちは誰でも貢献できことを励行します。

Emacs に貢献する多くの方法があります:

- バグの発見と報告。Section 34.3 [Bugs], page 542 を参照してください。
- Emacs ユーザーメーリングリスト`https://lists.gnu.org/mailman/listinfo/help-gnu-emacs`の質問にたいする回答。
- Wiki (`https://www.emacswiki.org/`) または Emacs source repository(Section 34.3.5 [Sending Patches], page 550 を参照してください) のドキュメントの記述。

- 既存のバグレポートが新しいバージョンの Emacshttps://debbugs.gnu.org/cgi/pkgreport.cgi?which=pkg&data=emacsで fix されているかのチェック。
- 既存バグの fix の報告。
- Emacs ディストリビューションの etc/TODOにリストされた機能の実装、およびパッチの送付。
- 新しい機能の実装、およびパッチの送付。
- Emacs で動作するパッケージの開発と、あなた自身または GNU ELPA(https://elpa.gnu.org/) での公開。
- 新しいプラットフォームへ Emacs をポート。ただしこれは現在一般的ではありません。

Emacs を改善する作業をしたい場合は、emacs-devel@gnu.orgのメンテナーに連絡してください。提案されたプロジェクトを尋ねたり、あなたのアイデアを提案することができます。

Emacs にたいする機能の要望や改善方法についての提案を送る場合には、bug-gnu-emacs@gnu.org が最善の場所です。どのような変更を望むのか、そしてなぜ、どのようにすればそれが Emacs を改善すると思うかを、可能なかぎり明快に説明するようにお願いします。

すでに改善を記述したことがある場合は、それについて教えてください。まだ作業を開始していなければ、作業を開始する前に、emacs-devel@gnu.org に連絡をとるのが有益です。Emacs の他の部分にたいして、あなたの拡張がより適合する方法を提案することが可能かもしれません。

機能を実装するときは、Emacs coding standards に従ってください。Section 34.4.1 [Coding Standards], page 553 を参照してください。くわえて重要な貢献にたいしては FSF への著作権の譲渡が必要になります。Section 34.4.2 [Copyright Assignment], page 554 を参照してください。

Emacs の開発バージョンは、開発者グループにより活発にメンテされている、レポジトリからダウンロードできます。アクセスの詳細については Emacs プロジェクトのページ https://savannah.gnu.org/projects/emacs/を参照してください。

カレントワーキングバージョンにたいしてパッチを記述することは重要です。古いバージョンから開始した場合、パッチは時代遅れかもしれず (そのためメンテナーはそれを適用するのが難しくなります)、Emacs の変更によりあなたのパッチが不必要になっているかもしれません。レポジトリソースからダウンロードした後は、ビルド手順についてファイル INSTALL.REPOを読むべきです (ビルド手順が通常のビルドとは異なります)。

より広範な貢献を望むなら、Emacs ソースツリーにある CONTRIBUTEファイルから Emacs 開発者になる方法に関する情報を確認してください。このファイルは Emacs のすべてのリリースバージョンのソース tar ファイルの一部として配布されており、Emacs on-line source repository (https://git.savannah.gnu.org/cgit/emacs.git/tree/CONTRIBUTE) で確認することもできます。https://savannah.gnu.org/projects/emacs/の手順にしたがって Emacs レポジトリを clone した場合には、このファイルは Emacs ソースツリーのトップディレクトリーにあります。

(望む変更を実装する方法を理解するために)、以下の Emacs ドキュメントを参照してください:

- emacsを参照してください。
- elispを参照してください。
- https://www.gnu.org/software/emacs
- https://www.emacswiki.org/

34.4.1 コーディング規約

貢献されたコードは GNU コーディング規約 (GNU Coding Standards: https://www.gnu.org/prep/standards/) にしたがうべきです。これはシステムの info でも利用可能かもしれません。

これにしたがっていない場合、わたしたちがそれを使えるように、そのコードを fix する誰かを探す必要があるでしょう。

Emacs には追加のスタイルとコーディング規約があります:

- Section “Tips Appendix” in *Emacs Lisp Reference* を参照してください。
- Emacs に含まれる Lisp コードでは、`defadvice`と `with-eval-after-load`の使用を避けてください。
- すべてのソースおよびテキストファイルの、すべての行末の空白文字を削除してください。
- Lisp コード内のスペース文字は、`?` のかわりに`?\s`を使用してください。

34.4.2 著作権の割り当て

FSF(Free Software Foundation) は、GNU Emacs の著作権所有者です。FSF はコンピューターユーザーの自由の促進、およびすべてのフリーソフトウェアーユーザーの権利を守るという、世界的な使命をもつ、非営利団体です。一般的な情報については、ウェブサイト `https://www.fsf.org/` を参照してください。

一般的には、GNU Emacs、および GNU ELPA に格納されているパッケージにたいする些細とは言えない貢献にたいして、わたしたちは著作権を FSF に譲渡することを求めます。この背景にある理由については、`http://www.gnu.org/licenses/why-assign.html`を参照してください。

著作権譲渡はシンプルなプロセスです。多くの国の住民は、これを完全にコンピューター上で行なうことができます。わたしたちは `emacs-devel@gnu.org`メーリングリストにおいて、著作権譲渡を満足するために必要なフォームを送ったり、あなたがもつであろうさまざまな疑問にたいして回答 (または回答をもつ人物を掲示) することで、あなたの着手を助けることができます。

(どうか注意してください: なぜいくつかの GNU プロジェクトは版権譲渡を求めるかについての一般的な議論は、emacs-devel では off-topic です。かわりに gnu-misc-discuss を参照してください)

著作権放棄も可能ですが、好ましいのは譲渡です。著作権放棄者は、著作権譲渡者と同様に、FSF にサインされた書類を送ることが必要になります (単に “this is in the public domain” というだけでは充分ではありません)。著作権放棄は将来の作業には適用できないので、新しい何かを送りたいときは、毎回これを繰り返す必要があります。

わたしたちは、譲渡なしで小さな変更 (大雑把に言うと 15 行以下) を受けとることができます。これはあなたの貢献全体にたいする蓄積的な制限です (たとえば 5 行パッチを 3 つ)。

34.5 GNU Emacs にたいして助けを得る方法

インストール、使用方法、または GNU Emacs の変更について助言を必要とする場合は、それを探す 2 つの方法があります:

- メッセージをメーリングリスト `help-gnu-emacs@gnu.org`に送るか、あなたの要求をニュースグループ `gnu.emacs.help`にポストしてください (このメーリングリストとニュースグループは相互接続しているので、どちらを使っても問題ありません)。
- 手数料を徴収して助言してくれる人物を、service directory (`https://www.fsf.org/resources/service/`) で探してください。

Appendix A GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. https://fsf.org/

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program—to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

   "This License" refers to version 3 of the GNU General Public License.

   "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

   "The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

   To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

   A "covered work" means either the unmodified Program or a work based on the Program.

   To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

   To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

   An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

   The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

   A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

   All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

   You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

   Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

   No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

   You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

   You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

   You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

   a. The work must carry prominent notices stating that you modified it, and giving a relevant date.

   b. The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

   c. You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

   d. If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

   A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

   You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c. Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d. Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e. Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source.

The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

   "Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

   When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

   Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

   a. Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
   b. Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
   c. Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d. Limiting the use for publicity purposes of names of licensors or authors of the material; or

e. Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f. Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance.

However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

    Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

    An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

    You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

    A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

    A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

    Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

    In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

    If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so

available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

    If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

    Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and a brief idea of what it does.
Copyright (C) year name of author

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or (at
your option) any later version.

This program is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see https://www.gnu.org/licenses/.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
program Copyright (C) year name of author
This program comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.
This is free software, and you are welcome to redistribute it
under certain conditions; type ‘show c’ for details.
```

The hypothetical commands ‘`show w`’ and ‘`show c`’ should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see `https://www.gnu.org/licenses/`.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read `https://www.gnu.org/licenses/why-not-lgpl.html`.

Appendix B GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.
`https://fsf.org/`

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released

under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any,

be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their

titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `https://www.gnu.org/licenses/`.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

    "Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

    "CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

    "Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

    An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

    The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
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 no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Appendix C Emacs 呼び出しにたいするコマンドライン引数

Emacs は、Emacs を開始するとき、様々な動作を要求するための、コマンドライン引数をサポートします。これらのコマンド引数の中には、他のエディターとの互換性のためのものや、高度な動作を指定するものがあります。通常の編集にこれらを使用することはお勧めしません (コマンドラインから既存の Emacs ジョブにアクセスする方法は、Section 31.6 [Emacs Server], page 469 を参照してください)。

'-'で始まる引数はオプションで、'*+linenum*'もオプションです。Emacs は、スタートアップ時に指定されたファイルを visit します。コマンドラインで最後に指定されたファイルがカレントバッファーになり、それ以外のファイルも別のバッファーで visit されます。ほとんどのプログラムと同様に、特別な引数 '--'は、それ以降のすべての引数が、(名前が '-'で始まっていても) オプションではなくファイル名であることを指定します。

Emacs のコマンドオプションは、X ウィンドウで Emacs が使用するウィンドウのサイズや位置、カラーなど、多くのことを指定できます。多くはありませんが、ファイルの Lisp 関数をバッチモードで実行するなど、高度な使い方をサポートするためのオプションもあります。このチャプターの各セクションでは、利用可能なオプションを、その目的に合わせて整理して説明します。

オプションを記述するには 2 つの方法があります。それは 1 つの '-'で始まる短い形式と、'--'で始まる長い形式です。たとえば、短い形式が '`-d`'で、それに対応する長い形式は '`--display`'です。

'--'の長い形式は、覚えるのが簡単ですが、多くタイプしなければなりません。しかしオプション名全体を綴る必要はありません。あいまいさのない省略形で充分です。長いオプションが引数を要求する場合に、オプション名とそのオプションにたいする引数を区切るには、スペースとイコール記号の両方を使用することができます。したがって '`--display`'オプションにたいしては、'`--display sugar-bombs:0.0`'と '`--display=sugar-bombs:0.0`'の、どちらでも記述することができます。関連性が明確になるのでイコール記号を推奨します。以下の表では、常にイコール記号で記すことにします。

ほとんどのオプションは、Emacs を初期化する方法や、Emacs セッションのパラメーターを指定します。これらを初期化オプション *(initial options)* と呼びます。多くはありませんが、ライブラリーのロードや、Lisp 関数の呼び出しなど、動作を指定するオプションもあります。これらを動作オプション *(action options)* と呼びます。これらとファイル名を合わせたものを、動作引数 *(action arguments)* と呼びます。動作引数は、変数 `command-line-args`に、文字列のリストとして保管されます (実際のところ、Emacs スタートアップ時には、コマンドラインで渡されたすべての引数が、`command-line-args`に含まれていますが、初期化を行なう間に、初期化引数は処理されるごとにこのリストから削除され、動作引数だけが残ります)。

C.1 動作引数

以下は動作引数の表です:

'*file*'
'`--file=`*file*'
'`--find-file=`*file*'
'`--visit=`*file*'

指定された *file*を visit します。Section 15.2 [Visiting], page 146 を参照してください。

Emacs スタートアップ時、1 つのウィンドウでスタートアップバッファーを表示し、*file*を visit するバッファーを別のウィンドウで表示します (Chapter 17 [Windows], page 186 を参照してください)。複数のファイル引数を与えた場合、コマンドラインの最後に指定されたファイルが表示され、他のファイルも visit されますが、表示はされません。

スタートアップバッファーが無効 (Section 3.1 [Entering Emacs], page 15 を参照) の場合、ファイル引数が 1 つのときは 1 つのウィンドウで *file*を visit するバッファーを表示し、ファイル引数が 2 つのときは、Emacs はファイルを 2 つの別のウィンドウに表示します。ファイル引数が 3 つ以上の場合、Emacs は最後に指定されたファイルを 1 つのウィンドウに表示して、別のウィンドウにその他すべてのファイルを表示するバッファーメニュー (Section 16.5 [Several Buffers], page 180 を参照) を表示します。このバッファーメニューの使用を抑制するには、変数 `inhibit-startup-buffer-menu` を `t`に変更してください。

‘`+`*linenum* *file*’
: 指定された *file*を visit し、行番号 *linenum*に移動します。

‘`+`*linenum*`:`*columnnum* *file*’
: 指定された *file*を visit し、行番号 *linenum*に移動して、列番号 *columnnum*にポイントを置きます。

‘`-l` *file*’
‘`--load=`*file*’
: 関数 `load`で、*file*という名前の Lisp ライブラリーをロードします。*file*が絶対ファイル名でない場合、Emacs は最初にカレントディレクトリーを探して、次に `load-path` (Section 24.8 [Lisp Libraries], page 327 を参照してください) にリストされたディレクトリーを探します。

  警告**:** 前のコマンドライン引数に visit されたファイルがある場合、カレントディレクトリーは最後に visit されたファイルのディレクトリーになります。

‘`-L` *dir*’
‘`--directory=`*dir*’
: 変数 `load-path`にリストされたディレクトリーの前に、ディレクトリー *dir*を追加します。複数の ‘`-L`’オプションを指定した場合、Emacs はその順番を保持します。たとえば ‘`-L /foo -L /bar`’を使用すると、`load-path`は `("/foo" "/bar" ...)`となります。*dir*が ‘`:`’で始まる場合、Emacs は ‘`:`’を削除した残りを、`load-path`にリストされたディレクトリーの (前ではなく) 後ろに追加します (MS Windows では ‘`:`’のかわりに、`path-separator`の値 ‘`;`’を使用します)。

‘`-f` *function*’
‘`--funcall=`*function*’
: Lisp 関数 *function*を呼び出します。それがインタラクティブな関数 (コマンド) の場合、同じ関数をキーシーケンスで呼び出したときのように、引数を対話的に読み取ります。そうでない場合は、その関数を引数なしで呼び出します。

‘`--eval=`*expression*’
‘`--execute=`*expression*’
: Lisp 式 *expression*を評価します。

‘`--insert=`*file*’
: *file*の内容を、コマンドライン引数が処理されるときにカレントのバッファーに挿入します。これは通常、`*scratch*`バッファー (Section 24.10 [Lisp Interaction], page 330 を参照してください) ですが、コマンドラインの前の引数がファイルを visit したりバッファーを切り替えるときは、別のバッファーになるでしょう。このコマンドライン引数の効果は、`M-x insert-file`が行なうのと同様です (Section 15.12 [Misc File Ops], page 167 を参照してください)。

‘--kill’ 確認なしで Emacs を終了します。

‘--help’ 利用可能なすべてのオプションがリストされた Usage メッセージをプリントしてから、正常終了します。

‘--version’
Emacs のバージョンをプリントしてから、正常終了します。

‘--fingerprint’
コンパイルされた Emacs のバージョンを一意に識別するために使用される “fingerprint(指紋)” をプリントします。

C.2 初期化オプション

初期化オプションは、Emacs セッションにたいするパラメーターを指定します。このセクションでは、より一般的な初期化オプションを説明します。いくつかのオプションは、厳密には以降のセクションで説明する X ウィンドウシステムに関連したオプションです。

いくつかの初期化オプションは、初期化ファイルのロードに影響します。Emacs は通常、`site-start.el`が存在する場合は最初にそれをロードし、ユーザーの初期化ファイルが存在すれば次にそれをロードして、デフォルト初期化ファイル `default.el`が存在すれば最後にそれをロードします (Section 33.4 [Init File], page 528 を参照してください)。それらのファイルのロードを抑制したり、それらのファイルを別のファイルで置き換えるオプションもあります。

‘-chdir *directory*’
‘--chdir=*directory*’
他のことを行なう前にまず、カレントディレクトリーを *directory*に変更します。これは Emacs が停止したのと同じディレクトリーで開始されるように、主に X でのセッション管理に使用されます。これによりデスクトップの保存と復元が簡単になります。

‘-t *device*’
‘--terminal=*device*’
端末の入出力に *device*をデバイスとして使用します。このオプションは暗に ‘--no-window-system’を含みます。

‘-d *display*’
‘--display=*display*’
Emacs 初期フレームを開くために、X ウィンドウシステムと *display*という名前のディスプレイを使用します。詳細は、Section C.5 [Display X], page 585 を参照してください。

‘-nw’
‘--no-window-system’
環境変数 `DISPLAY`がセットされていても、ウィンドウシステムと直接やりとりしません。これは Emacs が、Emacs のすべての表示と出力のために、開始された端末を使用することを意味します。

‘-batch’
‘--batch’ Emacs をバッチモード *(batch mode)* で実行します。バッチモードは、シェルスクリプトや make ファイルから、Emacs Lisp で記述されたプログラムを実行するために使用されます。Lisp プログラムを呼び出すには、‘-batch’オプションと併せて、1 つ以上の ‘-l’、‘-f’、‘--eval’を使用します (Section C.1 [Action Arguments], page 574 を

参照してください)。使用例は、Section C.3 [Command Example], page 579 を参照してください。

バッチモードでは、Emacs は編集されるテキストを表示せず、`C-z`や`C-c`のような、標準の端末割り込みが通常の効果をもちます。通常はエコーエリアにメッセージをプリントする Emacs 関数は、かわりに標準出力ストリーム (`stdout`)、または標準エラーストリーム (`stderr`) にメッセージをプリントします (正確に言うと、`prin1`、`princ`、`print`のような関数は `stdout`にプリントし、`message`や `error`は `stderr`にプリントします)。通常はミニバッファーからキーボード入力を読み取る関数は、かわりに端末の標準入力ストリーム (`stdin`) から入力を受け取ります。

'`--batch`'は暗に '`-q`'(初期化ファイルをロードしません) を含みますが、それでも `site-start.el`はロードされます。これはすべてのコマンドオプションを処理した後に、Emacs を終了します。それに加えて、自動保存が明示的に要求された場合を除き、自動保存を無効にします。また自動保存が要求されていない場合は、ファイルの保存での `fsync`システムコールを省略します。

'`--batch`'で Emacs を実行中に発生したエラーでは、Emacs Lisp のバックトレースがプリントされます。この動作を無効にするには `backtrace-on-error-noninteractive`に `nil`をセットしてください。

'`--script` *file*'
: '`--batch`'と同様に、Emacs をバッチモードで実行してから、*file*の Lisp コードを読み込み実行します。

このオプションは通常、Emacs を実行する実行可能スクリプトの中で使用します。これは以下のテキストを最初の行に記述します:

```
#!/usr/bin/emacs --script
```

これは '`--script`'で Emacs を呼び出し、スクリプトファイル名は *file*です。それから Emacs Lisp は最初の行の '`#!`'をコメント区切りとして扱います。

'`-x`'
: このオプションは実行可能なスクリプトファイルにのみ使用でき、以下のように呼び出す必要があります:

```
#!/usr/bin/emacs -x
```

'`--script`'と同様ですが (`--quick`のように)init ファイルのロードを抑止します。また通常のコマンドラインとして使うことはできません (ロードするスクリプトを指定できないので)。更にスクリプトの終端に達すると Emacs を exit するとともに、最後のフォームの値が最後の値が数値ならその値、そうでなければ常に 0 をそのスクリプトの exit 値として使用します。

'`--no-build-details`'
: Emacs 実行可能形式からシステム名やビルド日時などの詳細を取り除き、そのビルドがより決定論的になります。このスイッチにより `system-name`のようなコマンドは `nil`をリターンするようになるので、これは通常 (またはインタラクティブ) の使用を意図していません。

'`-q`'
'`--no-init-file`'
: 初期化ファイルをロードしません (Section 33.4 [Init File], page 528 を参照してください)。Emacs がこのオプションで呼び出された場合、Customize 機能は、保存のオプションを受け付けません (Section 33.1 [Easy Customization], page 499 を参照してください)。このオプションは `site-start.el`のロードは無効にしません。

‘--no-site-file’
‘-nsl’ site-start.elをロードしません (Section 33.4 [Init File], page 528 を参照してください)。‘-Q’オプションもこれを行ないますが、‘-q’のような他のオプションはこれを行ないません。

‘--no-site-lisp’
load-pathに site-lispディレクトリーを含めません (Section 33.4 [Init File], page 528 を参照してください)。‘-Q’オプションもこれを行ないます。

‘--init-directory’
Emacs の init ファイルを探す際に使用するディレクトリーを指定します。

‘--no-splash’
スタートアップ画面を表示しません。初期化ファイルで変数 inhibit-startup-screenを非 nilにセットしても、この効果を得ることができます (Section 3.1 [Entering Emacs], page 15 を参照してください)。

‘--no-x-resources’
X リソースをロードしません。初期化ファイルで変数 inhibit-x-resourcesを tにセットしても、この効果を得ることができます (Section D.1 [Resources], page 591 を参照してください)。

‘-Q’
‘--quick’ 最小のカスタマイズで Emacs を開始します。これは ‘-q’、‘--no-site-file’、‘--no-site-lisp’、‘--no-x-resources’、‘--no-splash’を一緒に使用したのと同様です。

‘-daemon’
‘--daemon[=*name*]’
‘--bg-daemon[=*name*]’
‘--fg-daemon[=*name*]’
Emacs をデーモンとして開始します。これはフレームをオープンせずに、Emacs サーバーを開始します。その後に emacsclientコマンドを使用して、編集のために Emacs に接続できます (オプションでサーバーに明示的に *name*を指定できる。これを行った際には、emacsclient呼び出し時に--socket-nameオプションで、同じ *name*を指定する必要があるだろう。Section 31.6.3 [emacsclient Options], page 472 を参照されたい)。デーモンとしての Emacs の使用についての情報は、Section 31.6 [Emacs Server], page 469 を参照してください。“バックグラウンド” のデーモンは端末から切り離され、バックグラウンドで実行されます (‘--daemon’は ‘--bg-daemon’のエイリアス)。

‘--no-desktop’
保存されたデスクトップをリロードしません。Section 31.10 [Saving Emacs Sessions], page 482 を参照してください。

‘-u *user*’
‘--user=*user*’
あなたの初期化ファイルのかわりに、*user*の初期化ファイルをロードします。[1]

[1] このオプションは MS-Windows では効果がありません。

‘`--debug-init`’
: init ファイルのエラーにたいして、Emacs Lisp デバッガーを有効にします。Section “Entering the Debugger on an Error” in *The GNU Emacs Lisp Reference Manual* を参照してください。

‘`--module-assertions`’
: 動的ロードモジュールを処理する際に、高価な正当性チェックを有効にします。これは、作成したモジュールがモジュール API 仕様に従っているか検証したいモジュール作者を意図したものです。モジュール関連の assert が発生した場合、Emacs は abort します。Section “Writing Dynamically-Loaded Modules” in *The GNU Emacs Lisp Reference Manual* を参照してください。

‘`--dump-file=`*`file`*’
: *file*という名前のファイルからダンプされた Emacs 状態をロードします。デフォルトではインストールされた Emacs はダンプ状態を、Emacs のインストールがアーキテクチャー依存のファイルを配置するディレクトリー内で *`emacs`*`.pdmp`という名前のファイルから探します。変数 `exec-directory`はそのディレクトリーの名前を保持します。*emacs*は Emacs 実行可能ファイルの名前であり、通常は単なる `emacs`です (インストールせずにビルドしたディレクトリー `src`から Emacs を呼び出す際には実行可能形式のディレクトリーでダンプファイルを探す)。違う場所にダンプファイルのリネームや移動を行なった場合には、ファイルを探す場所を Emacs に指示するために、このオプションを使用できます。

C.3 コマンド引数の例

以下は、引数とオプションを指定して Emacs を使用する例です。ここでは、ロードすることにより、C プログラムを visit したカレントバッファーにたいして、何か便利な操作を行なう、`hack-c.el`という名前の Lisp プログラムファイルがあるとしましょう。

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

これは `foo.c`を visit してから、`hack-c.el`(これは visit されたファイルに何らかの変更を行ないます) をロードし、`foo.c`を保存 (`save-buffer`は `C-x C-s`にバインドされた関数であることに注意してください) した後に、(‘`--batch`’オプションなので)Emacs を終了してシェルに戻ります。‘`--batch`’は、出力を `log`にリダイレクトしても問題がないことが保証されます。なぜなら Emacs は作業するためのディスプレイ端末をもたないと想定されるからです。

C.4 環境変数

環境 *(environment)* とは、オペレーティングシステムの機能です。これは値と名前をもつ変数のコレクションからなります。それぞれの変数は環境変数 *(environment variable)* と呼ばれます。環境変数名は大文字小文字を区別し、慣習では大文字だけを使用します。値はすべてテキスト文字列です。

何が環境を便利にしているかというと、それはサブプロセスが親プロセスから自動的に環境を継承できることです。これはログインシェルで環境変数をセットして、実行するすべて (Emacs を含む) のプログラムが、それを参照できることを意味します。Emacs のサブプロセス (シェル、コンパイラー、バージョンコントロールシステムなど) も、Emacs から環境を継承します。

Emacs の中では、コマンド `M-x getenv`は環境変数の名前を読み取って、その値をエコーエリアにプリントします。`M-x setenv`は Emacs 環境で変数をセットし、`C-u M-x setenv`は変数を削除します (‘`$`’による環境変数の置き換えは、ファイル名にたいする値と同様に機能します。[File Names

with $], page 146 を参照してください)。変数 initial-environmentは、Emacs により継承された初期環境を保管します。

Emacs の外で環境変数をセットする方法は、オペレーティングシステム、特に使用しているシェルに依存します。たとえば以下は Bash を使用して、環境変数 ORGANIZATIONに 'not very much'をセットする例です:

```
export ORGANIZATION="not very much"
```

以下は csh または tcsh でこれを行なう方法の例です:

```
setenv ORGANIZATION "not very much"
```

Emacs が X ウィンドウシステムを使用している場合、X を制御するさまざまな環境変数は、Emacs も同様に制御します。詳細は、X のドキュメントを参照してください。

C.4.1 一般的な変数

以下は、Emacs で特別な意味をもつ環境変数の、アルファベット順のリストです。これらの変数のほとんどは、他のプログラムでも使用されます。Emacs はこれらの環境変数がセットされていることを要求しませんが、セットされている場合はその値を使用します。

CDPATH 相対ディレクトリーを指定したときに、指定したディレクトリーを検索するために、cd コマンドにより使用されます。これは変数 shell-file-nameの初期化に使用されます (Section 31.5.1 [Single Shell], page 457 を参照)。

COLORTERM
この変数を値 'truecolor'にセットすると、terminfo データベースが未インストールでもテキストモードディスプレイで 24 ビット true カラーを使用するよう Emacs に指示します。Emacs は欠落している terminfo 情報のかわりに RGB 値で true カラーを要求するためにビルトインコマンドを使用します。

DBUS_SESSION_BUS_ADDRESS
D-Bus サポートつきで Emacs がコンパイルされている場合、D-Bus により使用されます。通常これを変更する必要はありません。'unix:path=/dev/null'のようなダミーアドレスをセットすることにより、D-Bus セッションバスとの接続を抑止し、同様に D-Bus セッションバスがまだ実行されていないときは自動的に開始します。

EMACSDATA
Emacs に含まれるアーキテクチャーに独立なファイルのディレクトリーです。これは変数 data-directoryの初期化に使用されます。

EMACSDOC ドキュメント文字列ファイル (documentation string file) にたいするディレクトリーで、Lisp 変数 doc-directoryの初期化に使用されます。

EMACSLOADPATH
Emacs Lisp ファイルを検索する、コロンで区切られたディレクトリーのリスト [2] です。セットされている場合は、変数 load-pathの初期値です (Section 24.8 [Lisp Libraries], page 327 を参照してください)。空の要素は、load-pathのデフォルト値を意味します。たとえば 'EMACSLOADPATH="/tmp:"'を使用すると、デフォルトの load-pathの前に/tmpを追加します。リストの中間に空の要素を指定するには、'EMACSLOADPATH="/tmp::/foo"'のように、2 つのコロンを続けます。

[2] これ以降、"コロンで区切られたディレクトリーのリスト" という場合は、Unix および GNU/Linux システムの場合を指します。MS-DOS および MS-Windows では、DOS/Windows ファイル名にはドライブ文字の後のコロンが含まれるので、かわりにセミコロンでディレクトリーが区切られます。

EMACSPATH
: 実行可能ファイルを検索するための、コロンで区切られたディレクトリーのリストです。セットされている場合、Emacs は変数 exec-path(Section 31.5 [Shell], page 457 を参照してください) を初期化するときに、PATH(以下参照) に加えてこれをを使用します。

EMAIL
: メールアドレスです。Lisp 変数 user-mail-addressを初期化するために使用されます。Emacs のメールインターフェースは、送信メッセージの 'From'ヘッダーにこれを使用します (Section 29.2 [Mail Headers], page 421 を参照してください)。

ESHELL
: シェルモードで、環境変数 SHELLをオーバーライドするために使用されます (Section 31.5.2 [Interactive Shell], page 459 を参照してください)。

HISTFILE
: ログインしている間のシェルコマンドが保存されるファイルの名前です。この変数のデフォルトは、Bash を使用している場合は~/.bash_history、ksh のときは~/.sh_history、それ以外では~/.historyです。

HOME
: ディレクトリーツリー上で、あなたのファイルがある場所です。チルダ (~) で始まるファイル名の展開に使用されます。セットする場合には絶対ファイル名をセットする必要があります (相対ファイル名をセットすると Emacs は Emacs が開始されたディレクトリーから相対的なディレクトリーと解釈するが、この機能の使用は推奨しない)。未セットなら HOMEのデフォルトは LOGNAME、USER、またはユーザー ID により与えられるユーザーのホームディレクトリー、いずれも失敗したら/になります。MS-DOS では、この変数のデフォルトは Emacs が開始されたディレクトリーで、そのディレクトリー名が '/bin'で終わる場合、'/bin'は取り除かれます。Windows では、HOMEのデフォルト値は、ユーザープロファイルディレクトリーの Application Dataサブディレクトリー (つまり通常は C:/Documents and Settings/*username*/Application Data。*username*はユーザー名) ですが、後方互換のため、C:/に.emacsがある場合は、かわりに C:/がデフォルト値になります。

HOSTNAME
: Emacs が実行されているホストの名前です。

INFOPATH
: コロンで区切られた、Info ファイルを検索するディレクトリーのリストです。

LC_ALL
LC_COLLATE
LC_CTYPE
LC_MESSAGES
LC_MONETARY
LC_NUMERIC
LC_TIME
LANG
: そのユーザーの優先される locale です。locale には 6 つのカテゴリーがあり、それぞれ環境変数で指定されます。ソートは LC_COLLATE、文字エンコーディングは LC_CTYPE、システムメッセージは LC_MESSAGES、通貨形式は LC_MONETARY、数字は LC_NUMERIC、日時は LC_TIMEで locale を指定します。これらの変数の 1 つがセットされていない場合、そのカテゴリーのデフォルトは環境変数 LANGの値、LANGがセットされていない場合は 'C' locale になります。しかし LC_ALLが指定された場合、これは他のすべての locale 環境変数のセッティングをオーバーライドします。

  MS-Windows と macOS では、環境で LANGがまだセットされていない場合、Emacs はシステムワイドなデフォルト言語にもとづき、LANGをセットします。MS-Windows のいくつかのバージョンでは、これはコントロールパネルの 'Regional Settings'で

セットでき、macOS では System Preference の “Language and Region” でセットできます。

LC_CTYPEカテゴリーの値は、デフォルトの言語環境 (language environment) とコーディングシステム (coding system) を選択するために、`locale-language-names`、`locale-charset-language-names`、`locale-preferred-coding-systems`のエントリーにたいしてマッチされます。Section 19.2 [Language Environments], page 218 を参照してください。

`LOGNAME` ユーザーのログイン名です。`USER`も参照してください。

`MAIL` システムのメール inbox 名です。

`NAME` あなたの現実世界での名前です。これは変数`user-full-name`を初期化するために使用されます (Section 29.2 [Mail Headers], page 421 を参照してください)。

`NNTPSERVER`
ニュースサーバーの名前です。Gnus パッケージで使用されます。

`ORGANIZATION`
あなたが属する組織の名前です。Gnus パッケージで、ポストの ‘`Organization:`’ヘッダーをセットするために使用されます。

`PATH` コロンで区切られた、実行可能ファイルを含むディレクトリーのリストです。これは変数`exec-path`を初期化するために使用されます (Section 31.5 [Shell], page 457 を参照してください)。

`PWD` セットされている場合、これは Emacs が開始されたときのデフォルトディレクトリーになるはずです。

`REPLYTO` セットされている場合は、変数`mail-default-reply-to`の初期値になります (Section 29.2 [Mail Headers], page 421 を参照してください)。

`SAVEDIR` 新しいアーティクルがデフォルトで保存されるディレクトリーの名前です。Gnus パッケージにより使用されます。

`SHELL` Emacs の中からプログラムのパースや実行に使用されるインタープリターの名前です。これは変数`shell-file-name`の初期化に使用されます (Section 31.5.1 [Single Shell], page 457 を参照)。

`SMTPSERVER`
送信メールサーバーの名前です。これは変数`smtpmail-smtp-server`を初期化するために使用されます (Section 29.4.1 [Mail Sending], page 423 を参照してください)。

`TERM` Emacs が使用する端末タイプです。Emacs がバッチモードで実行されていないとき、この変数はセットされていなければなりません。MS-DOS では、デフォルトは ‘`internal`’ で、これはその機種固有のディスプレイを扱う、ビルトインの端末エミュレーションを指定します。

`TERMCAP` `TERM`で指定された端末をどのようにプログラムするかを記述する、termcap ライブラリーファイルの名前です。デフォルトは`/etc/termcap`です。

`TMPDIR`
`TMP`
`TEMP` これらの環境変数は、変数`temporary-file-directory`を初期化するために使用されます。これは一時ファイルを置く場所を指定します (Section 15.3.2 [Backup], page 152

を参照してください)。Emacs は最初に TMPDIRの使用を試みます。これがセットされていない場合、通常 Emacs は/tmpにフォールバックします。しかし MS-Windows と MS-DOS では、かわりに TMP、次に TEMP、最後に c:/tempにフォールバックします。

TZ これはデフォルトのタイムゾーン (もしかしたらサマータイムの情報も) を指定します。Section “Time Zone Rules” in *The GNU Emacs Lisp Reference Manual* を参照してください。MS-DOS では、Emacs を開始したときの環境で TZがセットされていない場合、Emacs は DOS が返す国コードにたいして、適切なデフォルト値を定義します。MS-Windows では、Emacs は TZを使用しません。

USER ユーザーのログイン名です。LOGNAMEも参照してください。MS-DOS では、デフォルトは ‘root’です。

VERSION_CONTROL
変数 version-controlの初期化に使用されます (Section 15.3.2.1 [Backup Names], page 152 を参照してください)。

C.4.2 その他の変数

これらの変数は、特定の設定だけにたいして使用されます:

COMSPEC MS-DOS と MS-Windows で、バッチファイルとシェルの内部コマンドを呼び出すときに使用する、コマンドインタープリターの名前です。MS-DOS では、環境変数 SHELL のデフォルト値のためにも使用されます。

NAME MS-DOS で、この変数は変数 USERのデフォルト値として使用されます。

EMACSTEST
MS-DOS で、内部端末エミュレーターの操作ログに使用するファイルを指定します。この機能はバグレポートを送るときに便利です。

EMACSCOLORS
MS-DOS で、スクリーンカラーを指定します。Emacs は開始時の数瞬、デフォルトカラーを表示するので、それらの指定にこの方法が便利です。

この変数の値は 2 文字のエンコーディングで指定し、それぞれデフォルトフェイスにたいして、1 文字目はフォアグラウンドカラー、2 文字目はバックグラウンドカラーを指定します。それぞれの文字は、標準の PC テキストモードディスプレイのカラーを記述する、16 進コードを指定します。たとえば light gray の背景色で blue のテキストを得たい場合は、‘EMACSCOLORS=17’を指定します。ここで 1 はカラー blue、7 はカラー light gray を指定しています。

PC ディスプレイは通常、8 色のバックグラウンドカラーだけをサポートします。しかしバックグラウンドカラーにたいして、16 色すべてを使用できる DOS ディスプレイにモードを切り替えるので、実際には 4 ビットのバックグラウンドカラーすべてが使用されます。

PRELOAD_WINSOCK
MS-Windows でこの変数をセットしている場合、Emacs は開始時に、最初に要求されたときまで待たずに、ネットワークライブラリーのロードと初期化を行ないます。

WAYLAND_DISPLAY
(--with-pgtkでビルドされた Pgtk Emacs は Wayland でネイティブに実行可能です。WAYLAND_DISPLAYはコンポジターにたいする接続を指定します。

`emacs_dir`

MS-Windows では、`emacs_dir`は特別な環境変数で、これは Emacs がインストールされたディレクトリーのフルパスを指します。Emacs が標準的なディレクトリー構造にインストールされた場合、この変数の値は自動的に計算されます。他の環境変数とは異なり、これは開始時に Emacs によりオーバーライドされるので、標準的なインストールをした場合には、あなた自身がこの変数のセッティングを使用することは、あまりないでしょう。`EMACSLOADPATH`のような他の環境変数をセットするとき、絶対パスをハードコーディングするより、`emacs_dir`を使う法が便利だと気づくでしょう。これにより複数バージョンの Emacs が、同じ環境変数のセッティングを共有することが可能になり、環境やレジストリーのセッティングを変更せずに、Emacs のインストールディレクトリーを変更できるようになります。

C.4.3 MS-Windows のシステムレジストリー

MS-Windows では環境変数 `emacs_dir`、`EMACSLOADPATH`、`EMACSDATA`、`EMACSPATH`、`EMACSDOC`、`SHELL`、`TERM`、`HOME`、`LANG`、`PRELOAD_WINSOCK`をシステムレジストリーのセクション `HKEY_CURRENT_USER`か `HKEY_LOCAL_MACHINE`にある`/Software/GNU/Emacs`キー配下でもセットできます。Emacs の起動時には環境のチェックと同じように、システムレジストリーでもこれらの変数をチェックします。

これらの変数の値を決定するために、Emacs は次の手順にしたがいます。最初に環境がチェックされます。そこで変数が見つからない場合、Emacs は`/Software/GNU/Emacs`配下で変数名でレジストリーキーを探します。最初にレジストリーの `HKEY_CURRENT_USER`セクション、そこで見つからなければ、`HKEY_LOCAL_MACHINE`セクションを探します。それでもなお Emacs が値を決定できない場合には、コンパイルされているデフォルトが使用されます。

レジストリーの設定はシステム全体にグローバルな影響を与えることに注意してください。これらの設定はそのシステム上で実行される Emacs セッションすべてに影響します。したがって異なるバージョンの Emacs を実行したり、インストール済みとまだインストールされていない Emacs 実行可能ファイルの両方を使用する場合や、新しいバージョンの Emacs をビルドする場合には、このレジストリー設定によってそれらすべてが同じディレクトリーを使用することとなり、恐らくそれはあなたの期待した動作ではない筈です。この理由により、レジストリーでこれらの変数を設定しないことをお勧めします。レジストリー内でこれらの設定を行っている場合には、削除することをお勧めします。

MS-Windows 用の Emacs インストールプログラム `addpm.exe`を実行すると、その `addpm.exe`が付属していたインストール済みの Emacs のバージョンに相応しい値となるように `emacs_dir`、`EMACSLOADPATH`、`EMACSDATA`、`EMACSPATH`、`EMACSDOC`、`SHELL`、`TERM`の既存のレジストリー設定を更新します。`addpm.exe`は存在しないレジストリー設定の作成は行わず、恐らくほとんどが古いインストール済み Emacs から継承された既存の設定だけを、新たにインストールしたバージョンの Emacs と互換性があるように更新することに注意してください。最近のバージョンの Emacs ではインストール時に `addpm.exe`を実行する必要はなくなったので、古いバージョンからのアップグレードを行う際に古いバージョンの設定を何らかの理由によりレジストリーから削除できない場合のみこれを行うことをお勧めします。

上記の環境変数に加えて、X リソース (Appendix D [X Resources], page 591 を参照) を指定するために`/Software/GNU/Emacs`のレジストリーキーにセッティングを追加することもできます。`.Xdefaults`ファイル内で指定可能なほとんどのセッティングを、そのレジストリキーからセットできます。

C.5 ディスプレイ名の指定

環境変数 DISPLAYは、Emacs を含むすべての X クライアントに、ウィンドウをどこに表示するかを指定します。通常の状況では、この変数の値は、X サーバーを開始してローカルでジョブを実行したとき、デフォルトにセットされます。ディスプレイを自分で指定することもできます。これを行なう理由の 1 つは、他のシステムにログインして、そこで Emacs を実行し、ウィンドウはローカル端末のウィンドウに表示させたい場合です。

DISPLAYは、‘*host*:*display*.*screen*’という構文をもちます。ここで *host*は X ウィンドウシステムのサーバー機のホスト名、*display*は、あなたのサーバー (X サーバー) を、同一機種上の他のサーバーと区別するために任意に割り当てられる数字、そして *screen*フィールドは、X サーバーが複数の端末スクリーンを制御することを可能にします。ピリオドと *screen*フィールドはオプションです。*screen*が含まれる場合、通常は 0 です。

たとえば、あなたのホスト名が ‘glasperle’で、あなたのサーバーが設定にリストされた最初の (もしかしたら唯一の) サーバーの場合、DISPLAYは ‘glasperle:0.0’になります。

Emacs を実行するとき、変数 DISPLAYを変更するか、オプション ‘-d *display*’または ‘--display=*display*’で、明示的にディスプレイを指定できます。以下は例です:

```
emacs --display=glasperle:0 &
```

‘-nw’オプションで、X ウィンドウシステムの使用を抑止できます。その場合、Emacs はディスプレイとして制御テキスト端末を使用します。Section C.2 [Initial Options], page 576 を参照してください。

セキュリティーの規制により、リモートシステムのプログラムが、ローカルシステムで表示を行なうのを禁ずることがあります。この場合、Emacs は以下のようなメッセージを出力します:

```
Xlib:  connection to "glasperle:0.0" refused by server
```

ローカル機で xhostコマンドを使用することにより、リモート機からアクセスする権限を与えれば、この問題を解決できるでしょう。

C.6 フォント指定オプション

デフォルトのフォントを指定するために、コマンドラインオプション ‘-fn *font*’(または ‘--font’。これは ‘-fn’のエイリアスです) を使用できます。

‘-fn *font*’
‘--font=*font*’
: *font*をデフォルトフォントとして使用します。

コマンドラインでフォント名を Emacs に渡す場合、フォント名にシェルが特別に扱う文字 (たとえばスペース) が含まれる場合は、クォーテーションマークで囲んでクォートする必要があるでしょう。たとえば:

```
emacs -fn "DejaVu Sans Mono-12"
```

フォント名やデフォルトフォントを指定する別の方法についての詳細は、Section 18.8 [Fonts], page 201 を参照してください。

C.7 ウィンドウカラーオプション

Emacs ディスプレイのさまざまな場所で使用するカラーを指定するために、以下のコマンドラインオプションを使用できます。カラーは、カラーネーム (color names) か、RGB トリプレット (RGB triplets) のどちらかを使用して、指定します。

‘-fg *color*’
‘--foreground-color=*color*’
フォアグラウンドカラーを指定します。defaultフェイス (Section 11.8 [Faces], page 82 を参照してください) で指定されたカラーをオーバーライドします。

‘-bg *color*’
‘--background-color=*color*’
バックグラウンドカラーを指定します。defaultフェイスで指定されたカラーをオーバーライドします。

‘-bd *color*’
‘--border-color=*color*’
X ウィンドウのボーダーカラーを指定します。Emacs が GTK+サポートつきでンパイルされいてる場合、効果はありません。

‘-cr *color*’
‘--cursor-color=*color*’
ポイントがある位置を示す Emacs のカーソルカラーを指定します。

‘-ms *color*’
‘--mouse-color=*color*’
Emacs ウィンドウにマウスがあるときの、マウスカーソルカラーを指定します。

‘-r’
‘-rv’
‘--reverse-video’
フォアグラウンドカラーとバックグラウンドカラーを入れ替えて、反転表示します。

‘--color=*mode*’
Emacs をテキスト端末で実行するときの、カラーサポートモード *(color support mode)* をセットします。その文字端末の terminfoデータベースまたは termcapで示される、いくつかのサポートされたカラーをオーバーライドします。パラメーター *mode*には、以下の 1 つを指定できます:

‘never’
‘no’ 端末の能力がカラーをサポートすると指定していても、カラーを使用しません。

‘default’
‘auto’ --colorを指定しないときと同じです。Emacs は開始時に端末がカラーをサポートするか検知して、もしサポートされていれば、カラーディスプレイをオンに切り替えます。

‘always’
‘yes’
‘ansi8’ 無条件にカラーサポートをオンに切り替えて、標準的な 8 カラーにたいする ANSI エスケープシーケンスによるカラーコマンドを使用します。

‘*num*’ *num*個のカラーにたいするカラーモードを使用します。*num*が −1 の場合、カラーサポートをオフに切り替えます (‘never’と同じです)。0 の場合、この端末にたいするデフォルトのカラーサポートを使用します (‘auto’と同じです)。そうでない場合、*num*個のカラーにたいする適切な標準モードを使

用します。端末の能力により、*num*の値に応じて、Emacs は 8 色、16 色、88 色、256 色のカラーモードに切り替えることができるでしょう。*num*カラーをサポートするモードが存在しない場合、Emacs はあたかも *num*に 0 が指定された場合のように、その端末のデフォルトのカラーサポートを使用します。

*mode*が省略された場合のデフォルトは、*ansi8*です。

たとえば coral のマウスカーソルと、slate blue のテキストカーソルを使用するには、以下のように入力します:

```
emacs -ms coral -cr 'slate blue' &
```

‘`-rv`’オプションを指定するか、X リソースの ‘`reverseVideo`’で、フォアグラウンドとバックグラウンドのカラーを反転できます。

‘`-fg`’、‘`-bg`’、‘`-rv`’オプションの機能は、テキスト端末でも、グラフィカルなディスプレイと同様です。

C.8 ウィンドウのサイズと位置にたいするオプション

以下は、Emacs の初期フレームのサイズと位置を指定する、コマンドラインオプションのリストです:

‘`-g` *`widthxheight`*[{`+-`}*`xoffset`*{`+-`}*`yoffset`*]]’
‘`--geometry=`*`widthxheight`*[{`+-`}*`xoffset`*{`+-`}*`yoffset`*]]’
: サイズ *width*と *height*(文字の列数と行数)、および位置 *xoffset*と *yoffset*(ピクセル) を指定します。パラメーター *width*と *height*はすべてのフレームに適用されますが、*xoffset* と *yoffset*は初期フレームだけに適用されます。

‘`-fs`’
‘`--fullscreen`’
: スクリーンのサイズになるように、幅と高さを指定します。通常は、ウィンドウマネージャーによる装飾は表示されません (Emacs を起動した後に、`F11 toggle-frame-fullscreen`を使用して、この状態を切り替えることができます)。

‘`-mm`’
‘`--maximized`’
: Emacs のフレームが最大化されるよう指定します。これは通常、そのフレームがウィンドウマネージャーによる装飾をもつことを意味します (Emacs を起動した後に、`M-F10 toggle-frame-maximized`を使用して、この状態を切り替えることができます)。

‘`-fh`’
‘`--fullheight`’
: 高さがスクリーンの高さになるように指定します。

‘`-fw`’
‘`--fullwidth`’
: 幅がスクリーンと同じ幅になるように指定します。

‘`--geometry`’オプションでは、{`+-`}は、プラス記号かマイナス記号のどちらかを意味します。*xoffset* の前のプラス記号はスクリーンの左端からの距離、マイナス記号は右端からの距離を意味します。*yoffset*の前のプラス記号はスクリーンの上端からの距離、マイナス記号は下端からの距離を意味します。*xoffset*と *yoffset*の値自体に正または負の値を指定できますが、それはこれらの意味を変更するものではなく、方向だけを変更します。

Emacs は、`xterm`がジオメトリーを解釈するのと同じ単位を使用します。*width*と *height*は文字で数えられるので、大きなフォントは、小さなフォントより大きなフレームを作成します (プロポーショナルフォントを指定した場合、Emacs は幅の単位として、そのプロポーショナルフォントの最大幅を使用します)。*xoffset*と *yoffset*はピクセルで数えます。

ジオメトリー指定で、すべてのフィールドを指定する必要はありません。*xoffset*と *yoffset*の両方を省略した場合、ウィンドウマネージャーが Emacs フレームをどこに配置するかを決定するか、もしかしたらマウスでその場所を指定できるかもしれません。たとえば '`164x55`'は、ウィンドウの幅が 164(通常のウィンドウを横に 2 つ並べられる幅) で、55 行分の高さを指定します。

デフォルトのフレーム幅は 80 文字、デフォルトの高さは 35 行から 40 行です。幅と高さの、どちらか一方、または両方を省略できます。ジオメトリーが整数で開始される場合、Emacs はそれを幅と解釈します。ジオメトリーの開始が '`x`'でその後に整数が続く場合、Emacs はそれを高さと解釈します。したがって '`81`'は幅だけを指定し、'`x45`'は高さだけを指定します。

ジオメトリーを '`+`'か '`-`'で開始した場合、それはオフセットを示し、幅と高さの両方のサイズが省略されたことを意味します。したがって '`-3`'は *xoffset*だけを指定します (オフセットを 1 つだけ与えたとき、それは常に *xoffset*になります)。'`+3-3`'は *xoffset*と *yoffset*の両方を指定し、フレームをスクリーンの左下に配置します。

X リソースファイル (Section D.1 [Resources], page 591 を参照してください) で、任意またはすべてのフィールドにたいするデフォルトを指定できます。それらにたいしてフィールドを選択して、'`--geometry`'オプションでオーバーライドできます。

モードラインとエコーエリアはフレームの 2 行を占めるので、初期のテキストウィンドウの高さは、ジオメトリーで指定した高さより、2 つ少なくなります。非 X-toolkit バージョンの Emacs では、指定した数から、メニューバーの占める 1 行が引かれます。しかし X-toolkit バージョンでは、メニューバーは付加的で、指定された高さにたいして数えられません。ツールバーがある場合も、それは付加的です。

メニューバーやツールバーを有効または無効にすることにより、通常のテキストが利用できるスペースが変わります。したがって、Emacs がツールバーつき (デフォルト) で開始しされ、ツールバーがあることを前提にジオメトリー指定を処理して、それを初期化ファイルでツールバーを無効にしていた場合、あなたが指定したのと異なるフレームジオメトリーになるでしょう。ツールバーがないサイズを得るには、X リソースで "no tool bar" を指定します (Section D.2 [Table of Resources], page 592 を参照してください)。そうすれば、Emacs は指定されたジオメトリーを処理するときに、ツールバーがないことを知ることができます。

'`--fullscreen`'、'`--maximized`'、'`--fullwidth`'、'`--fullheight`'のどれか 1 つを使用するとき、フレームが本当に最大化またはフルスクリーンに見えるようにするために、変数`frame-resize-pixelwise`を非`nil`値にすることを要求するウィンドウマネージャーがいくつかあります。

位置にたいする、プログラム指定とユーザー指定の両方を無視するようにできるオプションをもつウィンドウマネージャーがいくつかあります。これらのオプションがセットされている場合、Emacs はウィンドウを正しく配置するのに失敗します。

C.9 内枠ボーダーと外枠ボーダー

1 つの Emacs フレームは内枠ボーダー (internal border) と、外枠ボーダー (outer border) をもちます。内枠ボーダーはフレームのテキスト部分の周囲にある、バックグラウンドカラーによる追加の縁取り (extra strip) です。内枠ボーダーは Emacs 自身が描画します。外枠ボーダーはフレームのツールバーとメニューバーの外側にあり、X により描画されます。ウィンドウマネージャーにより描画さ

れる外部ボーダー (external border) も存在します。外部ボーダーのサイズは、Emacs 内からセットすることはできません。

‘`-ib` *width*’
‘`--internal-border=`*width*’
: 内枠ボーダー (フレームのテキストエリア周辺) の幅に、ピクセル単位で *width*を指定します。

‘`-bw` *width*’
‘`--border-width=`*width*’
: 外枠ボーダーの幅に、ピクセル単位で *width*を指定します。

フレームのサイズを指定するとき、ボーダーは含まれません。フレームの位置は、外枠ボーダーの縁端から測られます。

ピクセル幅 *n*の内枠ボーダーを指定するには、‘`-ib` *n*’オプションを使用します。デフォルトは 1 です。外枠ボーダーの幅を指定するには、‘`-bw` *n*’を使用します (ウィンドウマネージャーはその指定に注意を払わないかもしれませんが)。外枠ボーダーのデフォルト幅は 2 です。

C.10 フレームタイトル

Emacsh フレームは常に指定されたタイトルをもち、それはウィンドウ装飾とアイコンに、フレームの名前として表示されます。デフォルトのタイトルは‘*invocation-name*`@`*machine*’という形式 (フレームが 1 つだけのとき) か、選択されたウィンドウのバッファー名 (複数のフレームがある場合) になります。

コマンドラインオプションで、Emacs の初期フレームにデフォルト以外のタイトルを指定できます:

‘`-T` *title*’
‘`--title=`*title*’
: Emacs 初期フレームのタイトルとして *title*を指定します。

‘`--name`’オプション (Section D.1 [Resources], page 591 を参照してください) でも、Emacs 初期フレームのタイトルを指定できます。

C.11 アイコン

‘`-iconic`’
‘`--iconic`’
: Emacs をアイコン化した状態で開始します。

‘`-nbi`’
‘`--no-bitmap-icon`’
: Emacs アイコンの使用を無効にします。

ほとんどのウィンドウマネージャーでは、Emacs フレームをアイコン化 (または “最小化”) して、見えなくすることができます。いくつかのウィンドウマネージャーでは、他のウィンドウマネージャーがアイコン化されたウィンドウを完全に見えなくするのにたいして、アイコン化されたウィンドウを小さなアイコンでおきかえるものもあります。‘`-iconic`’オプションは、フレームをすぐに表示するのではなく、アイコン化された状態で実行を開始するよう、Emacs に指示します。テキストフレームはアイコン化の解除 (または “最小化の解除”) をするまで表示されません。

デフォルトでは、Emacs は Emacs ロゴを含むアイコンを使用します。Gnome のようなデスクトップ環境では、このアイコンは他のコンテキスト、たとえば Emacs フレームに切り替えるときに

も表示されます。‘-nbi’または‘--no-bitmap-icon’オプションは、使用するアイコンの種類をウィンドウマネージャーに選択させるよう、Emacs に指示します。通常これはフレームのタイトルを含んだ、ただの矩形です。

C.12 その他のディスプレイオプション

‘--parent-id *id*’
: XEmbed プロトコルを通じて、*id*を親 X ウィンドウ ID とする、クライアント X ウィンドウとして、Emacs を開始します。現在のところ、このオプションは開発者にとって有用です。

‘-vb’
‘--vertical-scroll-bars’
: 垂直スクロールバーを有効にします。

‘-lsp *pixels*’
‘--line-spacing=*pixels*’
: 行間の追加のスペース *pixels*を、ピクセルで指定します。

‘-nbc’
‘--no-blinking-cursor’
: グラフィカルなディスプレイで、点滅カーソルを無効にします。

‘-D’
‘--basic-display’
: メニューバー、ツールバー、スクロールバー、ツールチップを無効、font-lock-mode と点滅カーソルをオフに切り替えます。これは表示問題のデバッグするテストケースを簡単にするのに有用です。

‘--xrm’オプション (Section D.1 [Resources], page 591 を参照してください) は、追加の X リソース値を指定します。

Appendix D X のオプションとリソース

X を使用するプログラムにたいして通常行なうように、X リソースを使用して Emacs の X に関連する外観をカスタマイズできます。

Emacs が GTK+サポートつきでコンパイルされている場合、メニューバー、スクロールバー、ダイアログボックスなどの、さまざまなグラフィカルウィジェットの外観は GTK+ resources. Emacs が GTK+のサポートなしでビルドされた場合、これらのウィジェットの外観は追加の X リソースにより決定されます。

MS-Windows では、システムレジストリー (Section C.4.3 [MS-Windows Registry], page 584 を参照してください) を使用して、いくつかの外観を同じようにカスタマイズできます。

D.1 X リソース

X ウィンドウシステムの下で実行されているプログラムは、クラスとリソースの階層の下にユーザーオプションを組織化します。これらのオプションにたいして、*X リソースファイル (X resource file)* でデフォルト値を指定できます。X リソースファイルは、通常`~/.Xdefaults`または`~/.Xresources`という名前です。このファイルの内容を変更しても、変更は即座に効果をもちません。これは X サーバーが独自にリソースのリストを保持しているからです。これを更新するには、たとえば‘`xrdb ~/.Xdefaults`’のように、コマンド `xrdb`を使用します。

一般的に X リソースを通じて指定されたセッティングは、特に初期フレームのパラメーター (Section 18.11 [Frame Parameters], page 206 を参照) に関して、Emacs の init ファイル (Section 33.4 [Init File], page 528 を参照) のセッティングをオーバーライドします。

(MS-Windows システムは X リソースファイルをサポートしません。そのようなシステムでは、Emacs は Windows レジストリーの中から X リソースを探します。最初にキー ‘`HKEY_CURRENT_USER\SOFTWARE\GNU\Emacs`’の下を探します。これはカレントユーザーだけに影響し、システムワイドなセッティングをオーバーライドします。次にキー‘`HKEY_LOCAL_MACHINE\SOFTWARE\GNU\Emacs`’の下を探します。これはシステムのすべてのユーザーに影響します。メニューとスクロールバーは、MS-Windows のネイティブなウィジェットなので、Display Control Panel のシステムワイドなセッティングからしかカスタマイズできません。以下で説明するように、コマンドラインオプション‘`-xrm`’を使用して、リソースをセットすることもできます。)

X リソースファイルの各行は、1 つのオプション、または関連するオプションのコレクションにたいして値を指定します。ファイル内で行が出現する順番に意味はありません。各リソース指定は、プログラム名 *(program name)* とリソース名 *(resource name)* から構成されます。これらの名前の大文字小文字は区別されます。以下は例です:

```
emacs.cursorColor: dark green
```

プログラム名は、そのリソースが適用される実行可能ファイルの名前です。Emacs では通常、‘`emacs`’です。Emacs の実行可能ファイル名の如何にかかわらずに、Emacs のすべてのインスタンスに適用される定義を指定するには、‘`Emacs`’を使用します。

リソース名はプログラムセッティングの名前です。たとえば Emacs は、‘`cursorColor`’リソースを、テキストカーソルのカラーを制御するリソースと認識します。

リソースは、名前のついたクラスにグループ化されます。たとえば ‘`Foreground`’クラスには、リソース ‘`cursorColor`’、‘`foreground`’、‘`pointerColor`’が含まれます (Section D.2 [Table of Resources], page 592 を参照してください)。以下のように、リソース名を使用するかわりに、クラス内のすべてのリソースにたいしてデフォルト値を指定するのに、クラス名を使用できます:

```
emacs.Foreground: dark green
```

変数 `inhibit-x-resources`を非 `nil`値にセットした場合、Emacs は X リソースを処理しません。コマンドラインオプション '`-Q`' (または '`--quick`') で Emacs を呼び出した場合、`inhibit-x-resources`は自動的に `t`にセットされます (Section C.2 [Initial Options], page 576 を参照してください)。

D.2 Emacs にたいする X リソースの表

以下のテーブルは、Emacs が認識する X リソース名です。いくつかのリソースは、Emacs が種々の X ツールキット (GTK+、Lucid、... 等) とともにコンパイルされていなければ効果がないことに注意してください (以下ではそのような場合は個々に示す)。

`alpha` (class `Alpha`)
: フレームの透過度を決定するフレームパラメーター '`alpha`'をセットします (Section "Frame Parameters" in *The Emacs Lisp Reference Manual* を参照)。

`alphaBackground` (class `AlphaBackground`)
: バックグラウンドの透過度を決定するフレームパラメーター '`alpha-background`'をセットします (Section "Frame Parameters" in *The Emacs Lisp Reference Manual* を参照)。

`background` (class `Background`)
: バックグラウンドカラーです (Section 11.9 [Colors], page 83 を参照してください)。

`bitmapIcon` (class `BitmapIcon`)
: '`on`'のときは Emacs アイコンを表示し、'`off`'のときは表示しないように、ウィンドウマネージャーに指示します。アイコンの説明は、Section C.11 [Icons X], page 589 を参照してください。

`cursorBlink` (class `CursorBlink`)
: 開始時に、このリソースの値が '`off`'、'`false`'、'`0`'の場合、Emacs は Blink Cursor モードを無効にします (Section 11.21 [Cursor Display], page 99 を参照してください)。

`cursorColor` (class `Foreground`)
: テキストカーソルのカラーです。Emacs 開始時にこのリソースが指定されている場合、Emacs はその値を `cursor`フェイス (Section 11.8 [Faces], page 82 を参照してください) のバックグラウンドカラーとしてセットします。

`font` (class `Font`)
: `default`フェイスにたいするフォント名です (Section 18.8 [Fonts], page 201 を参照してください)。フォントセット名 (Section 19.14 [Fontsets], page 232 を参照してください) を指定することもできます。

`fontBackend` (class `FontBackend`)
: フォントの描画に使用するバックエンドを、優先順にカンマ区切りで記述したリストです。たとえば値 '`x,xft`'は、Emacs に X コアフォントドライバー (X core font driver) を使用してフォントを描画し、それに失敗したときは Xft フォントドライバー (Xft font driver) にフォールバックするよう指示します。このリソースは通常、セットしないでおくべきです。その場合、Emacs は利用可能なすべてのバックエンドの使用を試みます。

`foreground` (class `Foreground`)
: テキストにたいするデフォルトのフォアグラウンドカラーです。

`fullscreen` (class `Fullscreen`)
: 望ましいフルスクリーンサイズを指定します。値は `fullboth`、`maximized`、`fullwidth`、`fullheight`のうちどれか 1 つを指定でき、これらはコマンドラインオプションの‘`-fs`’、‘`-mm`’、‘`-fw`’、‘`-fh`’に相当します (Section C.8 [Window Size X], page 587 を参照してください)。これは初期フレームだけに適用されることに注意してください。

`geometry` (class `Geometry`)
: ウィンドウのサイズと位置です。値にはコマンドラインオプション‘`-g`’または‘`--geometry`’と同じ形式で、サイズと位置を指定します。

  サイズは Emacs セッションのすべてのフレームに適用されますが、位置は初期フレーム (特定のフレームにたいするリソースの場合は、そのフレーム) だけに適用されます。

  このリソースを、‘`emacs*geometry`’のように指定しないように気をつけてください。これは Emacs のメインのフレームと同じように、個別のメニューにも影響します。

`lineSpacing` (class `LineSpacing`)
: 行間の追加のスペース (ピクセル) です。

`menuBar` (class `MenuBar`)
: このリソースの値が‘`off`’、‘`false`’、‘`0`’の場合、Emacs は開始時に Menu Bar モードを無効にします (Section 18.15 [Menu Bars], page 209 を参照してください)。

`pointerColor` (class `Foreground`)
: マウスカーソルのカラーです。多くのグラフィカルなデスクトップ環境では、この方法で Emacs がマウスカーソルを変更するのを許さないので、効果はありません。

`title` (class `Title`)
: 初期 Emacs フレームのタイトルバーに表示する名前です。

`toolBar` (class `ToolBar`)
: このリソースの値が‘`off`’、‘`false`’、‘`0`’の場合、Emacs は開始時に Tool Bar モードを無効にします (Section 18.16 [Tool Bars], page 209 を参照してください)。

`tabBar` (class `TabBar`)
: このリソースの値が‘`on`’、‘`yes`’、‘`1`’なら、Emacs は開始時に Tab Bar モードを無効にします (Section 18.17 [Tab Bars], page 210 を参照してください)。

`useXIM` (class `UseXIM`)
: ‘`false`’または‘`off`’の場合、X 入力メソッド (XIM: X input methods) の使用を無効にします。これは Emacs が XIM サポートつきでビルドされた場合だけ関係があります。X のクライアント/サーバーのリンクが遅いときは、XIM をオフにするのが便利かもしれません。

`inputStyle` (class `InputStyle`)
: これは X インプットメソッドが生成するプレビューテキストをどのように表示するかを制御するリソースです。値には以下のいずれかを指定できます:

  ‘`callback`’
  : カレントバッファーにプレビューテキストの内容を表示します。

  ‘`offthespot`’
  : Emacs が提供するディスプレイの別領域内ににプレビューテキストの内容を表示します。

‘overthespot’
カレントウィンドウのポイント位置にウィンドウをポップアップして、その内部にプレビューテキストを表示します。

‘none’ インプットメソッド自体に表示方法を任せます。通常は‘overthespot’と等価ですが、より多くのインプットメソッドで機能するかもしれません。

‘native’ インプットメソッドの処理にツールキットを使用します。現在のところGTKでのみ実装されています。

‘root’ プレビューテキストの表示にたいして、インプットメソッド固有のディスプレイ位置を使用します。

synchronizeResize (class SynchronizeResize)
‘off’か‘false’なら、Emacsはフレームのリサイズにたいする応答において、ディスプレイの再表示が完了した際にウィンドウマネージャーへの通知を試みません。それ以外の値ならウィンドウマネージャーはリサイズされたフレームの内容が更新されるまでフレームの描画を延期します。これによってまだ描画されていないフレームのブランク領域が表示されることを防ぐのです。‘extended’にセットされていれば、別種のフレーム同期プロトコルを有効にします。これはEmacsがデフォルトで使用するプロトコルではサポートされていませんが、一部のコンポジット型ウィンドウマネージャーがサポートされているかもしれないプロトコルのことを指し、互換性のあるコンポジット型ウィンドウマネージャー使用中はモニターのリフレッシュレートとEmacsの表示を同期させます。

verticalScrollBars (class ScrollBars)
‘left’の場合はフレームの左側、‘right’の場合は右側、‘off’のときはスクロールバーをもちません (Section 18.12 [Scroll Bars], page 206 を参照)。

特定のEmacsフェイスのカスタマイズに、Xリソースを使用することもできます (Section 11.8 [Faces], page 82 を参照してください)。たとえば‘face.attributeForeground’をセットするのは、フェイス *face*の‘foreground’属性をカスタマイズするのと等価です。しかし、わたしたちはXリソースを使用してフェイスのカスタマイズするかわりに、Emacs内でフェイスをカスタマイズする方法を推奨します。Section 33.1.5 [Face Customization], page 503 を参照してください。

D.3 GTK+リソース

EmacsがGTK+サポートつきでコンパイルされた場合、EmacsのGTK+ウィジェット (例: メニュー、ダイアログ、ツールバー、スクロールバー) をカスタマイズする一番簡単な方法は、たとえばGNOME theme selectorなどで適切なGTK+テーマを選択する方法です。

GTK+バージョン2では、Emacsで使用されるGTK+ウィジェットの外観をカスタマイズするために、*GTK+*リソースも使用できます。これらのリソースは、ファイル~/.emacs.d/gtkrc(Emacs固有のGTK+リソース用)、またはファイル~/.gtkrc-2.0(一般的なGTK+リソース用) のどちらかで指定されます。GTK+は、GNOMEでGConfを実行するとき~/.gtkrc-2.0を無視するように思われるため、わたしたちは~/.emacs.d/gtkrcの使用を推奨します。しかし~/.emacs.d/gtkrcによるカスタマイズをオーバーライドするGTK+テーマがあることに注意してください。これにたいして、わたしたちができることはありません。GTK+リソースは、EmacsのGTK+ウィジェットと関係のない側面、たとえばEmacsのメインウィンドウのフォントやカラーにたいしては影響しません。これらは通常のXリソースに管理されます (Section D.1 [Resources], page 591 を参照してください)。

以下のセクションでは、Emacs にたいして GTK+リソースをカスタマイズする方法を説明します。GTK+リソースについての詳細は、`https://developer-old.gnome.org/gtk2/stable/gtk2-Resource-Files.html`の GTK+ API ドキュメントを参照してください。

GTK+バージョン 3 では、GTK+リソースは完全に異なるシステムに置き換えられました。GTK+ウィジェットの外観は今や CSS-like なスタイルのファイル、すなわち GTK+をインストールしたディレクトリーの `gtk-3.0/gtk.css`、そしてローカルなスタイルセッティングにたいしては`~/.themes/theme/gtk-3.0/gtk.css`で決定されます (*theme*はカレント GTK+テーマ名)。したがってこのセクションの GTK+リソースの説明は GTK+3 に適合しません。GTK+3 スタイリングシステムについての詳細は、`https://developer-old.gnome.org/gtk3/3.0/GtkCssProvider.html`を参照してください。

D.3.1 GTK+ Resource Basics

GTK+2 リソースファイル (通常は`~/.emacs.d/gtkrc`) で、リソースをセットする一番簡単な方法は、単に変数に値を割り当てる方法です。たとえば以下の行を利ソースファイルに置くことにより、すべての GTK+ウィジェットのフォントが '`courier-12`'に変更されます:

```
gtk-font-name = "courier 12"
```

この場合、フォント名は Fontconfig スタイルのフォント名や XLFD ではなく、GTK フォントパターン (*Pango* フォント名とも呼ばれます) で記述しなければなりません。Section 18.8 [Fonts], page 201 を参照してください。

ウィジェットをカスタマイズするには、最初にスタイルを定義して、そのスタイルをウィジェットに適用します。以下はメニューにたいしてフォントをセットする例です (文字 '`#`'はコメントを示します):

```
# Define the style 'my_style'.
style "my_style"
{
  font_name = "helvetica bold 14"
}

# Specify that widget type '*emacs-menuitem*' uses 'my_style'.
widget "*emacs-menuitem*" style "my_style"
```

この例のウィジェット名にはワイルドカードが含まれるので、このスタイルは '`*emacs-menuitem*`' にマッチするすべてのウィジェットに適用されます。ウィジェットは、外側のウィジェットから内側のウィジェットへと、それが含まれる方法により、名前がつけられます。以下は Emacs メニューバーに '`my_style`'を指定して適用する例です:

```
widget "Emacs.pane.menubar.*" style "my_style"
```

以下はスクロールバーを部分的に変更する方法を示す、より複雑な例です:

```
style "scroll"
{
  fg[NORMAL] = "red"      # Arrow color.
  bg[NORMAL] = "yellow"   # Thumb and background around arrow.
  bg[ACTIVE] = "blue"     # Trough color.
  bg[PRELIGHT] = "white" # Thumb color when the mouse is over it.
}

widget "*verticalScrollBar*" style "scroll"
```

D.3.2 GTK+ウィジェット名

GTK+ウィジェットはウィジェット名 *(widget name)* とウィジェットクラス *(widget class)* により指定されます。ウィジェット名は特定のウィジェット (たとえば '`emacs-menuitem`') を参照し、ウィ

ジェットクラスは似通ったウィジェット (たとえば‘GtkMenuItem’) のコレクションを参照します。ウィジェットは常にクラスをもちますが、名前をもつ必要はありません。

絶対名 *(absolute names)* とは、一連のウィジェット名またはウィジェットクラスで、他のウィジェットに埋め込まれたウィジェットの階層に対応します。たとえば、topという名前の GtkWindowが、boxという名前の GtkVBoxを含み、それがさらに menubarという名前の GtkMenuBarを含む場合、メニューバーウィジェットの完全クラス名は GtkWindow.GtkVBox.GtkMenuBarで、完全ウィジェット名は top.box.menubarになります。

GTK+リソースファイルには、ウィジェットの外観を指定する 2 種類のコマンドを含めることができます:

`widget` クラス名または単にクラスにもとづいてウィジェットのスタイルを指定します。

`widget_class`
クラス名にもとづいてウィジェットのスタイルを指定します。

widgetコマンドを使用する例は、前のセクションを参照してください。widget_classコマンドも同じように使用されます。ウィジェット名/ウィジェットクラスとスタイルはダブルクォートで囲わなければならないこと、そしてこれらのコマンドは GTK+リソースファイルのトップレベルに記述しなければならないことに注意してください。

前に記したように、ウィジェット名とウィジェットクラスはシェルのワイルドカード構文で指定することができます。‘*’は 0 文字以上にマッチし、‘?’は 1 文字にマッチします。以下はすべてのウィジェットにスタイルを割り当てる例です:

```
widget "*" style "my_style"
```

D.3.3 Emacs での GTK+ウィジェット名

Emacs フレームが使用する GTK+ウィジェットを以下にリストします:

Emacs (class GtkWindow)
pane (class GtkVBox)
menubar (class GtkMenuBar)
[menu item widgets]
[unnamed widget] (class GtkHandleBox)
emacs-toolbar (class GtkToolbar)
[tool bar item widgets]
emacs (class GtkFixed)
verticalScrollBar (class GtkVScrollbar)

Emacs ウィンドウの内容は、emacsウィジェットにより描画されます。複数の Emacs ウィンドウがある場合でも、それぞれのスクロールバーウィジェットの名前は verticalScrollBarであることに注意してください。

たとえば、以下はメニューバーのスタイルをセットする、2 つの異なる方法の例です:

```
widget "Emacs.pane.menubar.*" style "my_style"
widget_class "GtkWindow.GtkVBox.GtkMenuBar.*" style "my_style"
```

GTK+ダイアログにたいしては、Emacs はクラス GtkDialogの、emacs-dialogという名前のウィジェットを使用します。ファイル選択にたいしては、Emacs はクラス GtkFileSelectionの、emacs-filedialogという名前のウィジェットを使用します。

ポップアップメニューとダイアログのウィジェットは独立したウィンドウであり、Emacsウィジェットに含まれていないので、それらの GTK+完全名は ‘Emacs’で始まりません。これらのウィジェットをカスタマイズするには、以下のようにワイルドカードを使用します:

```
widget "*emacs-dialog*" style "my_dialog_style"
widget "*emacs-filedialog* style "my_file_style"
widget "*emacs-menuitem* style "my_menu_style"
```

Emacs のすべてのメニューにスタイルを適用したい場合は、以下を使用します:

```
widget_class "*Menu*" style "my_menu_style"
```

D.3.4 GTK+スタイル

以下は 2 つの GTK+スタイル宣言の例です:

```
pixmap_path "/usr/share/pixmaps:/usr/include/X11/pixmaps"

style "default"
{
  font_name = "helvetica 12"

  bg[NORMAL] = { 0.83, 0.80, 0.73 }
  bg[SELECTED] = { 0.0, 0.55, 0.55 }
  bg[INSENSITIVE] = { 0.77, 0.77, 0.66 }
  bg[ACTIVE] = { 0.0, 0.55, 0.55 }
  bg[PRELIGHT] = { 0.0, 0.55, 0.55 }

  fg[NORMAL] = "black"
  fg[SELECTED] = { 0.9, 0.9, 0.9 }
  fg[ACTIVE] = "black"
  fg[PRELIGHT] = { 0.9, 0.9, 0.9 }

  base[INSENSITIVE] = "#777766"
  text[INSENSITIVE] = { 0.60, 0.65, 0.57 }

  bg_pixmap[NORMAL] = "background.xpm"
  bg_pixmap[INSENSITIVE] = "background.xpm"
  bg_pixmap[ACTIVE] = "background.xpm"
  bg_pixmap[PRELIGHT] = "<none>"

}

style "ruler" = "default"
{
  font_name = "helvetica 8"
}
```

スタイル ‘ruler’は ‘default’を継承します。この方法により、既存のスタイルを元にすることができます。フォントとカラーの構文は以下に記述されています。

この例が示すように、ウィジェットの状態にもとづいてフォアグラウンドとバックグラウンドのカラーに複数の値を指定することが可能です。以下の状態が利用できます:

NORMAL　ウィジェットのフォルトの状態です。

ACTIVE　ウィジェットが何かを行なう準備ができた状態です。これはスクロールバーの溝 (trough) にも適用できます。たとえば bg[ACTIVE] = "red"はスクロールバーの溝を赤にセットします。ボタンの armed(ボタンが押されてまだ離されていない状態) も、この状態です。

`PRELIGHT` マウスポインターがその上にきたとき、ウィジェットが操作可能な状態です — たとえばマウスがスクロールバーのハンドルの上やメニューアイテムの上にきたときの状態です。押されていないボタンの上にマウスがくると、そのボタンはこの状態になります。

`SELECTED` ユーザーにより選択されたデータにたいする状態です。それは選択されたテキストやリストの選択されたアイテムの場合もあります。この状態は Emacs では使用されません。

`INSENSITIVE`
ウィジェットが可視だが通常の方法では操作できない状態 — たとえば押せないボタンや、無効なメニューアイテムなどです。無効なメニューアイテムを黄色で表示するには、`fg[INSENSITIVE] = "yellow"`を使用します。

以下をスタイル宣言に記述できます:

`bg[`*state*`] = `*color*
ウィジェットのバックグラウンドカラーを指定します。編集可能なテキストは`bg`ではなく`base`を使用することに注意してください。

`base[`*state*`] = `*color*
編集可能なテキストのバックグラウンドカラーを指定します。Emacs では、ファイルダイアログのテキストフィールドのバックグラウンドに、このカラーが使用されます。

`bg_pixmap[`*state*`] = "`*pixmap*`"`
(バックグラウンドカラーのかわりに) バックグラウンドイメージを指定します。*pixmap* はイメージファイル名です。GTK+は XPM、XBM、GIF、JPEG、PNG ヲ含む、いくつかのイメージファイルを使用することができます。親ウィジェットと同じイメージをウィジェットに使用したい場合は、‘`<parent>`’を使用します。イメージを使わない場合は、‘`<none>`’を使用します。‘`<none>`’で、親スタイルから継承されたバックグラウンドイメージをキャンセルできます。

ファイル名は絶対ファイル名で指定できません。GTK+は`pixmap_path`で指定されたディレクトリーの pixmap ファイルを探します。`pixmap_path`は、ダブルクォートされたファイルをコロンで区切ったリストで、`gtkrc`ファイルのトップレベルで指定します (スタイル定義の中ではありません。上記の例を参照してください)。

```
pixmap_path "/usr/share/pixmaps:/usr/include/X11/pixmaps"
```

`fg[`*state*`] = `*color*
使用するウィジェットのフォアグラウンドカラーを指定します。これはメニューやボタンのテキストのカラー、スクロールバーの矢印のカラーです。編集可能なテキストにたいしては、`text`を使用します。

`text[`*state*`] = `*color*
編集可能なテキストのカラーです。Emacs では、ファイルダイアログのテキストフィールドに使用されるカラーです。

`font_name = "`*font*`"`
ウィジェット内のテキストのフォントを指定します。*font*は、‘`Sans Italic 10`’のような、GTK(Pango) スタイルのフォント名です。Section 18.8 [Fonts], page 201 を参照してください。名前は大文字小文字を区別しません。

カラーを指定する 3 つの方法があります。それはカラーネーム、RGB トリプレット、GTK スタイル RGB トリプレットです。カラーネームと RGB トリプレットについては、Section 11.9 [Colors], page 83 を参照してください。カラーネームは‘`"red"`’のように、ダブルクォートで囲む必要がありま

す。RGB トリプレットは‘`#ff0000`’のように、ダブルクォートで囲う必要がありません。GTK スタイル RGB トリプレットは`{ r, g, b }`という形式をもち、*r*、*g*、*b*は、0 から 65535 の整数か、0.0 から 1.0 の浮動小数点数です。

Appendix E Emacs 28 アンチニュース

時代に逆らって生きるユーザーのために、以下は Emacs バージョン 28.2 へのダウングレードに関する情報です。Emacs 29.1 機能の不在による結果としての偉大なる単純さを、ぜひ堪能してください。

- 新しいリリースと同じように、Emacs 28 でも Lisp プログラムのネイティブコンパイルのサポートとともにコンパイルが可能です。とはいえ以前のあるバージョンにおいて行われるこの機能の削除にたいする準備として、わたしたちは Emacs に同梱されている全 Lisp ファイルの事前ネイティブコンパイルの機能を削除しました。これにより Emacs のビルドプロセスがより高速になりました。
- tree-sitter ライブラリーとともに Emacs をビルドできなくなったので、プログラムしたい言語用にグラマーのライブラリーを見つけてインストールする必要がなくなりました。同様にサポートされているプログラミング言語それぞれにたいしてただ 1 つのメジャーモードが残るように、tree-sitter ライブラリーにもとづくすべてのモードも削除されました。tree-sitter にサポートされたモードをオンにして、それらがもつパーサーベースのフォント表示やインデント、その他の機能を試みるかどうかの判断は不要です。これは一部の言語とファイルタイプについてはメジャーモードが何も存在しなくなることを意味するため、あなたには自然かつ高性能な選択肢として由緒正しき Fundamental モードが残されました。たとえば Go、Rust、および CMake ファイルにたいするファイル編集用のメジャーモードは最早存在しません。これはシンプルで無駄のない Emacs に向けた新たなマイルストーンです。
- SQLite データベースにアクセスするためのビルトインサポートは削除されました。これまでいつも行ってきたように、SQLite ファイルを (Emacs が十分サポート可能な) 単なるバイナリーファイルとして再び編集できるようになりました。
- Haiku オペレーティングシステムのユーザーへの計らいとして、その OS 上での Emacs のビルドを可能にするコードを削除しました。わたしたちは Haiku ユーザーが彼ら自身の、よりシンプルなエディターを用いてファイル編集をエンジョイしてくれることを期待しています。
- X における XInput2 の入力イベント用のサポートはなくなりました。わたしたちは伝統的な X 入力イベントで十分すぎると考えており、これは時を遡れば確実と言えます。派手な入力メカニズムのサポートが全くもって無駄だということに X ウィンドウシステムのメンテナーが一旦気づいてしまえば、XInput2 も最終的には X から削除されることでしょう。
- “pure GTK” (PGTKとも呼ばれる) 構成による Emacs はサポートされなくなりました。これは Emacs から GTK ツールキットのサポートが完全に削除されることを見越したダウングレードであり、時を遡るにつれて GTK は消滅するというわたしたちの予測にも合致しています。わたしたちは X 上でサポートされる唯一の GUI 構成として独自ウィジェットを備えた純粋な X ビルドだけを残して、その他すべてのツールキットにたいするサポートを削除する計画です。
- コマンドラインオプションの`--init-directory`は削除されました。いずれにせよ別ユーザーの init ファイルを用いた Emacs の初期化というアイデアは馬鹿げてます。
- 単純化とネイティブコンパイルオプションの最終的な削除に則し、configure 時の`--with-native-compilation=aot`オプションを削除しました。これによってどのようにネイティブコンパイルが機能するかが劇的に簡略化されるとともに、Emacs のネイティブコンパイルに関する configure 時の決定が明確になりました。Emacs は事前ロードされていない Lisp パッケージを使う前のみネイティブコードにコンパイルするか、さもなくばネイティブコンパイルを全く使わないかのいずれかです。中途半端で特別な例外などたくさんです。同様の理由により、`native-compile-prune-cache`および `startup-redirect-eln-cache`の機能は Emacs から削除されました。

- Emacs で非常に長い行をもつファイルの編集時に、まともな性能と応答性を発揮させるための特別なコードと機能を削除しました。時を遡るにつれてそのようなファイルはどんどん希少になっていき、それ故に Emacs にそのようなトリッキーなコードすべてを含めるのは不必要な複雑さをもたらすと判断されました。
- Emacs から Eglot と LSP サーバーのサポートが削除されました。時を遡れば、ビルトインのソースコード解析手法でも十分すぎると判断しました。
- イメージのスケーリングや回転を行うコマンドは、タイプするのがより容易になるように`+`、`-`、`r`のような単一のキーに再度バインドされました。それらのキーを誤ってタイプしてしまうリスクについてですが Emacs ユーザー、とりわけ時を遡りどんどん若くなっていく Emacs ユーザーがタイプミスを犯すなど、わたしたちは信じていません。
- 多用されるコマンドのタイプを簡素化するために `C-x 8 . .` を `C-x 8 .`、`C-x 8 = =`は `C-x 8 =` というバインドに戻しました。そもそも時を遡るにしたがって、それらのキーにバインドされるコマンド自体どんどん少なくなっていくので、装飾に富んだ長いキーシーケンスへの需要はなくなることでしょう。
- うっかり`*scratch*`バッファーを kill してしまった場合に Emacs が再作成するのは、Lisp Interaction モードではなく Fundamental モードのバッファーになりました。自分で好きなモードをオンにすることができるのです。過去への Emacs リリースにおけるわたしたちの長期計画では`*scratch*`バッファー再作成の完全な削除を予定しており、これはそれに向けた最初の一歩となります。
- `rlogin`および `rsh`プロトコルにたいするサポートが復活しました。時を遡るにつれてこれらのプロトコルの重要性と人気がますます高まることが予想されるためです。
- Emacs から最終的に Unicode サポートを削除するための準備として、Unicode サポートのバージョンを 14.0 にダウングレードしました。
- フォントサイズのグローバルな変更ができなくなりました。そのようなコマンドがあるのは贅沢です。ない方が余程ましなので過去のある時点において、Emacs から可変サイズのフォントにたいするすべてのサポートは削除されることになるでしょう。
- Emacs 簡素化というわたしたちの果てなき探求にもとづき、コマンド `duplicate-line`および `duplicate-dwim`を削除しました。昔からの友である `M-w`と `C-y`(1 回以上タイプする) で十分な筈です。同じ理由からコマンド `rename-visited-file`もなくなりました。
- プレフィックスキーマップ `C-x 8 e`にバインドされていた Emoji 関連の多くのコマンドを削除しました。`C-x 8 RET`を用いて Emoji シーケンスをタイプする機能で十分であること、そしてユーザーがタイプしたい Emoji シーケンスにたいするコードポイントの知識を要求することによって、ユーザーにより良いサービスが提供できると判断しました。
- 多くの書体 (script) と入力メソッド、とりわけ誰も使わない古い書体を削除しました。同様の理由から、ギリシャ語とウクライナ語に翻訳された Emacs のチュートリアルは理由できなくなくなりました。
- `package.el`で VCS(バージョンコントロールシステム) のレポジトリからソースコードが取得できなくなりました。レポジトリのクローンには、Git のようなコマンドラインツールで十分な筈です。したがって `package-vc-install`コマンド、およびその他の同類コマンドも削除しました。
- Emacs 28.2 ではコンピューターのメモリーとディスクの容量を削減して良好な状態に保つために、その他の多くの機能とファイルが削除されました。

Appendix F Emacsとmacos / GNUstep

このセクションでは、GNU/Linuxまたはその他のオペレーティングシステムで GNUstep ライブラリーとともにビルドされた Emacs、または macOS でネイティブウィンドウシステムのサポートつきでビルドされた Emacs を使用する際の特性を説明します。macOS では、Emacs はウィンドウシステムサポートなし/あり、X11、Cocoa インターフェースのいずれかでビルドできます。このセクションは Cocoa ビルドだけに適合します。バージョン 10.6 より前の macOS はサポートしません。

GNUstep はフリーソフトウェアであり、macOS はそうではありません。非フリーなオペレーティングシステムのため、macOS はすべてのコンピューターユーザーが享受すべき自由を、ユーザーに与えません。これは不当です。あなたの自由のために、わたしたちはフリーなオペレーティングシステムへの切り替えを強く推奨します。

わたしたちは、商業オペレーティングシステム上の GNU Emacs をサポートします。なぜならそれは、この自由の経験が、商業オペレーティングシステムから脱却するよう、ユーザーを奮起するだろうからです。

さまざまな歴史的、技術的な理由により、Emacs は内部的に “Cocoa” や “macOS” ではなく、‘`Nextstep`’という用語を使用します。たとえばこのセクションで説明するほとんどのコマンドや変数は‘`ns-`’で始まりますが、これは‘`Nextstep`’を短縮したものです。NeXTstep は 1980 年代に NeXT Inc. からリリースされたアプリケーションインターフェースで、Cocoa はそれの直系の子孫です。Cocoa とは別に、他にも NeXTstep スタイルのシステムの GNUstep があり、これはフリーソフトウェアです。これを記述している時点で、Emacs の GNUstep サポートはアルファ状態 (Section F.4 [GNUstep Support], page 605 を参照してください) ですが、わたしたちは、将来これを改善したいと望んでいます。

F.1 macOS および GNUstep での Emacs の基本的な使い方

デフォルトでは、キー `Alt`と `Option`は、`Meta`と同じです。Mac の `Cmd`キーは `Super`と同じで、Emacs は他の Mac/GNUstep アプリケーション (Section F.3 [Mac / GNUstep Events], page 604 を参照してください) を模倣するこれらの修飾キーを使用した、一連のキーバインドを提供します。これらのキーバインドは通常の方法で変更できます (Section 33.3 [Key Bindings], page 519 を参照)。修飾キー自体はカスタマイズできます。Section F.2 [Mac / GNUstep Customization], page 603 を参照してください。

`S-mouse-1`は `mouse-3`と同様に、クリックした位置にリージョンを調整します (`mouse-save-then-kill`)。`S-mouse-1`が通常行なうように、デフォルトフェイスを変更するためのポップアップメニュー (Section 11.12 [Text Scale], page 88 を参照してください) は表示しません。この変更は、Emacs が他の Mac/GNUstep アプリケーションと同じように動作させるためです。

メニューを使用してファイルを開いたり保存するときや、`Cmd-o`や `Cmd-S`といったキーバインドを使用する場合、Emacs はファイル名の読み取りにグラフィカルなファイルダイアログを使用します。しかし `C-x C-f`のような標準の Emacs のキーシーケンスを使用する場合、Emacs はミニバッファーを使用してファイル名を読み取ります。

GNUstep では、X-windows 環境においてテキストを X のプライマリー選択 (primary selection) に転送するために、`C-w`や `M-w`のかわりに、`Cmd-c`を使用する必要があります。そうでない場合、Emacs はクリップボード選択を使用します。同様に (`C-y`のかわりに)`Cmd-y`は、kill リングやクリップボードではなく、X のプライマリー選択から yank します。

F.1.1 環境変数の取得

latex や man のような、Emacs の下で実行される多くのプログラムは、環境変数のセッティングに依存します。Emacs がシェルから起動された場合、自動的にこれらの環境変数を継承し、Emacs のサブプロセスもそれらを継承します。しかし Emacs が Finder から起動された場合は、シェルの子プロセスではないので、環境変数はセットされません。これによりサブプロセスの振る舞いが、シェルから実行したときと異なることが起こり得ます。

変数 PATH および MANPATH にたいしては、macOS では PATH をセットするシステムワイドな手法は、`/etc/paths`ファイルと`/etc/paths.d`ディレクトリーを使用することが推奨されています。

F.2 Mac/GNUstep でのカスタマイズ

多くはありませんが、Nextstep ポートに特有のカスタマイズオプションがいくつかあります。たとえば修飾キーやフルスクリーン動作に影響するオプションです。そのようなオプションをすべて閲覧するには、`M-x customize-group RET ns RET`を使用します。

F.2.1 修飾キー

以下の変数は実際の修飾キーの挙動を制御します:

`ns-alternate-modifier`
`ns-right-alternate-modifier`
: 左および右の `Option`キーまたは `Alt`キー。

`ns-command-modifier`
`ns-right-command-modifier`
: 左および右の `Command`キー。

`ns-control-modifier`
`ns-right-control-modifier`
: 左および右の `Control`キー。

`ns-function-modifier`
: `Function`キー (fn キー)。

各変数の値は目的にたいしてキーを記述するシンボル、通常のキーとともに使用された際の修飾を記述する (`:ordinary` *symbol* `:function` *symbol* `:mouse` *symbol*)という形式のリスト、ファンクションキー (矢印キーのように文字を生成しない)、マウスクリックのいずれかです。

*symbol*が `control`、`meta`、`alt`、`super`、`hyper`のいずれかなら、それを表す Emacs 修飾を記述します。*symbol*が `none`なら Emacs はそのキーを使用せず標準的な挙動のままとなります。たとえば macOS の `Option`キーなら追加の文字の合成に使用されます。

`ns-right-alternate-modifier`のような右手側のキーにたいする変数は、対応する左手側のキーと同じ振る舞いの使用を意味する `left`にもセットできます。

F.2.2 フレーム変数

`ns-use-proxy-icon`
: この変数は titlebar に proxy アイコンを表示するかどうかを指定します。

`ns-confirm-quit`
: この変数は quit 時にグラフィカルな confirmation ダイアログを表示するかどうかを指定します。

`ns-auto-hide-menu-bar`
: この変数は Emacs フレーム選択時に macOS のメニューバーを隠すかどうかを指定します。非 `nil`ならマウスポインターがスクリーン上端近傍に移動するまでメニューバーは表示されません。

`ns-use-native-fullscreen`
: この変数はネィティブフルスクリーンか非ネイティブフルスクリーンを使用するかどうかを制御します。ネイティブフルスクリーンは macOS 10.7 以降でのみ利用可能です。

F.2.3 macOS のトラックパッドとマウスホイールの変数

これらの変数は macOS 10.7(Lion) 以降にのみ適用されます。

`ns-use-mwheel-acceleration`
: この変数は Emacs がシステムのマウスホイールアクセラレーションを無視するかどうかを制御します。`nil`ならマウスホイールの 'クリック' はそれぞれ正確に 1 つのマウスホイールイベントに対応します。非 `nil`(デフォルト) ならマウスホイールの 'クリック' はそれぞれユーザー入力に依存して 1 つ以上のマウスホイールイベントに対応するかもしれません。

`ns-use-mwheel-momentum`
: この変数はトラックパッドを使用したスクロール時に Emacs がシステムの 'momentum' を無視するかどうかを制御します。非 `nil`(デフォルト) なら高速にスクロールするとユーザーがトラックパッドから指を離した後も少しの間バッファーがスクロールを継続するかもしれません。

`ns-mwheel-line-height`
: この変数はトラックパッドによるスクロールの感度を制御します。Apple のトラックパッドは行単位ではなくピクセル単位でスクロールするので、Emacs がシステムのピクセル値を行に変換します。数値をセットすると、この変数はそれを Emacs が 1 行とみなすピクセル数としてセットします。`nil`か非数値ならデフォルトの行高さを使用します。

  低い数値をセットするとトラックパッドはより高感度に、高い数値では低感度になります。

F.3 macOS および GNUstep でのウィンドウシステムイベント

Nextstep アプリケーションは、X では同等なものがない、特別なイベントを受け取ります。これらは、対応するキーストロークのシーケンスとしてではなく、特別に定義されたキーイベントとして送られます。Emacs では、これらのキーイベントを、通常のキーストロークのように、関数にバインドできます。以下はこのようなイベントのリストです。

`ns-open-file`
: このイベントは、他の Nextstep アプリケーションが Emacs にファイルを開くよう要求したときに発生します。これの典型的な理由としては、ユーザーが Finder アプリケーションでファイルをダブルクリックしたときなどです。デフォルトでは、Emacs はこのイベントにたいして、新しいフレームを開いて、そのフレームでファイルを visit して応答します (`ns-find-file`)。例外として、選択されたバッファーが`*scratch*`バッファーの場合、Emacs は選択されたフレームでファイルを visit します。

  Emacs が `ns-open-file`イベントにたいしてどのように応答するかは、`ns-pop-up-frames`を変更することにより、変えることができます。デフォルト値は '`fresh`'で、これは上で説明したとおりの動作を行ないます。値 `t`は、ファイルを常に新しいフレーム

で visit することを意味します。値 `nil`は、ファイルを常に選択されたフレームで visit することを意味します。

`ns-open-temp-file`
: このイベントは、他のアプリケーションが Emacs に一時ファイルを開くように要求したとき発生します。デフォルトでは、単に `ns-open-file`イベントを生成することにより処理され、結果は上で説明したとおりになります。

`ns-open-file-line`
: ProjectBuilder や gdb のようないくつかのアプリケーションは、特定のファイルだけではなく、そのファイルの特定の行、または一連の行を要求します。Emacs はそのファイルを visit して要求された行をハイライトすることにより、これを処理します (`ns-open-file-select-line`)。

`ns-power-off`
: このイベントは、ユーザーが Emacs を実行中にログアウトしたとき、またはアプリケーションメニューから “Quit Emacs” を選択したとき発生します。デフォルトの動作は、ファイルを visit しているすべてのバッファーを保存します。

`ns-show-prefs`
: このイベントはユーザーがアプリケーションのメニューから “Preferences” を選択したとき発生します。デフォルトではコマンド `customize`にバインドされています。

Emacs はユーザーに、‘`ns-service-`’で始まりサービス名で終わるコマンドを通じて、Nextstep サービスを使用することも可能にします。`M-x ns-service-TAB`とタイプして、これらのコマンドをリストを見ることができます。これらの関数は、マークされたテキストを処理 (結果でそれを置き換える) したり、文字列を引数として結果を文字列で返します。Lisp 関数 `ns-perform-service`を使用して、任意の文字列を任意のサービスに渡して、結果を受けとることもできます。新たに利用可能になったサービスにアクセスするには、Emacs の再起動が必要なことに注意してください。

F.4 GNUstep にたいするサポート

Emacs は GNUstep の下でビルドして実行することができますが、解決すべき問題が残っています。興味のある開発者は、`emacs-devel@gnu.org`に連絡してください。

Appendix G Emacs と Haiku

Haiku とはオペレーティングシステム BeOS の再実装に端を発する Unix 風オペレーティングシステムのことです。

このセクションでは Haiku 固有のウィンドウシステムであるアプリケーションキットとともにビルドされた Emacs 特有の使い方について説明します。ここで説明する特異性はウィンドウサポートのない Haiku での Emacs、および X11 とともにビルドされた Emacs の使い方には当てはまりません。

G.1 Haiku 特有なインストールと使い方

Haiku で Emacs をインストールすると通常の実行可能形式である小文字の名前をもつ `emacs`、そして Haiku 固有のアプリケーションメタデータを含んだ `Emacs`という名前のバイナリー形式という 2 つの実行可能形式が個別にインストールされます。どちらが自分にもっとも適しているかを決めるのはユーザーに任されています。

Tracker から Emacs を起動したり、Emacs を使用して Tracker にファイルをオープンさせたい場合には、`Emacs`という名前のバイナリーを使う必要があります。端末で Emacs を使いたい、Emacs のインスタンスを個別に起動したい、あるいは前述のシステム統合機能を考慮しない場合には、`emacs` という名前のバイナリーを使用してください。

Haiku では Hyper キーのように特殊な修飾キーはサポートされていません。デフォルトでは super キーはオペレーティングシステムによって定義された option キー、meta キーは command キー、control キーはシステムの control キー、shift キーはシステムの shift キーに対応しています。標準的な PC キーボードでは、Haiku はこれらのキーを GNU システムで慣れ親しんだ位置へとマップする必要がありますが、これを機能させるためにはシステム構成に幾らか調節を要するかもしれません。

システムの super キーのマップを用いてアクセントつき文字をタイプすることは不可能です。

システムが認識している修飾キーと Emacs が認識しているキーの対応づけは、以下に記述する変数によってカスタマイズすることができます。

`haiku-meta-keysym`
: Emacs が Meta キーとして扱うシステムの修飾キー。デフォルトは `command`。

`haiku-control-keysym`
: Emacs が Control キーとして扱うシステムの修飾キー。デフォルトは `control`。

`haiku-super-keysym`
: Emacs が Super キーとして扱うシステムの修飾キー。デフォルトは `option`。

`haiku-shift-keysym`
: Emacs が Shift キーとして扱うシステムの修飾キー。デフォルトは `shift`。

これらの変数それぞれにたいする値としては `command`、`control`、`option`、`shift`、あるいは `nil`のいずれかのシンボルを指定できます。`nil`やこれら以外のすべての値にたいしてはデフォルト値が使用されます。

Haiku での Emacs は、デフォルトではシステムのツールチップメカニズムを使用します。これにより通常はツールチップの応答性が向上しますが、このツールチップではテキストのプロパティやフェイスの表示はできません。これらの機能が必要な場合には、変数 `use-system-tooltips`を `nil`値にカスタマイズすることによって、Emacs 自身で実装されたツールチップを使うことができます。

X ウィンドウシステムとは異なり、Haiku にはシステムワイドなリソースデータベースがありません。重要なオプションの多くは X リソースを通じて指定されるオプションなので、それのエ

ミュレーションが提供されています。Emacs は起動時にユーザーの構成ディレクトリー (通常は `/boot/home/config/settings`) にある `GNU Emacs`という名前のファイルをロードします。このファイルはフラット構造のシステムメッセージでキーと値 (どちらも文字列) はそれぞれ属性とその属性の値に対応しています。

`xmlbmessage`ツールによって、このようなファイルを作成することができます。

G.1.1 Emacs がクラッシュしたら

変数 `haiku-debug-on-fatal-error`が非 `nil`の場合には、致命的シグナルの受信時に Emacs がシステムデバッガを起動します。この変数のデフォルトは `t`です。システムで GDB を使えない場合には、システムデバッガによって生成されたレポートをバグの報告時に添付してください。

G.2 Haiku におけるフォントとフォントバックエンドの選択。

Haiku のウィンドウサポートとともに Emacs をビルドする場合には、複数の異なるフォントバックエンドとともにビルドすることができます。Emacs を呼び出すコマンドラインで`-xrm Emacs.fontBackend:BACKEND` (`BACKEND`とは以下のバックエンドのいずれか) を指定するか、あるいはフレームごとにフレームパラメーター `font-backend`を変更することによってフォントのバックエンドを指定できます。

`ftcr`および `ftcrhb`という 2 つのバックエンドは、それぞれ X ウィンドウシステムの対応するバックエンドと同じです。App Server を使用してフォントを描画する `haiku`という名前の Haiku 固有のバックエンドも存在しますが、現在のところカラーフォントと emoji の表示はサポートしていません。

Appendix H Emacs と Microsoft Windows/MS-DOS

このセクションでは、Microsoft Windows で Emacs を使用する際の特性を説明します。これらの特性の中には、Microsoft's の古い MS-DOS オペレーティングシステムに関連するものもあります。しかし MS-DOS だけに関連する Emacs 機能については、別のマニュアル (Section "MS-DOS" in *Specialized Emacs Features* を参照してください) で説明します。

フリーなオペレーティングシステムのため、MS-Windows はすべてのコンピューターユーザーが享受すべき自由を、ユーザーに与えません。これは不当です。あなたの自由のために、わたしたちはフリーなオペレーティングシステムへの切り替えを強く推奨します。

わたしたちは、商業オペレーティングシステム上の GNU Emacs をサポートします。なぜならそれは、この自由の経験が、商業オペレーティングシステムから脱却するよう、ユーザーを奮起するだろうからです。

MS-Windows での Emacs の振る舞いは、ロングファイル名のサポート、複数フレーム、スクロールバー、マウスメニュー、サブプロセスを含めて、このマニュアルの他の部分でドキュメントされているのと、だいたい同じです。しかし多くはありませんが、特別に考慮すべきこともあるので、それらについてはここで説明します。

H.1 MS-Windows で Emacs を開始する方法

MS-Windows で Emacs を開始するには、いくつかの方法があります:

1. デスクトップのショートカットアイコンから、それを左マウスボタンでダブルクリックするか、1 回クリックしてから `RET`を押します。デスクトップのショートカットは、ショートカットの "Target"(ショートカットの 'Properties" の中にあります) に、`emacs.exe`ではなく、`runemacs.exe`の絶対ファイル名を指定する必要があります。なぜならショートカットのターゲットが `emacs.exe`(Windows から見る限りこれはコンソールプログラムです) のときに作成されるコンソールウィンドウを、`runemacs.exe`は隠すからです。この方法を使用する場合、Emacs はショートカットで指定されたディレクトリーで開始されます。これを制御するには、ショートカットを右クリックして "Properties" を選択し、"Shortcut" タブで "Start in" フィールドを変更します。
2. タスクバーのショートカットアイコンを左マウスボタンで 1 回クリックします。Windows Vista 以降のバージョンでは、タスクバー内に表示される実行中プログラムのアイコンをピン留め *(pinning)* することにより、そのようなショートカットが作成できます。Emacs でこれを行うことができますが、その後にピン留めされたショートカットのプロパティで、実行するプログラムを `emacs.exe` ではなく `runemacs.exe`に変更する必要があるでしょう。Start メニュー内の Emacs アイコンをマウスの右ボタンでクリックして、'`Pin to taskbar`'を選択する方法でも、タスクバーに Emacs をピン留めすることができます。繰り返しますが、実行するプログラムには、`runemacs.exe`を指定してください。ショートカットのプロパティで "Start in" をセットすることにより、Emacs を開始する場所を制御できます。
3. コマンドプロンプトウィンドウから、プロンプトにたいして `emacs RET`とタイプします。そのコマンドプロンプトウィンドウからは、Emacs を終了するまで、他のコマンドを呼び出すことはできなくなります。この場合、Emacs は Windows シェルのカレントディレクトリーで開始されます。
4. コマンドプロンプトウィンドウから、プロンプトにたいして `runemacs RET`とタイプします。そのコマンドプロンプトウィンドウから、すぐに別のコマンドを呼び出すことが可能になります。この場合 Emacs は Windows シェルのカレントディレクトリーで開始されます。

5. Windows の `Run`ダイアログ (通常は `Start`ボタンをクリックしてアクセスできる) を使用します。そのダイアログ内で `runemacs RET`とタイプすれば、Windows でのユーザーの `HOME`ディレクトリーの親ディレクトリーで、Emacs が起動するでしょう。Section H.5 [Windows HOME], page 612 を参照してください。
6. `emacsclient.exe`または `emacsclientw.exe`を通じて Emacs を開始します。これらのコマンドは、Emacs を他のプログラムから呼び出して、他のプログラムから要求された編集ジョブのために、実行中の Emacs プロセスを再使用します。Section 31.6 [Emacs Server], page 469 を参照してください。2 つのコマンドの違いは、`emacsclient.exe`がコンソールプログラムなのにたいして、`emacsclientw.exe`は Windows の GUI プログラムであるという点です。どちらのプログラムも、プログラムを終了して呼び出したプログラムに制御を戻す前に、Emacs が編集ジョブの終了をシグナルするまで待ちます。これらのコマンドを、それぞれどのような場合に使用するかは、編集サービスを必要とするプログラムが期待することに依存します。そのプログラム自身がコンソール (テキストモード) プログラムの場合は、`emacsclient.exe`を使用するべきです。そうすれば呼び出したプログラムと同じコマンドウィンドウにメッセージとプロンプトが表示されます。対照的に呼び出し側のプログラムが GUI プログラムの場合は、`emacsclientw.exe`を使用するほうがよいでしょう。なぜなら `emacsclient.exe`は GUI プログラムから呼び出された場合、コマンドウィンドウをポップアップするからです。`emacsclientw.exe`を使いたい状況としては、Windows Explorer でファイルを右クリックして、ポップアップメニューで “Open With” を選択する場合です。`emacsclient`を呼び出すときに Emacs が実行中でない (またはサーバーとして実行されていない) 場合は、‘`--alternate-editor=`’または ‘`-a`’オプションを使用します。このオプションは常にエディターを与えます。`emacsclient`を通じて呼び出された場合、Emacs は `emacsclient`を呼び出したプログラムのカレントディレクトリーで開始されます。

MS-Windows の制限により、Emacs は同一セッションで GUI とテキストモードのフレームをもつことはできないことに注意してください。また複数のコマンドプロンプトウィンドウでテキストモードのフレームを開くこともできません。なぜなら Windows のプログラムは、それぞれ 1 度に 1 つのコンソールしかもつことができないからです。これらの理由により、`emacsclient`を`-c`オプションで呼び出したとき、Emacs サーバーがテキストモードセッションで実行されている場合、Emacs は常にそれが開始されたのと同じコマンドプロンプトウィンドウに、テキストモードのフレームを開きます。GUI フレームはサーバーが GUI セッションで実行されているときだけ作成されます。同様に、`emacsclient`を`-t`オプションで呼び出したとき、サーバーが GUI セッションで実行されている場合は GUI フレームを作成し、サーバーセッションがコマンドプロンプトウィンドウのテキストモードで実行されている場合はテキストモードのフレームを作成します。Section 31.6.3 [emacsclient Options], page 472 を参照してください。

H.2 テキストファイルとバイナリーファイル

GNU Emacs はテキスト行を分けるのに改行文字を使用します。これは GNU、Unixm その他の POSIX 準拠システムで使用されている慣習です。

対照的に MS-DOS と MS-Windows は、テキスト行を分けるのに、通常は CR(carriage-return: キャリッジリターン) とその後に LF(linefeed: ラインフィード) の、2 文字からなるシーケンスを使用します (LF は改行と同じ文字です)。したがってこれらのファイルを便利に編集するために、Emacs はこれらの行末 (EOL: end-of-line) 文字を変換する必要があります。そしてこれは Emacs が通常行なっていることです。Emacs はファイルを読み込むとき CRLF を改行に変換して、ファイルに書き込むときは改行を CRLF に変換します。国際化文字コードの変換を処理するのと同じ仕組みが、この変換でも行なわれます (Section 19.5 [Coding Systems], page 223 を参照してください)。

ほとんどのファイルにおいて、この特別なフォーマット変換の重要な問題は、Emacs が報告する文字の位置 (Section 4.9 [Position Info], page 24 を参照してください) が、オペレーティングシステムが認識するファイルサイズの情報と一致しないことです。

それに加えて、ファイル内容から行の区切りに CRLF ではなく改行 (LF) が使用されていると Emacs が認識した場合、Emacs はファイルの読み書きで EOL 変換を行ないません。したがって特別に何かを行なわなくても、GNU および Unix システムのファイルを MS-DOS で読み書きでき、編集した後でも、それらのファイルの EOL は Unix スタイルの慣習にしたがいます。

カレントバッファーにたいして、どの EOL 変換が使用されているかは、モードラインに表示されます。そのバッファーにたいして MS-DOS の EOL 変換が使用されている場合、MS-Windows でビルドされた Emacs では、モードラインの先頭付近の、コーディングシステムニーモニックの後ろにバックスラッシュ'\'が表示されます。なんの EOL 変換も処理されていない場合、そのファイルの EOL フォーマットが通常の CRLF ではないことを警告するために、バックスラッシュのかわりに文字列 '`(Unix)`'が表示されます。

ファイルを visit して、DOS スタイルと Unix スタイルのどちらを使用するか指定するには、コーディングシステムを指定します (Section 19.9 [Text Coding], page 227 を参照してください)。たとえば `C-x RET c unix RET C-x C-f foobar.txt`は、EOL 変換をせずに、ファイル `foobar.txt`を visit します。CRLF で終わる行がある場合、Emacs は行末に '`^M`'を表示します。同様に、`C-x RET f`コマンドで、そのバッファーを指定した EOL フォーマットで保存するよう指示できます。たとえばバッファーを Unix の EOL フォーマットで保存するには、`C-x RET f unix RET C-x C-s`とタイプします。DOS の EOL 変換でファイルを visit していて、それを Unix の EOL フォーマットで保存すると、`dos2unix`コマンドのように、そのファイルを Unix の EOL スタイルに変換できます。

NFS、Samba、または他の類似した方法により、GNU および Unix システムを使用しているコンピューターのファイルシステムにアクセスする場合、Emacs はそれらのファイルシステム上のファイル— たとえファイルを新たに作成する場合でも、EOL 変換を行なうべきではありません。EOL 変換を行なわないようにするには、関数 `w32-add-untranslated-filesystem`を呼び出して、それらのファイルシステムが *untranslated(*変換なし*)* だと指定します。この関数は、ドライブ文字とオプションでディレクトリーを含む、ファイルシステム名を引数にとります。たとえば、

```
(w32-add-untranslated-filesystem "Z:")
```

これは Z ドライブが変換なしのファイルシステムであると指定し、

```
(w32-add-untranslated-filesystem "Z:\\foo")
```

これはドライブ Z のディレクトリー`\foo`は、変換なしのファイルシステムだと指定します。

`.emacs`や `init.el`などの init ファイル、または `site-start.el`の中で `w32-add-untranslated-filesystem`を使用するのが、ほとんどでしょう。`site-start.el`に記述しておけば、そのサイトのすべてのユーザーが恩恵にあずかることができます。

`w32-add-untranslated-filesystem`の効果を取り消すには、関数 `w32-remove-untranslated-filesystem`を使用します。この関数は、前に `w32-add-untranslated-filesystem`で使用された文字列と同様の文字列を引数にとります。

ファイルシステムを変換なしと指定しても、影響があるのは EOL 変換だけで、文字セットの変換に影響はありません。原則的として、行末に改行を使用する Unix スタイルをデフォルトとして、新たにファイルを作成するよう Emacs に指示します。Section 19.5 [Coding Systems], page 223 を参照してください。

H.3 MS-Windows のファイル名

MS-Windows と MS-DOS では通常、ファイル名の名前単位の区切りにバックスラッシュを使用します (他のシステムではスラッシュを使用します)。MS-DOS および MS-Windows の Emacs は、スラッシュとバックスラッシュのどちらも使用でき、ファイル名に含まれるドライブ文字も識別できます。

MS-DOS および MS-Windows では、ファイル名は大文字小文字を区別せず、Emacs もデフォルトではファイル名の補完で大文字小文字の違いを無視します。これを行うために、MS-DOS/MS-Windows では `read-file-name-completion-ignore-case`のデフォルト値は非 `nil`です。Section 5.4.5 [Completion Options], page 35 を参照してください。

変数 `w32-get-true-file-attributes`は、`file-attributes`や `directory-files-and-attributes`のように、より正確にプリミティブなファイル属性を判断するために、Emacs が追加のシステムコールを呼び出すべきかを制御します。これらの追加のシステムコールは、ファイルの正しい所有者、リンクカウントと、パイプのような特殊ファイルのファイルタイプを取得するのに必要となります。システムコールを使用しない場合、ファイルの所有者はカレントユーザーとなり、リンクカウントは常に 1 に、そして特殊ファイルは通常ファイルとなるでしょう。

この変数の値が `local`(デフォルト) の場合、Emacs はローカルの固定ドライブのファイルにたいしてのみ、システムコールを呼び出します。他の非 `nil`値は、ファイルがリムーバブルメディアやリモートボリュームにある場合も、システムコールを呼び出すことを意味し、これは Dired やその他の関連する機能の速度低下を招く恐れがあります。値 `nil`はシステムコールを呼び出さないことを意味します。非 `nil`値は、FAT、FAT32、exFAT のようなボリュームより、ハードリンクやファイルセキュリティーをサポートする NTFS のボリュームの場合のほうが有用です。

Unix とは異なり、MS-Windows のファイルシステムでは、ファイル名に使用されるかもしれない複数の文字にたいして制限があります。以下の文字は使用できません:

- シェルのリダイレクション文字 '`<`'、'`>`'、'`|`'。
- コロン '`:`'(ただしドライブ文字を除く)。
- スラッシュ'`/`'とバックスラッシュ'`\`'(ただしディレクトリー区切り文字の場合を除く)。
- ワイルドカード文字 '`*`'と '`?`'。
- コードポイントが 10 進の 1 から 31 の制御文字。特にファイル名の中の改行は許されていません。
- コードポイント 0 の null 文字 (この制限は Unix ファイルシステムにもあります)。

これらに加えて、ファイル名拡張子の有無に関わらず、`NUL`、`LPT1`、`PRN`、`CON`のような DOS の文字デバイスに名前がマッチする任意のファイルは、どのディレクトリーにあっても、文字デバイスとして解釈されます。したがってその文字デバイスを使用したいときだけ、そのようなファイル名を使用します。

H.4 MS-Windows での `ls`のエミュレーション

Dired は通常、Dired バッファーで表示するディレクトリーリストを生成するために、外部プログラムの `ls`を使用します。しかし MS-Windows と MS-DOS には、GNU `ls`のいくつかのポートは存在するものの、システムにはそのようなプログラムがありません。したがって、そのようなシステム上の Emacs は、`ls-lisp.el`パッケージを使用することにより、Lisp で `ls`をエミュレートします。`ls-lisp.el`は、`ls`のほぼ完全なエミュレーションを提供し、エミュレーションに特化したオプションと機能もあります。詳細については、名前が `ls-lisp`で始まる変数のドキュメントを参照してください。

H.5 MS-Windowsでの HOMEディレクトリーと開始ディレクトリー

Windows で HOMEに相当するのは、ユーザー固有のアプリケーションデータディレクトリーで、それの実際の位置は Windows のバージョンに依存します。典型的な値は、Windows 2000 から XP では C:\Documents and Settings*username*\Application Data、Windows Vista 以降では C:\\Users*username*\\AppData\\Roaming、そして Windows 9X/ME では C:\WINDOWS\Application Dataか C:\WINDOWS\Profiles*username*\Application Dataのいずれかです。このディレクトリーが存在しない、またはアクセスできない場合、Emacs は HOMEのデフォルト値を C:\にフォールバックします。

環境変数 HOMEをシステムの他のディレクトリーを指すように明示的にセットすることにより、HOMEのデフォルト値をオーバーライドできます。HOMEはコマンドシェルプロンプト、または 'My Computer' の 'Properties'ダイアログからセットできます。HOMEはシステムレジストリーからもセットできます。Section C.4.3 [MS-Windows Registry], page 584 を参照してください。

古いバージョンの Emacs[1] との互換性のため、ドライブ C:のルートディレクトリー C:\に.emacsという名前のファイルが存在する場合、そして HOMEが環境とレジストリーのどちらでもセットされていない場合、Emacs は C:\をデフォルトの HOMEの場所として扱い、たとえアプリケーションデータディレクトリーが存在する場合でも、そこを探しません。古い名前の_emacs(以下参照) ではなく、C:\の.emacsだけが探されることに注意してください。C:\.emacsを使用して HOMEを定義する方法は、推奨されていません。Emacs はスタートアップ時にそれが使用されていることに関して、警告を表示するでしょう。

最終的な場所がどこであれ、Emacs はその場所を指すように環境変数 HOMEの内部値をセットし、通常ホームディレクトリーで探したり作成するファイルとディレクトリーのために、その場所を使用します。

Emacs がホームディレクトリーをどこだと認識しているかは、C-x d ~/ RETとタイプして常に確認できます。これはホームディレクトリーのファイルのリストを表示し、最初の行にホームディレクトリーの完全な名前を表示します。同様に init ファイルを visit するには、(init ファイルの名前が.emacsの場合)C-x C-f ~/.emacs RETとタイプしてください。

Section 33.4 [Init File], page 528 で述べられるとおり、init ファイルは任意の名前をもつことができます。

MS-DOS はドットで始まるファイル名を使用できず、古い Windows システムではそのような名前のファイルを作成するのが困難だったので、Emacs の Windows ポートは、ホームディレクトリーに_emacsが存在して、.emacsが存在しない場合、_emacsという名前の init ファイルをサポートします。この名前は時代遅れと考えられおり、使用した場合、Emacs は警告を表示するでしょう。

H.6 MS-Windowsでのキーボードの使用方法

このセクションでは、Emacs でのキーボード入力に関する Windows 固有の機能について説明します。

MS-Windows プログラムで慣習的に使用されるキー組み合わせ (“キーボードショートカット” として知られる) の多くが、伝統的な Emacs のキーバインドと衝突します (これら Emacs のキーバインドは、Microsoft が設立される数年前には確立されていました)。衝突の例には C-c、C-x、C-z、C-aが含まれます。CUA モード (Section 9.6 [CUA Bindings], page 70 を参照してください) を有効にすることにより、これらのいくつかの意味を MS-Windows での意味に近づけるよう再定義できます。他の Windows のように Emacs を振る舞わせる他のオプション機能として、Delete Selection モード (Section 8.3 [Using Region], page 55 を参照) があります。

[1] 古いバージョンの Emacs は、アプリケーションデータディレクトリーをチェックしません。

変数 w32-apps-modifierは、Appsキー (通常は右 Altキーと右 Ctrlキーの間にあります) の効果を制御します。変数の値には、対応する修飾キーを示すシンボル hyper、super、meta、alt、control、shiftのどれか 1 つを指定するか、nilを指定してそれをキー appsとして扱います。デフォルトは nilです。

変数 w32-lwindow-modifierは、左 Windows キー (通常は startと Windows のロゴのラベル) の効果を決定します。この変数の値が nil (デフォルト) の場合、このキーはシンボル lwindowを生成します。シンボル hyper、super、meta、alt、control、shiftのうち、どれか 1 つをセットした場合は、対応する修飾が生成されます。これと似た変数 w32-rwindow-modifierは、右 Windows キーの効果を制御し、w32-scroll-lock-modifierは ScrLockと同様のことを行います。これらの変数が nilにセットされている場合、右 Windows キーがシンボル rwindow、ScrLockがシンボル scrollを生成します。たとえばキーボード上の Scroll Lock LED標示を切り替える等、他のアプリケーションのときと同様の効果を ScrLockに生成させたい場合は、w32-scroll-lock-modifier の値に上記の修飾シンボルではなく tか、任意の非 nilをセットしてください。

Emacs がネイティブの Windows アプリケーションとしてコンパイルされていると、Windows メニューを呼び出す Altをタップ (tapping: 覗き見) する Windows 機能をオフに切り替えます。これは Emacs では Metaとして用いられるからです。Emacs を使用するとき、ユーザーが 1 度 Meta キーを押して、後で気が変わることがあります。もしこのキーが Windows メニューを立ち上げる効果をもつ場合、それに続くコマンドの意味が変更されてしまいます。多くのユーザーは、これにイライラするでしょう。

w32-pass-alt-to-systemを非 nil値にセットすることにより、Altキーの覗き見にたいする Windows のデフォルトの処理を再び有効にできます。

H.7 MS-Windows でのマウスの使用方法

このセクションでは、マウスに関連した Windows 固有の変数について説明します。

変数 w32-mouse-button-toleranceは、2 ボタンマウスで、マウス中央ボタンを模倣する際の時間間隔を、ミリ秒で指定します。左ボタンと右ボタンの両方のボタンが、この時間間隔のうちに離された場合、Emacs はそれらどちらかのボタンのダブルクリックイベントのかわりに、マウス中央ボタンのクリックイベントを生成します。

変数 w32-pass-extra-mouse-buttons-to-systemが非 nilの場合、Emacs は Windows に第 4、第 5 マウスボタンを渡します。

変数 w32-swap-mouse-buttons、は 3 ボタンマウスが mouse-2イベントを生成するかを制御します。これが nil(デフォルト) の場合、中央のボタンは mouse-2を生成し、右ボタンは mouse-3を生成します。この変数が非 nilの場合、これら 2 つのボタンの役割は逆になります。

H.8 Windows 9X/ME および Windows NT/2000/XP/Vista/7/8/10 でのサブプロセス

Windows のネイティブアプリケーションとしてコンパイルされた Emacs には、(DOS バージョンとは対照的に) 非同期サブプロセスにたいする完全なサポートが含まれます。Windows バージョンでは、同期および非同期サブプロセスは、すべてのバージョンの Windows で 32 ビット、または 64bit の Windows アプリケーションを実行する限りうまく動作します。しかしサブプロセスで DOS アプリケーションを実行する場合は、問題に遭遇したり、そのアプリケーションを実行できないかもしれません。また、2 つのサブプロセスで 2 つの DOS アプリケーションを実行する場合は、システムを再起動する必要があるかもしれません。

Windows9X 標準のコマンドインプリターは DOS アプリケーションなので、そのようなシステムを使用する場合に、これらの問題が重要になります。しかしこれらの問題についてわたしたちができることはありません。Microsoft だけがこれを fix できるのです。

サブプロセスで 1 つの DOS アプリケーションを実行する場合、それが “様式に従って (well-behaved)” いる限り、そしてスクリーンへの直接アクセスや、その他の異例なことを行なわない限り、そのサブプロセスは期待されたとおりに動作すべきです。CPU モニターアプリケーションがある場合、その DOS アプリケーションがアイドル状態でも、CPU は 100%ビジーに見えるかもしれませんが、これは単に CPU モニターがプロセッサー負荷を計測する方法によるものです。

他の DOS アプリケーションを別のサブプロセスで開始する前に、DOS アプリケーションを終了しなければなりません。Emacs は DOS サブプロセスに割り込み、または終了させることができません。プログラムの終了コマンドを与えることだけが、そのようなサブプロセスを終了できる唯一の方法です。

同時に別のサブプロセスで 2 つの DOS アプリケーションの実行を試みた場合、それらの一方、または両方が非同期であっても、最初のサブプロセスが終了するまで、2 番目に開始されたサブプロセスはサスペンドされます。

もし最初のサブプロセスと対話することができ、終了を指示できたら 2 番目のプロセスは通常どおり実行を継続するはずです。しかし 2 番目のサブプロセスが同期実行されている場合、Emacs は最初のサブプロセスが終了するまでハングするでしょう。最初のプロセスがユーザーによる入力なしには終了しない場合、Windows9X では再起動する以外に選択肢はありません。Windows NT 以降で実行している場合、プロセスビューアーアプリケーションを使用して、適切な NTVDM のインスタンスを kill することができます (これにより、両方の DOS サブプロセスが終了します)。

このような状況で Windows9X を再起動する場合、`Start`メニューの `Shutdown`コマンドを使用しないでください。これは通常システムをハングさせます。かわりに `Ctrl-Alt-DEL`とタイプして `Shutdown`を選択します。これは処理を行なうのに数分かかるかもしれませんが、通常どおり機能します。

変数 `w32-quote-process-args`は、Emacs がプロセス引数をクォートする方法を制御します。非 `nil`は文字`"`でクォートすることを意味します。変数の値が文字の場合、Emacs は任意のクォート文字をエスケープするのに、その文字を使用します。それ以外の場合、プログラムのタイプにもとづいて、適切なエスケープ文字を選択します。

変数 `w32-pipe-buffer-size`は、サブプロセスとの通信のためにパイプを作成するときに、Emacs がシステムに要求するバッファーサイズを制御します。デフォルト値は 0 で、この場合は OS がサイズを選択します。有効な正の値を指定した場合は、そのサイズ (byte) のバッファーを要求します。これは、サブプロセスと、バッファーされたパイプ入出力にたいして通常とは異なる動作を見せるプログラムとの通信を調整するのに使用できます。

H.9 MS-Windows での印刷

POSIX スタイルの `lpr`プログラムが利用できない場合、MS-DOS と MS-Windows では、`lpr-buffer`(Section 31.7 [Printing], page 476 を参照してください) や `ps-print-buffer`(Section 31.7.1 [PostScript], page 477 を参照してください) のような印刷コマンドは、プリンターポートの 1 つに出力を送ります。同じ Emacs 変数がすべてのシステムでの印刷を制御しますが、MS-DOS と MS-Windows では、それらの変数が異なるデフォルト値をもつ場合があります。

MS Windows の Emacs は、(関数 `default-printer-name`を使用して) デフォルトプリンターの自動検出を試みます。しかし、これはある稀なケースでは失敗することがあり、Emacs から別のプリンターを使用したいと思うときがあるかもしれません。このセクションの残りの部分では、Emacs に使用するプリンターを指示する方法を説明します。

ローカルプリンターを使用したい場合、Lisp 変数 lpr-commandに""(これは Windows ではデフォルトです) をセットして、printer-nameにプリンターポート、たとえば通常のローカルプリンターポート"PRN"、または"LPT2"、またはシリアルプリンターにたいする"COM1"などをセットします。printer-nameにファイル名をセットすることもできます。この場合、“印刷” された出力は、そのファイルに追加されます。printer-nameを"NUL"にセットした場合、印刷された出力は破棄されます (システムの null デバイスに送られます)。

printer-nameにそのプリンターの UNC 共有名 — たとえば"//joes_pc/hp4si"のような — をセットすることにより、他のマシンで共有されているプリンターを使用することもできます (ここではスラッシュを使用するか、バックスラッシュを使用するかは問題ではありません)。共有プリンターの名前を探すには、サーバーのリストを取得するために、コマンドプロンプトでコマンド ‘net view’ を実行して、‘net view *server-name*’でそのサーバーで共有されているプリンター (とディレクトリー) の名前を確認します。かわりにデスクトップの ‘Network Neighborhood’アイコンをクリックして、ネットワークを通じてプリンターを共有しているマシンを確認することもできます。

プリンターが ‘net view’で出力されない場合、または printer-nameに UNC 共有名をセットしても、そのプリンターからハードコピーが出力されない場合、‘net use’コマンドを使用して、"LPT2" のようなローカルプリンターポートを、ネットワークプリンターに接続できます。たとえば net use LPT2: \\joes_pc\hp4si[2] により、Windows に LPT2ポートをキャプチャーさせて、印刷物をマシン joes_pcに接続されているプリンターにリダイレクトします。このコマンド後は、printer-name に"LPT2"をセットすることにより、そのネットワークプリンターでハードコピーが印刷されます。

ある Windows ネットワークソフトウェアーでは、"LPT2"のような特定のプリンターポートをキャプチャーして、‘net use’のかわりに Control Panel->Printers を通じてネットワークプリンターにリダイレクトするよう、Windows に指示できます。

printer-nameにファイル名をセットする場合、絶対ファイル名を使用するのが最良です。Emacs はカレントバッファーのデフォルトディレクトリーに合わせて作業ディレクトリーを変更するので、printer-nameのファイル名が相対ファイル名の場合、結果として印刷が行なわれたバッファーのディレクトリーごとに、複数のファイルができてしまいます。

変数 printer-nameの値が正しいのに、印刷してもそのプリンターからハードコピーが印刷されない場合、そのプリンターがプレーンテキストの印刷をサポートしない可能性があります (安価なプリンターのいくつかでは、この機能が省略されています)。そのような場合、以下で説明する PostScript プリントコマンドを試してみてください。

コマンド print-bufferおよび print-regionは、印刷された各ページにヘッダーを生成するために、prプログラムを呼び出すか、lprプログラムの特別なスイッチを使用します。通常 MS-DOS と MS-Windows にはこれらのプログラムがないので、デフォルトでは印刷ページのヘッダーの印刷リクエストは単に無視されるように、lpr-headers-switchesがセットされています。したがって print-bufferと print-regionは、lpr-bufferおよび lpr-regionと同じ出力を生成します。適切な prプログラム (たとえば GNU Coreutils のもの) がある場合は、lpr-headers-switchesに nilをセットします。すると Emacs はページヘッダーを生成するために prを呼び出し、その結果を printer-nameで指定されたプリンターで印刷します。

最後に、もし lprによく似たものがある場合、変数 lpr-commandを"lpr"にセットします。すると Emacs は他のシステムと同じように、印刷に lprを使用します (そのプログラムの名前が lpr でない場合は、lpr-commandに適切な値をセットします)。lpr-commandが""でないときに、変数

[2] printer-nameの値は、スラッシュとバックスラッシュの両方でセットできますが、‘net use’コマンドは UNC 共有名が Windows スタイルのバックスラッシュでタイプされるのを要求することに注意してください。

`lpr-switches`は、その標準的な意味をもちます。変数`printer-name`の値が文字列のとき、Unixの場合のように、`lpr`の`-P`オプションの値としてその文字列が使用されます。

類似の変数`ps-lpr-command`、`ps-lpr-switches`、`ps-printer-name` (Section 31.7.2 [PostScript Variables], page 478 を参照してください) は、PostScript ファイルがどのように印刷されるかを定義します。これらの変数は、上記で説明した非 PostScript 印刷にたいする変数と同じ方法で使用されます。したがって非 PostScript 印刷にたいして`printer-name`が使用される方法と同様に、`ps-printer-name`の値は PostScript 出力が送られるデバイス (またはファイル) の名前として使用されます (2 つの異なるポートに接続された 2 つのプリンターがあり、それらの 1 つだけが PostScript プリンターの場合、2 つの個別の変数セットをもつことができます)。

変数`ps-lpr-command`のデフォルト値は`""`で、これは PostScript 出力を`ps-printer-name`で指定されたプリンターポートに送りますが、`ps-lpr-command`には PostScript ファイルを受け付けるプログラム名をセットすることもできます。したがって非 PostScript プリンターがある場合、この変数に (Ghostscript のような)PostScript のインタープリタープログラムをセットできます。インタープリタープログラムに渡す必要があるスイッチを指定するには、`ps-lpr-switches`を使用します (`ps-printer-name`の値が文字列の場合、`-P`オプションにたいする値として、スイッチのリストが追加されます。これはおそらく`lpr`を使用する場合だけ有用なので、インタープリターを使用するときは`ps-printer-name`に文字列以外の何かをセットすれば、無視させることができます)。

たとえばシステムのデフォルトプリンターで、Ghostscript を使用して印刷するには、以下を`.emacs`に記述します:

```
(setq ps-printer-name t)
(setq ps-lpr-command "D:/gs6.01/bin/gswin32c.exe")
(setq ps-lpr-switches '("-q" "-dNOPAUSE" "-dBATCH"
                        "-sDEVICE=mswinpr2"
                        "-sPAPERSIZE=a4"))
```

(Ghostscript がディレクトリー`D:/gs6.01`にインストールされていると仮定します。)

H.10 MS-Windows でのフォント指定

フォントはフォント名、サイズ、オプションのプロパティにより指定されます。フォントを指定するフォーマットは、モダンなフリーデスクトップで使用されている fontconfig ライブラリーから由来しています。

```
[Family[-PointSize]][:Option1=Value1[:Option2=Value2[...]]]
```

後方互換のため、古い XLFD ベースのフォーマットもサポートされます。

Emacs on MS-Windows では、いくつかのフォントバックエンドがサポートされています。Windows では現在のところバックエンドとして`gdi`、`uniscribe`、および`harfbuzz`が利用可能です。`gdi`フォントバックエンドは、すべてのバージョンの Windows で利用でき、Windows でネイティブにサポートされるすべてのフォントをサポートします。`uniscribe`フォントバックエンドは Windows 2000 以降で利用でき TrueType フォント、OpenType フォントをサポートします。`harfbuzz`フォントバックエンドは Emacs が HarfBuzz サポートつきでビルドされた場合に利用可能で、システムに HarfBuzz DLL がインストールされていれば、このバックエンドは`uniscribe`のように TrueType フォントと OpenType フォントをサポートします。複雑なレイアウトを要求するいくつかの言語は、Uniscribe バックエンドか HarfBuzz バックエンドだけが正しくサポートできます。デフォルトでは`gdi`、および`harfbuzz`か`uniscribe`のいずれかのバックエンドはすべてのフレームで有効であり、どれが有効かは利用可能なバックエンドに依存します (両方が利用可能ならデフォルトでは`harfbuzz`だけが有効)。Emacs が適切なフォントを探す際には、`gdi`より`harfbuzz`および

`uniscribe`のバックエンドが優先されます。これをオーバーライドして、Uniscribe が利用できる場合も GDI バックエンドを使用するには、コマンドライン引数`-xrm Emacs.fontBackend:gdi`を指定して Emacs を呼び出すか、レジストリーのキー '`HKEY_CURRENT_USER\SOFTWARE\GNU\Emacs`' または '`HKEY_LOCAL_MACHINE\SOFTWARE\GNU\Emacs`'のいずれかの下に、リソース`Emacs.fontBackend`を追加して値を`gdi`にセットします (Section D.1 [Resources], page 591 を参照)。同様に HarfBuzz が利用可能であっても Uniscribe バックエンドを使用するには、Emacs を呼び出すコマンドラインで`-xrm Emacs.fontBackend:uniscribe`を使用してください。フレームパラメーターを通じて 3 つのバックエンドすべてを要求することもできますが、その場合にはシステムに利用可能なフォントがない文字にたいするフォント検索には時間を要すると警告されます。

かわりに`modify-frame-parameters`を使用して`font-backend`フレームパラメーターを通じてフレームにたいするフォントバックエンドを指定できます (Section "Parameter Access" in *The Emacs Lisp Reference Manual* を参照)。`default-frame-alist`と`initial-frame-alist`を通じてすべてのフレームにたいするフォントバックエンド (複数可) も要求できます (Section 18.11 [Frame Parameters], page 206 を参照)。`font-backend`パラメーターの値は`(uniscribe)`や`(harfbuzz uniscribe gdi)`のようにシンボルのリストであることに注意してください。

MS-Windows でサポートされるオプションのフォントプロパティです:

`weight` フォントの weight を指定します。特別な値`light`、`medium`、`demibold`、`bold`、`black`は、`weight=`を使わずに指定できます (例: `Courier New-12:bold`)。それ以外の場合、weight は 100 から 900 の数字か、`font-weight-table`の中の名前のついた weight を指定します。指定されない場合は、regular フォントが指定されたとみなします。

`slant` フォントが italic かどうかを指定します。特別な値`roman`、`italic`、`oblique`は、`slant=`を使わずに指定できます (例: `Courier New-12:italic`)。それ以外の場合は、数字か、`font-slant-table`内のなまえつきの slant の 1 つを指定します。Windows では、150 を越える任意の slant は italic として扱われ、150 以下のものはすべて roman として扱われます。

`family` フォントファミリーを指定しますが、通常はフォント名の最初でファミリーを指定します。

`pixelsize`
フォントサイズをピクセルで指定します。これはファミリー名の後のポイントサイズ指定のかわりに使用することができます。

`adstyle` そのフォントにたいする、追加のスタイル情報を指定します。MS-Windows では、値`mono`、`sans`、`serif`、`script`、`decorative`が認識されます。これはフォントファミリーが指定されていない場合のフォールバックとして、もっとも有用です。

`registry` そのフォントがカバーすることを期待される、文字セット registry を指定します。ほとんどの TrueType フォントと OpenType フォントは、複数の国際化文字セット (national character sets) をカバーする Unicode フォントですが、ここで`w32-charset-info-alist`から、特定の文字セットをサポートする registry 指定を使用することにより、選択されるフォントを絞り込むことができます。

`spacing` フォントが spacing される方法を指定します。`p`はプロポーショナルフォントを指定し、`m`および`c`はモノスペースフォントを指定します。

`foundry` Windows では使用されませんが、情報的な目的のために、そしてこれをセットしようとするコードによる問題を防ぐため、ビットマップフォントでは`raster`、スケーラブルフォントでは`outline`、どちらともタイプが特定できなかった場合は`unknown`が、内部的にセットされます。

`script` そのフォントがサポートすべき Unicode の部分範囲 (subrange) を指定します。

Emacs が知るすべてのスクリプト (一般的にはもっとも最近の Unicode 標準で定義されたすべてのスクリプトを意味する) は MS-Windows で認識されます。しかし `GDI`フォントは既知のスクリプトのサブセット `greek`、`hangul`、`kana`、`kanbun`、`bopomofo`、`tibetan`、`yi`、`mongolian`、`hebrew`、`arabic`、`thai`だけをサポートします。

`antialias`

アンチエイリアシング (antialiasing) の方法を指定します。値 `none`は、アンチエイリアシングを行なわないことを意味します。`standard`は、標準のアンチエイリアシングを使用することを意味します。`subpixel`は、subpixel アンチエイリアシング (Windows では *Cleartype* として知られる) を使用することを意味します。`natural`は、文字間の spacing 調整つきで subpixel アンチエイリアシングを使用することを意味します。指定されない場合、そのフォントはシステムのデフォルトのアンチエイリアシングを使用します。

Emacs on MS-Windows が与えられた非 ASCII 文字の表示に適切なフォントを探すために使用する手法はいくつかの希少なスクリプト、特に比較的最近 Unicode に追加されたスクリプトは、たとえそのスクリプトをサポートするフォントをインストールしていても失敗するかもしれません。これは Emacs on MS-Windows がフォントを探すために使用する情報内でそれらのスクリプトがスクリプトにたいして定義された Unicode Subrange Bits (USB) をもたないからです。この問題を克服するために `w32-find-non-USB-fonts`関数を使用できます。これは Emacs セッションの開始に一度実行して、新たにフォントをインストールしたら再度実行する必要があります。Emacs 開始時に毎回この関数を実行するために init ファイルに以下の行を追加できます:

```
(w32-find-non-USB-fonts)
```

かわりに任意のタイミングで `M-:` (Section 24.9 [Lisp Eval], page 329 を参照) からこの関数を実行できます。多くのフォントがインストールされたシステムでは `w32-find-non-USB-fonts`の実行に数秒かかるかもしれません。スタートアップ中に実行するには長すぎると思い、かつフォントを新たにインストールすることが稀なら、`M-:`からこの関数を一度実行してからリターン値が非 `nil`なら init ファイルで変数 `w32-non-USB-fonts`にそれを割り当ててください (関数が `nil`をリターンしたらこの機能を必要とするスクリプトから表示できるインストール済みのフォントはない)。

変数 `w32-use-w32-font-dialog`は `S-mouse-1`を通じてフォントを選択する方法を制御します (`mouse-appearance-menu`)。値が `t`(デフォルト) なら Emacs は標準的な Windows のフォント選択ダイアログを使用します。`nil`なら Emacs はかわりにフォントの固定セットのメニューをポップアップします。メニューに表示されるフォントは `w32-fixed-font-alist`により決定されます。

H.11 その他の Windows 固有の機能

このセクションでは、他のどれにも当てはまらない Windows 固有の機能について説明します。

変数 `w32-use-visible-system-caret`は、システムカレット (system caret) を可視にするか決定するフラグです。スクリーンリーダーソフトウェアーが使用されていないときのデフォルトは `nil`で、これは Emacs がポイント位置を示すために自分でカーソルを描画することを意味します。非 `nil`値は、Emacs がシステムカレットでポイント位置を示すことを意味します。これはスクリーンリーダーソフトウェアーの使用を容易にし、そのようなソフトウェアーが Emacs の実行を検知したときのデフォルトになります。この変数が非 `nil`の場合、カーソル表示に影響を与える他の変数は効果がなくなります。

Windows 10(バージョン 1809 以降) と Windows 11 での Emacs のタイトルバーとスクロールバーは、エクスプローラーやコマンドプロンプトのような他のプログラムと同じよう

に、システムのライトモードやダークモードにしたがいます。このカラーモードを変更するには `Windows Settings` の `Personalization`から `Colors->Choose your color`(あるいは `Choose your default app mode`) を選択して Emacs を再起動してください。

The GNU Manifesto

The GNU Manifesto which appears below was written by Richard Stallman at the beginning of the GNU project, to ask for participation and support. For the first few years, it was updated in minor ways to account for developments, but now it seems best to leave it unchanged as most people have seen it.

Since that time, we have learned about certain common misunderstandings that different wording could help avoid. Footnotes added in 1993 help clarify these points.

For up-to-date information about available GNU software, please see our web site, `https://www.gnu.org`. For software tasks and other ways to contribute, see `https://www.gnu.org/help`.

What's GNU? Gnu's Not Unix!

GNU, which stands for Gnu's Not Unix, is the name for the complete Unix-compatible software system which I am writing so that I can give it away free to everyone who can use it.[1] Several other volunteers are helping me. Contributions of time, money, programs and equipment are greatly needed.

So far we have an Emacs text editor with Lisp for writing editor commands, a source level debugger, a yacc-compatible parser generator, a linker, and around 35 utilities. A shell (command interpreter) is nearly completed. A new portable optimizing C compiler has compiled itself and may be released this year. An initial kernel exists but many more features are needed to emulate Unix. When the kernel and compiler are finished, it will be possible to distribute a GNU system suitable for program development. We will use TEX as our text formatter, but an nroff is being worked on. We will use the free, portable X window system as well. After this we will add a portable Common Lisp, an Empire game, a spreadsheet, and hundreds of other things, plus on-line documentation. We hope to supply, eventually, everything useful that normally comes with a Unix system, and more.

GNU will be able to run Unix programs, but will not be identical to Unix. We will make all improvements that are convenient, based on our experience with other operating systems. In particular, we plan to have longer file names, file version numbers, a crashproof file system, file name completion perhaps, terminal-independent display support, and perhaps eventually a Lisp-based window system through which several Lisp programs and ordinary Unix programs can share a screen. Both C and Lisp will be available as system programming languages. We will try to support UUCP, MIT Chaosnet, and Internet protocols for communication.

---

[1] The wording here was careless. The intention was that nobody would have to pay for *permission* to use the GNU system. But the words don't make this clear, and people often interpret them as saying that copies of GNU should always be distributed at little or no charge. That was never the intent; later on, the manifesto mentions the possibility of companies providing the service of distribution for a profit. Subsequently I have learned to distinguish carefully between "free" in the sense of freedom and "free" in the sense of price. Free software is software that users have the freedom to distribute and change. Some users may obtain copies at no charge, while others pay to obtain copies—and if the funds help support improving the software, so much the better. The important thing is that everyone who has a copy has the freedom to cooperate with others in using it.

GNU is aimed initially at machines in the 68000/16000 class with virtual memory, because they are the easiest machines to make it run on. The extra effort to make it run on smaller machines will be left to someone who wants to use it on them.

To avoid horrible confusion, please pronounce the "G" in the word "GNU" when it is the name of this project.

Why I Must Write GNU

I consider that the golden rule requires that if I like a program I must share it with other people who like it. Software sellers want to divide the users and conquer them, making each user agree not to share with others. I refuse to break solidarity with other users in this way. I cannot in good conscience sign a nondisclosure agreement or a software license agreement. For years I worked within the Artificial Intelligence Lab to resist such tendencies and other inhospitalities, but eventually they had gone too far: I could not remain in an institution where such things are done for me against my will.

So that I can continue to use computers without dishonor, I have decided to put together a sufficient body of free software so that I will be able to get along without any software that is not free. I have resigned from the AI lab to deny MIT any legal excuse to prevent me from giving GNU away.

Why GNU Will Be Compatible with Unix

Unix is not my ideal system, but it is not too bad. The essential features of Unix seem to be good ones, and I think I can fill in what Unix lacks without spoiling them. And a system compatible with Unix would be convenient for many other people to adopt.

How GNU Will Be Available

GNU is not in the public domain. Everyone will be permitted to modify and redistribute GNU, but no distributor will be allowed to restrict its further redistribution. That is to say, proprietary modifications will not be allowed. I want to make sure that all versions of GNU remain free.

Why Many Other Programmers Want to Help

I have found many other programmers who are excited about GNU and want to help.

Many programmers are unhappy about the commercialization of system software. It may enable them to make more money, but it requires them to feel in conflict with other programmers in general rather than feel as comrades. The fundamental act of friendship among programmers is the sharing of programs; marketing arrangements now typically used essentially forbid programmers to treat others as friends. The purchaser of software must choose between friendship and obeying the law. Naturally, many decide that friendship is more important. But those who believe in law often do not feel at ease with either choice. They become cynical and think that programming is just a way of making money.

By working on and using GNU rather than proprietary programs, we can be hospitable to everyone and obey the law. In addition, GNU serves as an example to inspire and a banner to rally others to join us in sharing. This can give us a feeling of harmony which

is impossible if we use software that is not free. For about half the programmers I talk to, this is an important happiness that money cannot replace.

How You Can Contribute

I am asking computer manufacturers for donations of machines and money. I'm asking individuals for donations of programs and work.

One consequence you can expect if you donate machines is that GNU will run on them at an early date. The machines should be complete, ready to use systems, approved for use in a residential area, and not in need of sophisticated cooling or power.

I have found very many programmers eager to contribute part-time work for GNU. For most projects, such part-time distributed work would be very hard to coordinate; the independently-written parts would not work together. But for the particular task of replacing Unix, this problem is absent. A complete Unix system contains hundreds of utility programs, each of which is documented separately. Most interface specifications are fixed by Unix compatibility. If each contributor can write a compatible replacement for a single Unix utility, and make it work properly in place of the original on a Unix system, then these utilities will work right when put together. Even allowing for Murphy to create a few unexpected problems, assembling these components will be a feasible task. (The kernel will require closer communication and will be worked on by a small, tight group.)

If I get donations of money, I may be able to hire a few people full or part time. The salary won't be high by programmers' standards, but I'm looking for people for whom building community spirit is as important as making money. I view this as a way of enabling dedicated people to devote their full energies to working on GNU by sparing them the need to make a living in another way.

Why All Computer Users Will Benefit

Once GNU is written, everyone will be able to obtain good system software free, just like air.[2]

This means much more than just saving everyone the price of a Unix license. It means that much wasteful duplication of system programming effort will be avoided. This effort can go instead into advancing the state of the art.

Complete system sources will be available to everyone. As a result, a user who needs changes in the system will always be free to make them himself, or hire any available programmer or company to make them for him. Users will no longer be at the mercy of one programmer or company which owns the sources and is in sole position to make changes.

Schools will be able to provide a much more educational environment by encouraging all students to study and improve the system code. Harvard's computer lab used to have the policy that no program could be installed on the system if its sources were not on public display, and upheld it by actually refusing to install certain programs. I was very much inspired by this.

[2] This is another place I failed to distinguish carefully between the two different meanings of "free." The statement as it stands is not false—you can get copies of GNU software at no charge, from your friends or over the net. But it does suggest the wrong idea.

Finally, the overhead of considering who owns the system software and what one is or is not entitled to do with it will be lifted.

Arrangements to make people pay for using a program, including licensing of copies, always incur a tremendous cost to society through the cumbersome mechanisms necessary to figure out how much (that is, which programs) a person must pay for. And only a police state can force everyone to obey them. Consider a space station where air must be manufactured at great cost: charging each breather per liter of air may be fair, but wearing the metered gas mask all day and all night is intolerable even if everyone can afford to pay the air bill. And the TV cameras everywhere to see if you ever take the mask off are outrageous. It's better to support the air plant with a head tax and chuck the masks.

Copying all or parts of a program is as natural to a programmer as breathing, and as productive. It ought to be as free.

Some Easily Rebutted Objections to GNU's Goals

> "Nobody will use it if it is free, because that means they can't rely on any support."
>
> "You have to charge for the program to pay for providing the support."

If people would rather pay for GNU plus service than get GNU free without service, a company to provide just service to people who have obtained GNU free ought to be profitable.[3]

We must distinguish between support in the form of real programming work and mere handholding. The former is something one cannot rely on from a software vendor. If your problem is not shared by enough people, the vendor will tell you to get lost.

If your business needs to be able to rely on support, the only way is to have all the necessary sources and tools. Then you can hire any available person to fix your problem; you are not at the mercy of any individual. With Unix, the price of sources puts this out of consideration for most businesses. With GNU this will be easy. It is still possible for there to be no available competent person, but this problem cannot be blamed on distribution arrangements. GNU does not eliminate all the world's problems, only some of them.

Meanwhile, the users who know nothing about computers need handholding: doing things for them which they could easily do themselves but don't know how.

Such services could be provided by companies that sell just hand-holding and repair service. If it is true that users would rather spend money and get a product with service, they will also be willing to buy the service having got the product free. The service companies will compete in quality and price; users will not be tied to any particular one. Meanwhile, those of us who don't need the service should be able to use the program without paying for the service.

> "You cannot reach many people without advertising, and you must charge for the program to support that."
>
> "It's no use advertising a program people can get free."

There are various forms of free or very cheap publicity that can be used to inform numbers of computer users about something like GNU. But it may be true that one can reach more

[3] Several such companies now exist.

microcomputer users with advertising. If this is really so, a business which advertises the service of copying and mailing GNU for a fee ought to be successful enough to pay for its advertising and more. This way, only the users who benefit from the advertising pay for it.

On the other hand, if many people get GNU from their friends, and such companies don't succeed, this will show that advertising was not really necessary to spread GNU. Why is it that free market advocates don't want to let the free market decide this?[4]

"My company needs a proprietary operating system to get a competitive edge."

GNU will remove operating system software from the realm of competition. You will not be able to get an edge in this area, but neither will your competitors be able to get an edge over you. You and they will compete in other areas, while benefiting mutually in this one. If your business is selling an operating system, you will not like GNU, but that's tough on you. If your business is something else, GNU can save you from being pushed into the expensive business of selling operating systems.

I would like to see GNU development supported by gifts from many manufacturers and users, reducing the cost to each.[5]

"Don't programmers deserve a reward for their creativity?"

If anything deserves a reward, it is social contribution. Creativity can be a social contribution, but only in so far as society is free to use the results. If programmers deserve to be rewarded for creating innovative programs, by the same token they deserve to be punished if they restrict the use of these programs.

"Shouldn't a programmer be able to ask for a reward for his creativity?"

There is nothing wrong with wanting pay for work, or seeking to maximize one's income, as long as one does not use means that are destructive. But the means customary in the field of software today are based on destruction.

Extracting money from users of a program by restricting their use of it is destructive because the restrictions reduce the amount and the ways that the program can be used. This reduces the amount of wealth that humanity derives from the program. When there is a deliberate choice to restrict, the harmful consequences are deliberate destruction.

The reason a good citizen does not use such destructive means to become wealthier is that, if everyone did so, we would all become poorer from the mutual destructiveness. This is Kantian ethics; or, the Golden Rule. Since I do not like the consequences that result if everyone hoards information, I am required to consider it wrong for one to do so. Specifically, the desire to be rewarded for one's creativity does not justify depriving the world in general of all or part of that creativity.

"Won't programmers starve?"

I could answer that nobody is forced to be a programmer. Most of us cannot manage to get any money for standing on the street and making faces. But we are not, as a result, condemned to spend our lives standing on the street making faces, and starving. We do something else.

---

[4] The Free Software Foundation raises most of its funds from a distribution service, although it is a charity rather than a company. If *no one* chooses to obtain copies by ordering from the FSF, it will be unable to do its work. But this does not mean that proprietary restrictions are justified to force every user to pay. If a small fraction of all the users order copies from the FSF, that is sufficient to keep the FSF afloat. So we ask users to choose to support us in this way. Have you done your part?

[5] A group of computer companies recently pooled funds to support maintenance of the GNU C Compiler.

But that is the wrong answer because it accepts the questioner's implicit assumption: that without ownership of software, programmers cannot possibly be paid a cent. Supposedly it is all or nothing.

The real reason programmers will not starve is that it will still be possible for them to get paid for programming; just not paid as much as now.

Restricting copying is not the only basis for business in software. It is the most common basis because it brings in the most money. If it were prohibited, or rejected by the customer, software business would move to other bases of organization which are now used less often. There are always numerous ways to organize any kind of business.

Probably programming will not be as lucrative on the new basis as it is now. But that is not an argument against the change. It is not considered an injustice that sales clerks make the salaries that they now do. If programmers made the same, that would not be an injustice either. (In practice they would still make considerably more than that.)

"Don't people have a right to control how their creativity is used?"

"Control over the use of one's ideas" really constitutes control over other people's lives; and it is usually used to make their lives more difficult.

People who have studied the issue of intellectual property rights[6] carefully (such as lawyers) say that there is no intrinsic right to intellectual property. The kinds of supposed intellectual property rights that the government recognizes were created by specific acts of legislation for specific purposes.

For example, the patent system was established to encourage inventors to disclose the details of their inventions. Its purpose was to help society rather than to help inventors. At the time, the life span of 17 years for a patent was short compared with the rate of advance of the state of the art. Since patents are an issue only among manufacturers, for whom the cost and effort of a license agreement are small compared with setting up production, the patents often do not do much harm. They do not obstruct most individuals who use patented products.

The idea of copyright did not exist in ancient times, when authors frequently copied other authors at length in works of non-fiction. This practice was useful, and is the only way many authors' works have survived even in part. The copyright system was created expressly for the purpose of encouraging authorship. In the domain for which it was invented—books, which could be copied economically only on a printing press—it did little harm, and did not obstruct most of the individuals who read the books.

All intellectual property rights are just licenses granted by society because it was thought, rightly or wrongly, that society as a whole would benefit by granting them. But in any particular situation, we have to ask: are we really better off granting such license? What kind of act are we licensing a person to do?

The case of programs today is very different from that of books a hundred years ago. The fact that the easiest way to copy a program is from one neighbor to another, the

[6] In the 80s I had not yet realized how confusing it was to speak of "the issue" of "intellectual property." That term is obviously biased; more subtle is the fact that it lumps together various disparate laws which raise very different issues. Nowadays I urge people to reject the term "intellectual property" entirely, lest it lead others to suppose that those laws form one coherent issue. The way to be clear is to discuss patents, copyrights, and trademarks separately. See `https://www.gnu.org/philosophy/not-ipr.xhtml` for more explanation of how this term spreads confusion and bias.

fact that a program has both source code and object code which are distinct, and the fact that a program is used rather than read and enjoyed, combine to create a situation in which a person who enforces a copyright is harming society as a whole both materially and spiritually; in which a person should not do so regardless of whether the law enables him to.

"Competition makes things get done better."

The paradigm of competition is a race: by rewarding the winner, we encourage everyone to run faster. When capitalism really works this way, it does a good job; but its defenders are wrong in assuming it always works this way. If the runners forget why the reward is offered and become intent on winning, no matter how, they may find other strategies—such as, attacking other runners. If the runners get into a fist fight, they will all finish late.

Proprietary and secret software is the moral equivalent of runners in a fist fight. Sad to say, the only referee we've got does not seem to object to fights; he just regulates them ("For every ten yards you run, you can fire one shot"). He really ought to break them up, and penalize runners for even trying to fight.

"Won't everyone stop programming without a monetary incentive?"

Actually, many people will program with absolutely no monetary incentive. Programming has an irresistible fascination for some people, usually the people who are best at it. There is no shortage of professional musicians who keep at it even though they have no hope of making a living that way.

But really this question, though commonly asked, is not appropriate to the situation. Pay for programmers will not disappear, only become less. So the right question is, will anyone program with a reduced monetary incentive? My experience shows that they will.

For more than ten years, many of the world's best programmers worked at the Artificial Intelligence Lab for far less money than they could have had anywhere else. They got many kinds of non-monetary rewards: fame and appreciation, for example. And creativity is also fun, a reward in itself.

Then most of them left when offered a chance to do the same interesting work for a lot of money.

What the facts show is that people will program for reasons other than riches; but if given a chance to make a lot of money as well, they will come to expect and demand it. Low-paying organizations do poorly in competition with high-paying ones, but they do not have to do badly if the high-paying ones are banned.

> "We need the programmers desperately. If they demand that we stop helping our neighbors, we have to obey."

You're never so desperate that you have to obey this sort of demand. Remember: millions for defense, but not a cent for tribute!

"Programmers need to make a living somehow."

In the short run, this is true. However, there are plenty of ways that programmers could make a living without selling the right to use a program. This way is customary now because it brings programmers and businessmen the most money, not because it is the only way to make a living. It is easy to find other ways if you want to find them. Here are a number of examples.

A manufacturer introducing a new computer will pay for the porting of operating systems onto the new hardware.

The sale of teaching, hand-holding and maintenance services could also employ programmers.

People with new ideas could distribute programs as freeware[7], asking for donations from satisfied users, or selling hand-holding services. I have met people who are already working this way successfully.

Users with related needs can form users' groups, and pay dues. A group would contract with programming companies to write programs that the group's members would like to use.

All sorts of development can be funded with a Software Tax:

> Suppose everyone who buys a computer has to pay x percent of the price as a software tax. The government gives this to an agency like the NSF to spend on software development.
>
> But if the computer buyer makes a donation to software development himself, he can take a credit against the tax. He can donate to the project of his own choosing—often, chosen because he hopes to use the results when it is done. He can take a credit for any amount of donation up to the total tax he had to pay.
>
> The total tax rate could be decided by a vote of the payers of the tax, weighted according to the amount they will be taxed on.
>
> The consequences:
>
> - The computer-using community supports software development.
> - This community decides what level of support is needed.
> - Users who care which projects their share is spent on can choose this for themselves.

In the long run, making programs free is a step toward the post-scarcity world, where nobody will have to work very hard just to make a living. People will be free to devote themselves to activities that are fun, such as programming, after spending the necessary ten hours a week on required tasks such as legislation, family counseling, robot repair and asteroid prospecting. There will be no need to be able to make a living from programming.

We have already greatly reduced the amount of work that the whole society must do for its actual productivity, but only a little of this has translated itself into leisure for workers because much nonproductive activity is required to accompany productive activity. The main causes of this are bureaucracy and isometric struggles against competition. Free software will greatly reduce these drains in the area of software production. We must do this, in order for technical gains in productivity to translate into less work for us.

[7] Subsequently we have discovered the need to distinguish between "free software" and "freeware". The term "freeware" means software you are free to redistribute, but usually you are not free to study and change the source code, so most of it is not free software. See `https://www.gnu.org/philosophy/words-to-avoid.html` for more explanation.

Glossary

Abbrev
: An abbrev is a text string that expands into a different text string when present in the buffer. For example, you might define a few letters as an abbrev for a long phrase that you want to insert frequently. See Chapter 26 [Abbrevs], page 373.

Aborting
: Aborting means getting out of a recursive edit (q.v.). The commands `C-]` and `M-x top-level` are used for this. See Section 34.1 [Quitting], page 536.

Active Region
: Setting the mark (q.v.) at a position in the text also activates it. When the mark is active, we call the region an active region. See Chapter 8 [Mark], page 52.

`Alt`
: Alt is the name of a modifier bit that a keyboard input character may have. To make a character Alt, type it while holding down the `Alt` key. Such characters are given names that start with `Alt-` (usually written `A-` for short). (Note that many terminals have a key labeled `Alt` that is really a `Meta` key.) See Section 2.1 [User Input], page 12.

Argument
: See [Glossary—Numeric Argument], page 644.

ASCII character
: An ASCII character is either an ASCII control character or an ASCII printing character. See Section 2.1 [User Input], page 12.

ASCII control character
: An ASCII control character is the Control version of an upper-case letter, or the Control version of one of the characters '`@[\]^_?`'.

ASCII printing character
: ASCII letters, digits, space, and the following punctuation characters: '`!@#$%^&*()_-+=|\~`{}[]:;"'<>,.?/`'.

Auto Fill Mode
: Auto Fill mode is a minor mode (q.v.) in which text that you insert is automatically broken into lines of a given maximum width. See Section 22.6 [Filling], page 256.

Auto Saving
: Auto saving is the practice of periodically saving the contents of an Emacs buffer in a specially-named file, so that the information will be preserved if the buffer is lost due to a system error or user error. See Section 15.6 [Auto Save], page 159.

Autoloading
: Emacs can automatically load Lisp libraries when a Lisp program requests a function from those libraries. This is called "autoloading". See Section 24.8 [Lisp Libraries], page 327.

Backtrace
: A backtrace is a trace of a series of function calls showing how a program arrived at a certain point. It is used mainly for finding and correcting bugs (q.v.). Emacs can display a backtrace when it signals an error or when you

type `C-g` (see [Glossary—Quitting], page 645). See Section 34.3.4 [Checklist], page 545.

Backup File
: A backup file records the contents that a file had before the current editing session. Emacs makes backup files automatically to help you track down or cancel changes you later regret making. See Section 15.3.2 [Backup], page 152.

Balancing Parentheses
: Emacs can balance parentheses (or other matching delimiters) either manually or automatically. You do manual balancing with the commands to move over parenthetical groupings (see Section 23.4.2 [Moving by Parens], page 293). Automatic balancing works by blinking or highlighting the delimiter that matches the one you just inserted, or inserting the matching delimiter for you (see Section 23.4.3 [Matching Parens], page 293).

Balanced Expressions
: A balanced expression is a syntactically recognizable expression, such as a symbol (q.v.), number, string constant, block, or parenthesized expression in C. See Section 23.4.1 [Expressions], page 292.

Balloon Help
: See [Glossary—Tooltips], page 649.

Base Buffer
: A base buffer is a buffer whose text is shared by an indirect buffer (q.v.).

Bidirectional Text
: Some human languages, such as English, are written from left to right. Others, such as Arabic, are written from right to left. Emacs supports both of these forms, as well as any mixture of them—this is "bidirectional text". See Section 19.20 [Bidirectional Editing], page 238.

Bind
: To bind a key sequence means to give it a binding (q.v.). See Section 33.3.5 [Rebinding], page 521.

Binding
: A key sequence gets its meaning in Emacs by having a binding, which is a command (q.v.)—a Lisp function that is run when you type that sequence. See Section 2.4 [Commands], page 14. Customization often involves rebinding a character to a different command function. The bindings of all key sequences are recorded in the keymaps (q.v.). See Section 33.3.1 [Keymaps], page 519.

Blank Lines
: Blank lines are lines that contain only whitespace. Emacs has several commands for operating on the blank lines in the buffer. See Section 4.7 [Blank Lines], page 23.

Bookmark
: Bookmarks are akin to registers (q.v.) in that they record positions in buffers to which you can return later. Unlike registers, bookmarks persist between Emacs sessions. See Section 10.8 [Bookmarks], page 75.

Border
: A border is a thin space along the edge of the frame, used just for spacing, not for displaying anything. An Emacs frame has an ordinary external border,

outside of everything including the menu bar, plus an internal border that surrounds the text windows, their scroll bars and fringes, and separates them from the menu bar and tool bar. You can customize both borders with options and resources (see Section C.9 [Borders X], page 588). Borders are not the same as fringes (q.v.).

Buffer
: The buffer is the basic editing unit; one buffer corresponds to one text being edited. You normally have several buffers, but at any time you are editing only one, the current buffer, though several can be visible when you are using multiple windows or frames (q.v.). Most buffers are visiting (q.v.) some file. See Chapter 16 [Buffers], page 176.

Buffer Selection History
: Emacs keeps a buffer selection history that records how recently each Emacs buffer has been selected. This is used for choosing which buffer to select. See Chapter 16 [Buffers], page 176.

Bug
: A bug is an incorrect or unreasonable behavior of a program, or inaccurate or confusing documentation. Emacs developers treat bug reports, both in Emacs code and its documentation, very seriously and ask you to report any bugs you find. See Section 34.3 [Bugs], page 542.

Button Down Event
: A button down event is the kind of input event (q.v.) generated right away when you press down on a mouse button. See Section 33.3.10 [Mouse Buttons], page 525.

By Default
: See [Glossary—Default], page 633.

Byte Compilation
: See [Glossary—Compilation], page 632.

cf.
c.f.
: Short for “confer” in Latin, which means “compare with” or “compare to”. The second variant, “c.f.”, is a widespread misspelling.

`C-`
: `C-` in the name of a character is an abbreviation for Control. See Section 2.1 [User Input], page 12.

`C-M-`
: `C-M-` in the name of a character is an abbreviation for Control-Meta. If your terminal lacks a real `Meta` key, you type a Control-Meta character by typing `ESC` and then typing the corresponding Control character. See Section 2.1 [User Input], page 12.

Case Conversion
: Case conversion means changing text from upper case to lower case or vice versa. See Section 22.7 [Case], page 260.

Case Folding
: Case folding means ignoring the differences between case variants of the same letter: upper-case, lower-case, and title-case. Emacs performs case folding by default in text search. See Section 12.9 [Lax Search], page 120.

Character Characters form the contents of an Emacs buffer. Also, key sequences (q.v.) are usually made up of characters (though they may include other input events as well). See Section 2.1 [User Input], page 12.

Character Folding
Character folding means ignoring differences between similarly looking characters, such as between `a`, and `ä` and `á`. Emacs performs character folding by default in text search. See Section 12.9 [Lax Search], page 120.

Character Set
Emacs supports a number of character sets, each of which represents a particular alphabet or script. See Chapter 19 [International], page 216.

Character Terminal
See [Glossary—Text Terminal], page 649.

Click Event
A click event is the kind of input event (q.v.) generated when you press a mouse button and release it without moving the mouse. See Section 33.3.10 [Mouse Buttons], page 525.

Client See [Glossary—Server], page 647.

Clipboard A clipboard is a buffer provided by the window system for transferring text between applications. On the X Window System, the clipboard is provided in addition to the primary selection (q.v.); on MS-Windows and Mac, the clipboard is used *instead* of the primary selection. See Section 9.3.1 [Clipboard], page 65.

Coding System
A coding system is a way to encode text characters in a file or in a stream of information. Emacs has the ability to convert text to or from a variety of coding systems when reading or writing it. See Section 19.5 [Coding Systems], page 223.

Command A command is a Lisp function specially defined to be able to serve as a key binding in Emacs or to be invoked by its name (see [Glossary—Command Name], page 631). (Another term for *command* is *interactive function*—they are used interchangeably.) When you type a key sequence (q.v.), its binding (q.v.) is looked up in the relevant keymaps (q.v.) to find the command to run. See Section 2.4 [Commands], page 14.

Command History
See [Glossary—Minibuffer History], page 643.

Command Name
A command name is the name of a Lisp symbol (q.v.) that is a command (see Section 2.4 [Commands], page 14). You can invoke any command by its name using `M-x` (see Chapter 6 [Running Commands by Name], page 40).

Comment A comment is text in a program which is intended only for humans reading the program, and which is specially marked so that it will be ignored when the program is loaded or compiled. Emacs offers special commands for creating, aligning and killing comments. See Section 23.5 [Comments], page 295.

Common Lisp
: Common Lisp is a dialect of Lisp (q.v.) much larger and more powerful than Emacs Lisp. Emacs provides a subset of Common Lisp in the CL package. See Section "Overview" in *Common Lisp Extensions*.

Compilation
: Compilation is the process of creating an executable program from source code. Emacs has commands for compiling files of Emacs Lisp code (see Section "Byte Compilation" in *the Emacs Lisp Reference Manual*) and programs in C and other languages (see Section 24.1 [Compilation], page 309). Byte-compiled Emacs Lisp code loads and executes faster.

Complete Key
: A complete key is a key sequence that fully specifies one action to be performed by Emacs. For example, `X` and `C-f` and `C-x m` are complete keys. Complete keys derive their meanings from being bound (see [Glossary—Bind], page 629) to commands (q.v.). Thus, `X` is conventionally bound to a command to insert '`X`' in the buffer; `C-x m` is conventionally bound to a command to begin composing a mail message. See Section 2.2 [Keys], page 12.

Completion
: Completion is what Emacs does when it automatically expands an abbreviation for a name into the entire name. Completion is done for minibuffer (q.v.) arguments when the set of possible valid inputs is known; for example, on command names, buffer names, and file names. Completion usually occurs when `TAB`, `SPC` or `RET` is typed. See Section 5.4 [Completion], page 31.

Continuation Line
: When a line of text is longer than the width of the window, it normally takes up more than one screen line when displayed (but see [Glossary—Truncation], page 650). We say that the text line is continued, and all screen lines used for it after the first are called continuation lines. See Section 4.8 [Continuation Lines], page 23. A related Emacs feature is filling (q.v.).

Control Character
: A control character is a character that you type by holding down the `Ctrl` key. Some control characters also have their own keys, so that you can type them without using `Ctrl`. For example, `RET`, `TAB`, `ESC` and `DEL` are all control characters. See Section 2.1 [User Input], page 12.

Copyleft
: A copyleft is a notice giving the public legal permission to redistribute and modify a program or other work of art, but requiring modified versions to carry similar permission. Copyright is normally used to keep users divided and helpless; with copyleft we turn that around to empower users and encourage them to cooperate.

  The particular form of copyleft used by the GNU project is called the GNU General Public License. See Appendix A [Copying], page 555.

`Ctrl`
: The `Ctrl` or control key is what you hold down in order to enter a control character (q.v.). See [Glossary—C-], page 630.

Current Buffer
: The current buffer in Emacs is the Emacs buffer on which most editing commands operate. You can select any Emacs buffer as the current one. See Chapter 16 [Buffers], page 176.

Current Line
: The current line is the line that point is on (see Section 1.1 [Point], page 7).

Current Paragraph
: The current paragraph is the paragraph that point is in. If point is between two paragraphs, the current paragraph is the one that follows point. See Section 22.3 [Paragraphs], page 253.

Current Defun
: The current defun is the defun (q.v.) that point is in. If point is between defuns, the current defun is the one that follows point. See Section 23.2 [Defuns], page 286.

Cursor
: The cursor is the rectangle on the screen which indicates the position (called point; q.v.) at which insertion and deletion takes place. The cursor is on or under the character that follows point. Often people speak of "the cursor" when, strictly speaking, they mean "point". See Section 1.1 [Point], page 7.

Customization
: Customization is making minor changes in the way Emacs works, to reflect your preferences or needs. It is often done by setting variables (see Section 33.2 [Variables], page 508) or faces (see Section 33.1.5 [Face Customization], page 503), or by rebinding key sequences (see Section 33.3.1 [Keymaps], page 519).

Cut and Paste
: See [Glossary—Killing], page 641, and [Glossary—Yanking], page 651.

Daemon
: A daemon is a standard term for a system-level process that runs in the background. Daemons are often started when the system first starts up. When Emacs runs in daemon-mode, it does not open a display. You connect to it with the `emacsclient` program. See Section 31.6 [Emacs Server], page 469.

Default Argument
: The default for an argument is the value that will be assumed if you do not specify one. When the minibuffer is used to read an argument, the default argument is used if you just type `RET`. See Chapter 5 [Minibuffer], page 28.

Default
: A default is the value that is used for a certain purpose when you do not explicitly specify a value to use.

Default Directory
: When you specify a file name that does not start with '`/`' or '`~`', it is interpreted relative to the current buffer's default directory. (On MS systems, file names that start with a drive letter '`x:`' are treated as absolute, not relative.) See Section 5.2 [Minibuffer File], page 28.

Defun
: A defun is a major definition at the top level in a program. The name "defun" comes from Lisp, where most such definitions use the construct `defun`. See Section 23.2 [Defuns], page 286.

DEL
: DEL is a character that runs the command to delete one character of text before the cursor. It is typically either the Delete key or the BACKSPACE key, whichever one is easy to type. See Section 4.3 [Erasing], page 21.

Deletion
: Deletion means erasing text without copying it into the kill ring (q.v.). The alternative is killing (q.v.). See Chapter 9 [Killing], page 59.

Deletion of Files
: Deleting a file means erasing it from the file system. (Note that some systems use the concept of a trash can, or recycle bin, to allow you to undelete files.) See Section 15.12 [Miscellaneous File Operations], page 167.

Deletion of Messages
: Deleting a message (in Rmail, and other mail clients) means flagging it to be eliminated from your mail file. Until you expunge (q.v.) the Rmail file, you can still undelete the messages you have deleted. See Section 30.4 [Rmail Deletion], page 431.

Deletion of Windows
: Deleting a window means eliminating it from the screen. Other windows expand to use up the space. The text that was in the window is not lost, and you can create a new window with the same dimensions as the old if you wish. See Chapter 17 [Windows], page 186.

Directory
: File directories are named collections in the file system, within which you can place individual files or subdirectories. They are sometimes referred to as "folders". See Section 15.8 [Directories], page 162.

Directory Local Variable
: A directory local variable is a local variable (q.v.) that applies to all the files within a certain directory. See Section 33.2.5 [Directory Variables], page 516.

Directory Name
: On GNU and other Unix-like systems, directory names are strings that end in '/'. For example, /no-such-dir/ is a directory name whereas /tmp is not, even though /tmp names a file that happens to be a directory. On MS-Windows the relationship is more complicated. See Section "Directory Names" in *the Emacs Lisp Reference Manual*.

Dired
: Dired is the Emacs facility that displays the contents of a file directory and allows you to "edit the directory", performing operations on the files in the directory. See Chapter 27 [Dired], page 380.

Disabled Command
: A disabled command is one that you may not run without special confirmation. The usual reason for disabling a command is that it is confusing for beginning users. See Section 33.3.11 [Disabling], page 527.

Down Event
: Short for "button down event" (q.v.).

Drag Event
: A drag event is the kind of input event (q.v.) generated when you press a mouse button, move the mouse, and then release the button. See Section 33.3.10 [Mouse Buttons], page 525.

Dribble File
: A dribble file is a file into which Emacs writes all the characters that you type on the keyboard. Dribble files can be used to make a record for debugging Emacs bugs. Emacs does not make a dribble file unless you tell it to. See Section 34.3 [Bugs], page 542.

e.g.
: Short for "exempli gratia" in Latin, which means "for example".

Echo Area
: The echo area is the bottom line of the screen, used for echoing the arguments to commands, for asking questions, and showing brief messages (including error messages). The messages are stored in the buffer `*Messages*` so you can review them later. See Section 1.2 [Echo Area], page 8.

Echoing
: Echoing is acknowledging the receipt of input events by displaying them (in the echo area). Emacs never echoes single-character key sequences; longer key sequences echo only if you pause while typing them.

Electric
: We say that a character is electric if it is normally self-inserting (q.v.), but the current major mode (q.v.) redefines it to do something else as well. For example, some programming language major modes define particular delimiter characters to reindent the line, or insert one or more newlines in addition to self-insertion.

End Of Line
: End of line is a character or a sequence of characters that indicate the end of a text line. On GNU and Unix systems, this is a newline (q.v.), but other systems have other conventions. See Section 19.5 [Coding Systems], page 223. Emacs can recognize several end-of-line conventions in files and convert between them.

Environment Variable
: An environment variable is one of a collection of variables stored by the operating system, each one having a name and a value. Emacs can access environment variables set by its parent shell, and it can set variables in the environment it passes to programs it invokes. See Section C.4 [Environment], page 579.

EOL
: See [Glossary—End Of Line], page 635.

Error
: An error occurs when an Emacs command cannot execute in the current circumstances. When an error occurs, execution of the command stops (unless the command has been programmed to do otherwise) and Emacs reports the error by displaying an error message (q.v.).

Error Message
: An error message is output displayed by Emacs when you ask it to do something impossible (such as, killing text forward when point is at the end of the buffer), or when a command malfunctions in some way. Such messages appear in the echo area, accompanied by a beep.

ESC ESC is a character used as a prefix for typing Meta characters on keyboards lacking a Meta key. Unlike the Meta key (which, like the SHIFT key, is held down while another character is typed), you press the ESC key as you would press a letter key, and it applies to the next character you type.

etc. Short for "et cetera" in Latin, which means "and so on".

Expression
See [Glossary—Balanced Expression], page 629.

Expunging
Expunging an Rmail, Gnus newsgroup, or Dired buffer is an operation that truly discards the messages or files you have previously flagged for deletion.

Face A face is a style of displaying characters. It specifies attributes such as font family and size, foreground and background colors, underline and strike-through, background stipple, etc. Emacs provides features to associate specific faces with portions of buffer text, in order to display that text as specified by the face attributes. See Section 11.8 [Faces], page 82.

File Local Variable
A file local variable is a local variable (q.v.) specified in a given file. See Section 33.2.4 [File Variables], page 512, and [Glossary—Directory Local Variable], page 634.

File Locking
Emacs uses file locking to notice when two different users start to edit one file at the same time. See Section 15.3.4 [Interlocking], page 155.

File Name A file name is a name that refers to a file. File names may be relative or absolute; the meaning of a relative file name depends on the current directory, but an absolute file name refers to the same file regardless of which directory is current. On GNU and Unix systems, an absolute file name starts with a slash (the root directory) or with '`~/`' or '`~user/`' (a home directory). On MS-Windows/MS-DOS, an absolute file name can also start with a drive letter and a colon, e.g., '`d:`'.

Some people use the term "pathname" for file names, but we do not; we use the word "path" only in the term "search path" (q.v.).

File-Name Component
A file-name component names a file directly within a particular directory. On GNU and Unix systems, a file name is a sequence of file-name components, separated by slashes. For example, `foo/bar` is a file name containing two components, '`foo`' and '`bar`'; it refers to the file named '`bar`' in the directory named '`foo`' in the current directory. MS-DOS/MS-Windows file names can also use backslashes to separate components, as in `foo\bar`.

Fill Prefix The fill prefix is a string that should be expected at the beginning of each line when filling is done. It is not regarded as part of the text to be filled. See Section 22.6 [Filling], page 256.

Filling Filling text means adjusting the position of line-breaks to shift text between consecutive lines, so that all the lines are approximately the same length. See

Section 22.6 [Filling], page 256. Some other editors call this feature "line wrapping".

Font Lock Font Lock is a mode that highlights parts of buffer text in different faces, according to the syntax. Some other editors refer to this as "syntax highlighting". For example, all comments (q.v.) might be colored red. See Section 11.13 [Font Lock], page 89.

Fontset A fontset is a named collection of fonts. A fontset specification lists character sets and which font to use to display each of them. Fontsets make it easy to change several fonts at once by specifying the name of a fontset, rather than changing each font separately. See Section 19.14 [Fontsets], page 232.

Formfeed Character
See [Glossary—Page], page 644.

Frame A frame is a rectangular cluster of Emacs windows. Emacs starts out with one frame, but you can create more. You can subdivide each frame into Emacs windows (q.v.). When you are using a window system (q.v.), more than one frame can be visible at the same time. See Chapter 18 [Frames], page 194. Some other editors use the term "window" for this, but in Emacs a window means something else.

Free Software
Free software is software that gives you the freedom to share, study and modify it. Emacs is free software, part of the GNU project (q.v.), and distributed under a copyleft (q.v.) license called the GNU General Public License. See Appendix A [Copying], page 555.

Free Software Foundation
The Free Software Foundation (FSF) is a charitable foundation dedicated to promoting the development of free software (q.v.). For more information, see the FSF website (`https://fsf.org/`).

Fringe On a graphical display (q.v.), there's a narrow portion of the frame (q.v.) between the text area and the window's border. These "fringes" are used to display symbols that provide information about the buffer text (see Section 11.15 [Fringes], page 93). Emacs displays the fringe using a special face (q.v.) called `fringe`. See Section 11.8 [Faces], page 82.

FSF See [Glossary—Free Software Foundation], page 637.

FTP FTP is an acronym for File Transfer Protocol. This is one standard method for retrieving remote files (q.v.).

Function Key
A function key is a key on the keyboard that sends input but does not correspond to any character. See Section 33.3.8 [Function Keys], page 524.

Global Global means "independent of the current environment; in effect throughout Emacs". It is the opposite of local (q.v.). Particular examples of the use of "global" appear below.

Global Abbrev
: A global definition of an abbrev (q.v.) is effective in all major modes that do not have local (q.v.) definitions for the same abbrev. See Chapter 26 [Abbrevs], page 373.

Global Keymap
: The global keymap (q.v.) contains key bindings that are in effect everywhere, except when overridden by local key bindings in a major mode's local keymap (q.v.). See Section 33.3.1 [Keymaps], page 519.

Global Mark Ring
: The global mark ring records the series of buffers you have recently set a mark (q.v.) in. In many cases you can use this to backtrack through buffers you have been editing, or in which you have found tags (see [Glossary—Tags Table], page 649). See Section 8.5 [Global Mark Ring], page 57.

Global Substitution
: Global substitution means replacing each occurrence of one string by another string throughout a large amount of text. See Section 12.10 [Replace], page 122.

Global Variable
: The global value of a variable (q.v.) takes effect in all buffers that do not have their own local (q.v.) values for the variable. See Section 33.2 [Variables], page 508.

GNU
: GNU is a recursive acronym for GNU's Not Unix, and it refers to a Unix-compatible operating system which is free software (q.v.). See [Manifesto], page 620. GNU is normally used with Linux as the kernel since Linux works better than the GNU kernel. For more information, see the GNU website (`https://www.gnu.org/`).

Graphic Character
: Graphic characters are those assigned pictorial images rather than just names. All the non-Meta (q.v.) characters except for the Control (q.v.) characters are graphic characters. These include letters, digits, punctuation, and spaces; they do not include `RET` or `ESC`. In Emacs, typing a graphic character inserts that character (in ordinary editing modes). See Section 4.1 [Inserting Text], page 17.

Graphical Display
: A graphical display is one that can display images and multiple fonts. Usually it also has a window system (q.v.).

Highlighting
: Highlighting text means displaying it with a different foreground and/or background color to make it stand out from the rest of the text in the buffer.

  Emacs uses highlighting in several ways. It highlights the region whenever it is active (see Chapter 8 [Mark], page 52). Incremental search also highlights matches (see Section 12.1 [Incremental Search], page 105). See [Glossary—Font Lock], page 637.

Hardcopy
: Hardcopy means printed output. Emacs has various commands for printing the contents of Emacs buffers. See Section 31.7 [Printing], page 476.

`HELP` `HELP` is the Emacs name for `C-h` or `F1`. You can type `HELP` at any time to ask what options you have, or to ask what a command does. See Chapter 7 [Help], page 42.

Help Echo Help echo is a short message displayed in the echo area (q.v.) when the mouse pointer is located on portions of display that require some explanations. Emacs displays help echo for menu items, parts of the mode line, tool-bar buttons, etc. On graphical displays, the messages can be displayed as tooltips (q.v.). See Section 18.19 [Tooltips], page 213.

Home Directory
Your home directory contains your personal files. On a multi-user GNU or Unix system, each user has his or her own home directory. When you start a new login session, your home directory is the default directory in which to start. A standard shorthand for your home directory is '`~`'. Similarly, '`~user`' represents the home directory of some other user.

Hook A hook is a list of functions to be called on specific occasions, such as saving a buffer in a file, major mode activation, etc. By customizing the various hooks, you can modify Emacs's behavior without changing any of its code. See Section 33.2.2 [Hooks], page 509.

Hyper Hyper is the name of a modifier bit that a keyboard input character may have. To make a character Hyper, type it while holding down the `Hyper` key. Such characters are given names that start with `Hyper-` (usually written `H-` for short). See Section 33.3.7 [Modifier Keys], page 523.

i.e. Short for "id est" in Latin, which means "that is".

Iff "Iff" means "if and only if". This terminology comes from mathematics. Try to avoid using this term in documentation, since many are unfamiliar with it and mistake it for a typo.

Inbox An inbox is a file in which mail is delivered by the operating system. Rmail transfers mail from inboxes to Rmail files in which the mail is then stored permanently or until explicitly deleted. See Section 30.5 [Rmail Inbox], page 432.

Incremental Search
Emacs provides an incremental search facility, whereby Emacs begins searching for a string as soon as you type the first character. As you type more characters, it refines the search. See Section 12.1 [Incremental Search], page 105.

Indentation
Indentation means blank space at the beginning of a line. Most programming languages have conventions for using indentation to illuminate the structure of the program, and Emacs has special commands to adjust indentation. See Chapter 21 [Indentation], page 247.

Indirect Buffer
An indirect buffer is a buffer that shares the text of another buffer, called its base buffer (q.v.). See Section 16.6 [Indirect Buffers], page 183.

Info Info is the hypertext format used by the GNU project for writing documentation.

Input Event
: An input event represents, within Emacs, one action taken by the user on the terminal. Input events include typing characters, typing function keys, pressing or releasing mouse buttons, and switching between Emacs frames. See Section 2.1 [User Input], page 12.

Input Method
: An input method is a system for entering non-ASCII text characters by typing sequences of ASCII characters (q.v.). See Section 19.3 [Input Methods], page 220.

Insertion
: Insertion means adding text into the buffer, either from the keyboard or from some other place in Emacs.

Interactive Function
: A different term for *command* (q.v.).

Interactive Invocation
: A function can be called from Lisp code, or called as a user level command (via `M-x`, a key binding or a menu). In the latter case, the function is said to be *called interactively*.

Interlocking
: See [Glossary—File Locking], page 636.

Isearch
: See [Glossary—Incremental Search], page 639.

Justification
: Justification means adding extra spaces within lines of text in order to adjust the position of the text edges. See Section 22.6.2 [Fill Commands], page 256.

Key Binding
: See [Glossary—Binding], page 629.

Keyboard Macro
: Keyboard macros are a way of defining new Emacs commands from sequences of existing ones, with no need to write a Lisp program. You can use a macro to record a sequence of commands, then play them back as many times as you like. See Chapter 14 [Keyboard Macros], page 137.

Keyboard Shortcut
: A keyboard shortcut is a key sequence (q.v.) that invokes a command. What some programs call "assigning a keyboard shortcut", Emacs calls "binding a key sequence". See [Glossary—Binding], page 629.

Key Sequence
: A key sequence (key, for short) is a sequence of input events (q.v.) that are meaningful as a single unit. If the key sequence is enough to specify one action, it is a complete key (q.v.); if it is not enough, it is a prefix key (q.v.). See Section 2.2 [Keys], page 12.

Keymap
: The keymap is the data structure that records the bindings (q.v.) of key sequences to the commands that they run. For example, the global keymap binds the character `C-n` to the command function `next-line`. See Section 33.3.1 [Keymaps], page 519.

Keyboard Translation Table
: The keyboard translation table is an array that translates the character codes that come from the terminal into the character codes that make up key sequences.

Kill Ring
: The kill ring is where all text you have killed (see [Glossary—Killing], page 641) recently is saved. You can reinsert any of the killed text still in the ring; this is called yanking (q.v.). See Section 9.2 [Yanking], page 62.

Killing
: Killing means erasing text and saving it on the kill ring so it can be yanked (q.v.) later. Some other systems call this "cutting". Most Emacs commands that erase text perform killing, as opposed to deletion (q.v.). See Chapter 9 [Killing], page 59.

Killing a Job
: Killing a job (such as, an invocation of Emacs) means making it cease to exist. Any data within it, if not saved in a file, is lost. See Section 3.2 [Exiting], page 16.

Language Environment
: Your choice of language environment specifies defaults for the input method (q.v.) and coding system (q.v.). See Section 19.2 [Language Environments], page 218. These defaults are relevant if you edit non-ASCII text (see Chapter 19 [International], page 216).

Line Wrapping
: See [Glossary—Filling], page 636.

Lisp
: Lisp is a programming language. Most of Emacs is written in a dialect of Lisp, called Emacs Lisp, which is extended with special features that make it especially suitable for text editing tasks.

List
: A list is, approximately, a text string beginning with an open parenthesis and ending with the matching close parenthesis. In C mode and other non-Lisp modes, groupings surrounded by other kinds of matched delimiters appropriate to the language, such as braces, are also considered lists. Emacs has special commands for many operations on lists. See Section 23.4.2 [Moving by Parens], page 293.

Local
: Local means "in effect only in a particular context"; the relevant kind of context is a particular function execution, a particular buffer, or a particular major mode. It is the opposite of "global" (q.v.). Specific uses of "local" in Emacs terminology appear below.

Local Abbrev
: A local abbrev definition is effective only if a particular major mode is selected. In that major mode, it overrides any global definition for the same abbrev. See Chapter 26 [Abbrevs], page 373.

Local Keymap
: A local keymap is used in a particular major mode; the key bindings (q.v.) in the current local keymap override global bindings of the same key sequences. See Section 33.3.1 [Keymaps], page 519.

Local Variable
: A local value of a variable (q.v.) applies to only one buffer. See Section 33.2.3 [Locals], page 511.

`M-`
: `M-` in the name of a character is an abbreviation for `Meta`, one of the modifier keys that can accompany any character. See Section 2.1 [User Input], page 12.

`M-C-`
: `M-C-` in the name of a character is an abbreviation for Control-Meta; it means the same thing as `C-M-` (q.v.).

`M-x`
: `M-x` is the key sequence that is used to call an Emacs command by name. This is how you run commands that are not bound to key sequences. See Chapter 6 [Running Commands by Name], page 40.

Mail
: Mail means messages sent from one user to another through the computer system, to be read at the recipient's convenience. Emacs has commands for composing and sending mail, and for reading and editing the mail you have received. See Chapter 29 [Sending Mail], page 420. See Chapter 30 [Rmail], page 429, for one way to read mail with Emacs.

Mail Composition Method
: A mail composition method is a program runnable within Emacs for editing and sending a mail message. Emacs lets you select from several alternative mail composition methods. See Section 29.7 [Mail Methods], page 427.

Major Mode
: The Emacs major modes are a mutually exclusive set of options, each of which configures Emacs for editing a certain sort of text. Ideally, each programming language has its own major mode. See Section 20.1 [Major Modes], page 240.

Margin
: The space between the usable part of a window (including the fringe) and the window edge.

Mark
: The mark points to a position in the text. It specifies one end of the region (q.v.), point being the other end. Many commands operate on all the text from point to the mark. Each buffer has its own mark. See Chapter 8 [Mark], page 52.

Mark Ring
: The mark ring is used to hold several recent previous locations of the mark, in case you want to move back to them. Each buffer has its own mark ring; in addition, there is a single global mark ring (q.v.). See Section 8.4 [Mark Ring], page 56.

Menu Bar
: The menu bar is a line at the top of an Emacs frame. It contains words you can click on with the mouse to bring up menus, or you can use a keyboard interface to navigate it. See Section 18.15 [Menu Bars], page 209.

Message
: See [Glossary—Mail], page 642.

Meta
: Meta is the name of a modifier bit which you can use in a command character. To enter a meta character, you hold down the `Meta` key while typing the character. We refer to such characters with names that start with `Meta-` (usually written `M-` for short). For example, `M-<` is typed by holding down `Meta` and

at the same time typing `<` (which itself is done, on most terminals, by holding down `SHIFT` and typing `,`). See Section 2.1 [User Input], page 12.

On some terminals, the `Meta` key is actually labeled `Alt` or `Edit`.

Meta Character
: A Meta character is one whose character code includes the Meta bit.

Minibuffer
: The minibuffer is the window that appears when necessary inside the echo area (q.v.), used for reading arguments to commands. See Chapter 5 [Minibuffer], page 28.

Minibuffer History
: The minibuffer history records the text you have specified in the past for minibuffer arguments, so you can conveniently use the same text again. See Section 5.5 [Minibuffer History], page 36.

Minor Mode
: A minor mode is an optional feature of Emacs, which can be switched on or off independently of all other features. Each minor mode has a command to turn it on or off. Some minor modes are global (q.v.), and some are local (q.v.). See Section 20.2 [Minor Modes], page 241.

Minor Mode Keymap
: A minor mode keymap is a keymap that belongs to a minor mode and is active when that mode is enabled. Minor mode keymaps take precedence over the buffer's local keymap, just as the local keymap takes precedence over the global keymap. See Section 33.3.1 [Keymaps], page 519.

Mode Line
: The mode line is the line at the bottom of each window (q.v.), giving status information on the buffer displayed in that window. See Section 1.3 [Mode Line], page 9.

Modified Buffer
: A buffer (q.v.) is modified if its text has been changed since the last time the buffer was saved (or since it was created, if it has never been saved). See Section 15.3 [Saving], page 150.

Moving Text
: Moving text means erasing it from one place and inserting it in another. The usual way to move text is by killing (q.v.) it and then yanking (q.v.) it. See Chapter 9 [Killing], page 59.

MULE
: Prior to Emacs 23, MULE was the name of a software package which provided a *MULtilingual Enhancement* to Emacs, by adding support for multiple character sets (q.v.). MULE was later integrated into Emacs, and much of it was replaced when Emacs gained internal Unicode support in version 23.

  Some parts of Emacs that deal with character set support still use the MULE name. See Chapter 19 [International], page 216.

Multibyte Character
: A multibyte character is a character that takes up several bytes in a buffer. Emacs uses multibyte characters to represent non-ASCII text, since the number

of non-ASCII characters is much more than 256. See Section 19.1 [International Chars], page 216.

Named Mark
: A named mark is a register (q.v.), in its role of recording a location in text so that you can move point to that location. See Chapter 10 [Registers], page 72.

Narrowing
: Narrowing means creating a restriction (q.v.) that limits editing in the current buffer to only a part of the text. Text outside that part is inaccessible for editing (or viewing) until the boundaries are widened again, but it is still there, and saving the file saves it all. See Section 11.5 [Narrowing], page 81.

Newline
: Control-J characters in the buffer terminate lines of text and are therefore also called newlines. See [Glossary—End Of Line], page 635.

`nil`
: `nil` is a value usually interpreted as a logical "false". Its opposite is `t`, interpreted as "true".

Numeric Argument
: A numeric argument is a number, specified before a command, to change the effect of the command. Often the numeric argument serves as a repeat count. See Section 4.10 [Arguments], page 25.

Overwrite Mode
: Overwrite mode is a minor mode. When it is enabled, ordinary text characters replace the existing text after point rather than pushing it to one side. See Section 20.2 [Minor Modes], page 241.

Package
: A package is a collection of Lisp code that you download and automatically install from within Emacs. Packages provide a convenient way to add new features. See Chapter 32 [Packages], page 490.

Page
: A page is a unit of text, delimited by formfeed characters (ASCII control-L, code 014) at the beginning of a line. Some Emacs commands are provided for moving over and operating on pages. See Section 22.4 [Pages], page 254.

Paragraph
: Paragraphs are the medium-size unit of human-language text. There are special Emacs commands for moving over and operating on paragraphs. See Section 22.3 [Paragraphs], page 253.

Parsing
: We say that certain Emacs commands parse words or expressions in the text being edited. Really, all they know how to do is find the other end of a word or expression.

Point
: Point is the place in the buffer at which insertion and deletion occur. Point is considered to be between two characters, not at one character. The terminal's cursor (q.v.) indicates the location of point. See Section 1.1 [Point], page 7.

Prefix Argument
: See [Glossary—Numeric Argument], page 644.

Prefix Key
: A prefix key is a key sequence (q.v.) whose sole function is to introduce a set of longer key sequences. `C-x` is an example of prefix key; any two-character sequence starting with `C-x` is therefore a legitimate key sequence. See Section 2.2 [Keys], page 12.

Primary Selection
: The primary selection is one particular X selection (q.v.); it is the selection that most X applications use for transferring text to and from other applications.

  The Emacs commands that mark or select text set the primary selection, and clicking the mouse inserts text from the primary selection when appropriate. See Section 8.6 [Shift Selection], page 57.

Prompt
: A prompt is text used to ask you for input. Displaying a prompt is called prompting. Emacs prompts always appear in the echo area (q.v.). One kind of prompting happens when the minibuffer is used to read an argument (see Chapter 5 [Minibuffer], page 28); the echoing that happens when you pause in the middle of typing a multi-character key sequence is also a kind of prompting (see Section 1.2 [Echo Area], page 8).

q.v.
: Short for "quod vide" in Latin, which means "which see".

Query-Replace
: Query-replace is an interactive string replacement feature provided by Emacs. See Section 12.10.4 [Query Replace], page 124.

Quitting
: Quitting means canceling a partially typed command or a running command, using `C-g` (or `C-Break` on MS-DOS). See Section 34.1 [Quitting], page 536.

Quoting
: Quoting means depriving a character of its usual special significance. The most common kind of quoting in Emacs is with `C-q`. What constitutes special significance depends on the context and on convention. For example, an ordinary character as an Emacs command inserts itself; so in this context, a special character is any character that does not normally insert itself (such as `DEL`, for example), and quoting it makes it insert itself as if it were not special. Not all contexts allow quoting. See Section 4.1 [Inserting Text], page 17.

Quoting File Names
: Quoting a file name turns off the special significance of constructs such as '`$`', '`~`' and '`:`'. See Section 15.16 [Quoted File Names], page 170.

Read-Only Buffer
: A read-only buffer is one whose text you are not allowed to change. Normally Emacs makes buffers read-only when they contain text which has a special significance to Emacs; for example, Dired buffers. Visiting a file that is write-protected also makes a read-only buffer. See Chapter 16 [Buffers], page 176.

Rectangle
: A rectangle consists of the text in a given range of columns on a given range of lines. Normally you specify a rectangle by putting point at one corner and putting the mark at the diagonally opposite corner. See Section 9.5 [Rectangles], page 68.

Recursive Editing Level
: A recursive editing level is a state in which part of the execution of a command involves asking you to edit some text. This text may or may not be the same as the text to which the command was applied. The mode line (q.v.) indicates recursive editing levels with square brackets ('`[`' and '`]`'). See Section 31.11 [Recursive Edit], page 484.

Redisplay
: Redisplay is the process of correcting the image on the screen to correspond to changes that have been made in the text being edited. See Chapter 1 [Screen], page 7.

Regexp
: See [Glossary—Regular Expression], page 646.

Region
: The region is the text between point (q.v.) and the mark (q.v.). Many commands operate on the text of the region. See Chapter 8 [Mark], page 52.

Register
: Registers are named slots in which text, buffer positions, or rectangles can be saved for later use. See Chapter 10 [Registers], page 72. A related Emacs feature is bookmarks (q.v.).

Regular Expression
: A regular expression is a pattern that can match various text strings; for example, '`a[0-9]+`' matches '`a`' followed by one or more digits. See Section 12.6 [Regexps], page 115.

Remote File
: A remote file is a file that is stored on a system other than your own. Emacs can access files on other computers provided that they are reachable from your machine over the network, and (obviously) that you have a supported method to gain access to those files. See Section 15.15 [Remote Files], page 169.

Repeat Count
: See [Glossary—Numeric Argument], page 644.

Replacement
: See [Glossary—Global Substitution], page 638.

Restriction
: A buffer's restriction is the amount of text, at the beginning or the end of the buffer, that is temporarily inaccessible. Giving a buffer a nonzero amount of restriction is called narrowing (q.v.); removing a restriction is called widening (q.v.). See Section 11.5 [Narrowing], page 81.

`RET`
: `RET` is a character that in Emacs runs the command to insert a newline into the text. It is also used to terminate most arguments read in the minibuffer (q.v.). See Section 2.1 [User Input], page 12.

Reverting
: Reverting means returning to the original state. For example, Emacs lets you revert a buffer by re-reading its file from disk. See Section 15.4 [Reverting], page 157.

Saving
: Saving a buffer means copying its text into the file that was visited (q.v.) in that buffer. This is the way text in files actually gets changed by your Emacs editing. See Section 15.3 [Saving], page 150.

Scroll Bar
: A scroll bar is a tall thin hollow box that appears at the side of a window. You can use mouse commands in the scroll bar to scroll the window. The scroll bar feature is supported only under windowing systems. See Section 18.12 [Scroll Bars], page 206.

Scrolling
: Scrolling means shifting the text in the Emacs window so as to see a different part of the buffer. See Section 11.1 [Scrolling], page 77.

Searching
: Searching means moving point to the next occurrence of a specified string or the next match for a specified regular expression. See Chapter 12 [Search], page 105.

Search Path
: A search path is a list of directories, to be used for searching for files for certain purposes. For example, the variable `load-path` holds a search path for finding Lisp library files. See Section 24.8 [Lisp Libraries], page 327.

Secondary Selection
: The secondary selection is one particular X selection (q.v.); some X applications can use it for transferring text to and from other applications. Emacs has special mouse commands for transferring text using the secondary selection. See Section 9.3.3 [Secondary Selection], page 66.

Selected Frame
: The selected frame is the one your input currently operates on. See Chapter 18 [Frames], page 194.

Selected Window
: The selected window is the one your input currently operates on. See Section 17.1 [Basic Window], page 186.

Selecting a Buffer
: Selecting a buffer means making it the current (q.v.) buffer. See Section 16.1 [Select Buffer], page 176.

Selection
: Windowing systems allow an application program to specify selections whose values are text. A program can also read the selections that other programs have set up. This is the principal way of transferring text between window applications. Emacs has commands to work with the primary (q.v.) selection and the secondary (q.v.) selection, and also with the clipboard (q.v.).

Self-Documentation
: Self-documentation is the feature of Emacs that can tell you what any command does, or give you a list of all commands related to a topic you specify. You ask for self-documentation with the help character, `C-h`. See Chapter 7 [Help], page 42.

Self-Inserting Character
: A character is self-inserting if typing that character inserts that character in the buffer. Ordinary printing and whitespace characters are self-inserting in Emacs, except in certain special major modes.

Sentences
: Emacs has commands for moving by or killing by sentences. See Section 22.2 [Sentences], page 252.

Server
: Within Emacs, you can start a "server" process, which listens for connections from "clients". This offers a faster alternative to starting several Emacs instances. See Section 31.6 [Emacs Server], page 469, and [Glossary—Daemon], page 633.

Sexp
: A sexp (short for "s-expression") is the basic syntactic unit of Lisp in its textual form: either a list, or Lisp atom. Sexps are also the balanced expressions (q.v.)

of the Lisp language; this is why the commands for editing balanced expressions have '`sexp`' in their name. See Section 23.4.1 [Expressions], page 292.

Simultaneous Editing
: Simultaneous editing means two users modifying the same file at once. Simultaneous editing, if not detected, can cause one user to lose his or her work. Emacs detects all cases of simultaneous editing, and warns one of the users to investigate. See Section 15.3.4 [Simultaneous Editing], page 155.

`SPC`
: `SPC` is the space character, which you enter by pressing the space bar.

Speedbar
: The speedbar is a special tall frame that provides fast access to Emacs buffers, functions within those buffers, Info nodes, and other interesting parts of text within Emacs. See Section 18.9 [Speedbar], page 204.

Spell Checking
: Spell checking means checking correctness of the written form of each one of the words in a text. Emacs can use various external spelling-checker programs to check the spelling of parts of a buffer via a convenient user interface. See Section 13.4 [Spelling], page 134.

String
: A string is a kind of Lisp data object that contains a sequence of characters. Many Emacs variables are intended to have strings as values. The Lisp syntax for a string consists of the characters in the string with a '`"`' before and another '`"`' after. A '`"`' that is part of the string must be written as '`\"`' and a '`\`' that is part of the string must be written as '`\\`'. All other characters, including newline, can be included just by writing them inside the string; however, backslash sequences as in C, such as '`\n`' for newline or '`\241`' using an octal character code, are allowed as well.

String Substitution
: See [Glossary—Global Substitution], page 638.

Symbol
: A symbol in Emacs Lisp is an object with a name. The object can be a variable (q.v.), a function or command (q.v.), or a face (q.v.). The symbol's name serves as the printed representation of the symbol. See Section "Symbol Type" in *The Emacs Lisp Reference Manual*.

Syntax Highlighting
: See [Glossary—Font Lock], page 637.

Syntax Table
: The syntax table tells Emacs which characters are part of a word, which characters balance each other like parentheses, etc. See Section "Syntax Tables" in *The Emacs Lisp Reference Manual*.

Super
: Super is the name of a modifier bit that a keyboard input character may have. To make a character Super, type it while holding down the `SUPER` key. Such characters are given names that start with `Super-` (usually written `s-` for short). See Section 33.3.7 [Modifier Keys], page 523.

Suspending
: Suspending Emacs means stopping it temporarily and returning control to its parent process, which is usually a shell. Unlike killing a job (q.v.), you can

later resume the suspended Emacs job without losing your buffers, unsaved edits, undo history, etc. See Section 3.2 [Exiting], page 16.

`TAB` `TAB` is the tab character. In Emacs it is typically used for indentation or completion.

Tab Bar The tab bar is a row of tabs at the top of an Emacs frame. Clicking on one of these tabs switches named persistent window configurations. See Section 18.17 [Tab Bars], page 210.

Tab Line The tab line is a line of tabs at the top of an Emacs window. Clicking on one of these tabs switches window buffers. See Section 17.8 [Tab Line], page 193.

Tag A tag is an identifier in a program source. See Section 25.4 [Xref], page 357.

Tags Table
A tags table is a file that serves as an index to identifiers: definitions of functions, macros, data structures, etc., in one or more other files. See Section 25.4.2 [Tags Tables], page 363.

Termscript File
A termscript file contains a record of all characters sent by Emacs to the terminal. It is used for tracking down bugs in Emacs redisplay. Emacs does not make a termscript file unless you tell it to. See Section 34.3 [Bugs], page 542.

Text
"Text" has two meanings (see Chapter 22 [Text], page 251):

- Data consisting of a sequence of characters, as opposed to binary numbers, executable programs, and the like. The basic contents of an Emacs buffer (aside from the text properties, q.v.) are always text in this sense.
- Data consisting of written human language (as opposed to programs), or following the stylistic conventions of human language.

Text Terminal
A text terminal, or character terminal, is a display that is limited to displaying text in character units. Such a terminal cannot control individual pixels it displays. Emacs supports a subset of display features on text terminals.

Text Properties
Text properties are annotations recorded for particular characters in the buffer. Images in the buffer are recorded as text properties; they also specify formatting information. See Section 22.14.3 [Editing Format Info], page 276.

Theme A theme is a set of customizations (q.v.) that give Emacs a particular appearance or behavior. For example, you might use a theme for your favorite set of faces (q.v.).

Tool Bar The tool bar is a line (sometimes multiple lines) of icons at the top of an Emacs frame. Clicking on one of these icons executes a command. You can think of this as a graphical relative of the menu bar (q.v.). See Section 18.16 [Tool Bars], page 209.

Tooltips Tooltips are small windows displaying a help echo (q.v.) text, which explains parts of the display, lists useful options available via mouse clicks, etc. See Section 18.19 [Tooltips], page 213.

Top Level Top level is the normal state of Emacs, in which you are editing the text of the file you have visited. You are at top level whenever you are not in a recursive editing level (q.v.) or the minibuffer (q.v.), and not in the middle of a command. You can get back to top level by aborting (q.v.) and quitting (q.v.). See Section 34.1 [Quitting], page 536.

Transient Mark Mode
The default behavior of the mark (q.v.) and region (q.v.), in which setting the mark activates it and highlights the region, is called Transient Mark mode. It is enabled by default. See Section 8.7 [Disabled Transient Mark], page 57.

Transposition
Transposing two units of text means putting each one into the place formerly occupied by the other. There are Emacs commands to transpose two adjacent characters, words, balanced expressions (q.v.) or lines (see Section 13.2 [Transpose], page 132).

Trash Can See [Glossary—Deletion of Files], page 634.

Truncation
Truncating text lines in the display means leaving out any text on a line that does not fit within the right margin of the window displaying it. See Section 4.8 [Continuation Lines], page 23, and [Glossary—Continuation Line], page 632.

TTY See [Glossary—Text Terminal], page 649.

Undoing Undoing means making your previous editing go in reverse, bringing back the text that existed earlier in the editing session. See Section 13.1 [Undo], page 131.

Unix Unix is a class of multi-user computer operating systems with a long history. There are several implementations today. The GNU project (q.v.) aims to develop a complete Unix-like operating system that is free software (q.v.).

User Option
A user option is a face (q.v.) or a variable (q.v.) that exists so that you can customize Emacs by setting it to a new value. See Section 33.1 [Easy Customization], page 499.

Variable A variable is an object in Lisp that can store an arbitrary value. Emacs uses some variables for internal purposes, and has others (known as “user options”; q.v.) just so that you can set their values to control the behavior of Emacs. The variables used in Emacs that you are likely to be interested in are listed in the Variables Index in this manual (see [Variable Index], page 679). See Section 33.2 [Variables], page 508, for information on variables.

Version Control
Version control systems keep track of multiple versions of a source file. They provide a more powerful alternative to keeping backup files (q.v.). See Section 25.1 [Version Control], page 332.

Visiting Visiting a file means loading its contents into a buffer (q.v.) where they can be edited. See Section 15.2 [Visiting], page 146.

Whitespace
Whitespace is any run of consecutive formatting characters (space, tab, newline, backspace, etc.).

Widening Widening is removing any restriction (q.v.) on the current buffer; it is the opposite of narrowing (q.v.). See Section 11.5 [Narrowing], page 81.

Window Emacs divides a frame (q.v.) into one or more windows, each of which can display the contents of one buffer (q.v.) at any time. See Chapter 1 [Screen], page 7, for basic information on how Emacs uses the screen. See Chapter 17 [Windows], page 186, for commands to control the use of windows. Some other editors use the term "window" for what we call a "frame" in Emacs.

Window System
A window system is software that operates on a graphical display (q.v.), to subdivide the screen so that multiple applications can have their own windows at the same time. All modern operating systems include a window system.

Word Abbrev
See [Glossary—Abbrev], page 628.

Word Search
Word search is searching for a sequence of words, considering the punctuation between them as insignificant. See Section 12.3 [Word Search], page 112.

Yanking Yanking means reinserting text previously killed (q.v.). It can be used to undo a mistaken kill, or for copying or moving text. Some other systems call this "pasting". See Section 9.2 [Yanking], page 62.

Key (Character) Index

?

^

{

}

~

1

2

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

Command and Function Index

2

5

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

Variable Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Concept Index

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z