Manuel GNU Emacs

Manuel GNU Emacs

Quatorzième édition, mise à jour pour Emacs Version 21.0.100.

Richard Stallman

Quatorzième Édition
Mise à jour pour Emacs Version 21.0.100,
Août 2000
ISBN 1-882114-06-X

Publié par la Free Software Foundation
59 Temple Place, Suite 330
Boston, MA 02111-1307 USA

Couverture par Etienne Suvasa.

Préface

Ce manuel documente l'utilisation et la personnalisation simple de l'éditeur Emacs. Le lecteur n'est pas supposé être un programmeur ; de simples personnalisations ne nécessitent pas de compétences en programmation. Néammoins, l'utilisateur non intéressé par la personnalisation pourra ignorer les diverses allusions à ce sujet.

Il est en premier lieu un manuel de référence, mais peut aussi être utilisé comme abécédaire. Pour les complets débutants, il est préférable de commencer avec le tutoriel en ligne, avant d'explorer ce manuel. Pour démarrer le tutoriel, lancez Emacs et tapez `C-h t`. Vous pouvez alors apprendre Emacs en l'utilisant, avec un fichier spécialement conçu qui décrit les commandes, vous indique quand les utiliser et explique les résultats obtenus.

En première lecture, parcourez juste les chapitres 1 et 2, qui décrivent les conventions de notation du manuel, ainsi que l'apparence générale de l'écran d'Emacs. Notez quelles questions sont résolues dans ces chapitres, vous pourrez vous y référer plus tard. Après avoir lu le chapitre 4, vous devriez pratiquer les commandes qui s'y trouvent. Les quelques chapitres suivants décrivent des techniques et des concepts fondamentaux utilisés constamment. Vous devez les comprendre parfaitement, en les expérimentant si nécessaire.

Les chapitres 14 à 19 décrivent des caractéristiques de niveau intermédiaire utiles dans tous les cas. Les chapitres 20 et suivants décrivent des caractéristiques que vous voudrez ou non utiliser. Lisez-les si besoin.

Lisez le chapitre Troubles si Emacs semble ne pas fonctionner correctement. Il explique comment venir à bout de problèmes courants (see Section 32.2 [Lossage], page 492), et quand et comment reporter des bogues d'Emacs (see Section 32.3 [Bugs], page 497).

Pour trouver la documentation sur une commande particulière, jetez un œil à l'index. Les touches (caractères de commande) et les noms de commandes ont des index séparés. Il existe aussi un glossaire, avec une référence croisée pour chaque terme.

Ce manuel est disponible en livre mais aussi en fichier Info. Le fichier Info est destiné à être lu en ligne avec le programme Info, qui est le principal moyen de se documenter en ligne sur un système GNU. Le fichier et le programme Info sont tous deux distribués avec GNU Emacs. Le fichier Info et le livre imprimé contiennent en substance le même texte et sont générés à partir du même fichier source, aussi distribué avec GNU Emacs.

GNU Emacs est un membre de la famille des éditeurs Emacs. Il existe un grand nombre d'éditeurs Emacs, partageant tous des principes communs d'organisation. Pour des informations sur la philosophie sous-jacente d'Emacs et les leçons tirées de son développement, écrivez pour obtenir une copie de AI memo 519a, "Emacs, l'éditeur extensible, personnalisable et auto-documenté," à Publications Department, Artificial Intelligence Lab, 545 Tech Square, Cambridge, MA 02139, USA. Ils coûtent $2.25 pièce. Une autre publication utile est LCS TM-165, "A CookBook for an Emacs," par

Craig Finseth, disponible auprès de Publications Department, Laboratory for Computer Science, 545 Tech Square, Cambridge, MA 02139, USA. Son prix est aujourd'hui de $3.

L'édition de ce manuel est prévue pour être utilisée avec GNU Emacs installé sur des systèmes GNU et Unix. GNU Emacs peut aussi être utilisé sur les systèmes VMS, MS-DOS (connu sous le nom de MS-DOG), Windows NT et Windows 95. Ces systèmes utilisent une syntaxe de noms de fichiers différente ; de plus, VMS et MS-DOS ne supportent pas toutes les fonctionnalités de GNU Emacs. Nous ne décrivons pas l'usage sur VMS dans ce manuel. See Appendix E [MS-DOS], page 539, pour des informations sur l'utilisation d'Emacs sur MS-DOS.

Distribution

GNU Emacs est un *logiciel libre* ; cela veut dire que tout le monde est libre de l'utiliser et libre de le redistribuer sous certaines conditions. GNU Emacs n'est pas dans le domaine public ; il est sous copyright et il y a des restrictions sur sa distribution, mais ces restrictions sont conçues pour tout ce qu'un bon citoyen coopératif voudrait faire. Ce qui n'est pas permis est d'essayer d'empêcher les autres de partager les versions de GNU Emacs qu'ils auraient obtenues de vous. Les conditions précises peuvent être trouvées dans la GNU General Public License distribuée avec Emacs et qui apparaît aussi dans le chapitre suivant.

Un moyen d'obtenir une copie de GNU Emacs est de se la procurer auprès de quelqu'un qui l'a déjà. Vous n'avez pas besoin de notre permission pour faire cela, ou de le dire à quiconque. Copiez-le, c'est tout. Si vous avez accès à l'Internet, vous pouvez récupérer la dernière version de la distribution d'Emacs par FTP anonyme ; lisez le fichier '`etc/FTP`' dans la distribution d'Emacs pour plus d'informations.

Vous pouvez aussi recevoir Emacs à l'achat d'un ordinateur. Les revendeurs informatiques sont libres d'en distribuer des copies dans les mêmes conditions que les autres. Ces termes les obligent à vous donner les sources complètes, comprenant les modifications qu'ils auraient apportées, et à vous permettre de redistribuer GNU Emacs tel qu'ils vous l'ont fourni dans les conditions de la General Public License. En d'autres mots, le programme doit être libre pour vous lorsque vous le recevez, pas seulement libre pour le revendeur.

Vous pouvez aussi commander des copies de GNU Emacs auprès de la Free Software Foundation sur CD-ROM. C'est un moyen commode et sûr d'en obtenir une copie ; c'est aussi un bon moyen d'aider à financer nos travaux (La Fondation a toujours reçu la plupart de ses fonds ainsi). Vous trouverez un bon de commande dans le fichier '`etc/ORDERS`' de la distribution d'Emacs et sur notre site Web à `http://www.gnu.org/order/order.html` . Pour plus d'informations, écrivez à :

Free Software Foundation
59 Temple Place, Suite 330
Boston, MA 02111-1307 USA
USA

Les profits sur le prix de la distribution vont au soutien des buts de la Fondation : le développement de nouveaux logiciels libres et l'amélioration de nos logiciels existants comme Emacs.

Si vous trouvez GNU Emacs utile, veuillez **envoyer une donation** à la Free Software Foundation pour soutenir notre travail. Les donations à la Free Software Foundation sont déductibles des impôts aux Etats-Unis. Si vous utilisez Emacs au travail, veuillez suggérer à votre employeur de faire une donation. Si l'idée de donner à la charité lui est douloureuse, vous

devriez plutôt suggérer de commander un CD-ROM auprès de la Fondation à l'occasion, ou de souscrire à des mises à jour périodiques.

Les contributeurs à GNU Emacs sont Per Abrahamsen, Jay K. Adams, Joe Arceneaux, Boaz Ben-Zvi, Jim Blandy, Terrence Brannon, Frank Bresz, Peter Breton, Kevin Broadey, Vincent Broman, David M. Brown, Bill Carpenter, Hans Chalupsky, Bob Chassell, James Clark, Mike Clarkson, Glynn Clements, Andrew Csillag, Doug Cutting, Michael DeCorte, Gary Delp, Matthieu Devin, Eri Ding, Carsten Dominik, Scott Draves, Viktor Dukhovni, John Eaton, Rolf Ebert, Stephen Eglen, Torbjörn Einarsson, Tsugumoto Enami, Hans Henrik Eriksen, Michael Ernst, Ata Etemadi, Frederick Farnback, Fred Fish, Karl Fogel, Gary Foster, Noah Friedman, Keith Gabryelski, Kevin Gallagher, Kevin Gallo, Howard Gayle, Stephen Gildea, David Gillespie, Bob Glickstein, Boris Goldowsky, Michelangelo Grigni, Michael Gschwind, Henry Guillaume, Doug Gwyn, Ken'ichi Handa, Chris Hanson, K. Shane Hartman, John Heidemann, Markus Heritsch, Karl Heuer, Manabu Higashida, Anders Holst, Kurt Hornik, Tom Houlder, Lars Ingebrigtsen, Andrew Innes, Michael K. Johnson, Kyle Jones, Tomoji Kagatani, Brewster Kahle, David Kaufman, Henry Kautz, Howard Kaye, Michael Kifer, Richard King, Larry K. Kolodney, Robert Krawitz, Sebastian Kremer, Geoff Kuenning, David Kågedal, Daniel LaLiberte, Aaron Larson, James R. Larus, Frederic Lepied, Lars Lindberg, Eric Ludlam, Neil M. Mager, Ken Manheimer, Bill Mann, Brian Marick, Simon Marshall, Bengt Martensson, Charlie Martin, Thomas May, Roland McGrath, David Megginson, Wayne Mesard, Richard Mlynarik, Keith Moore, Erik Naggum, Thomas Neumann, Mike Newton, Jurgen Nickelsen, Jeff Norden, Andrew Norman, Jeff Peck, Damon Anton Permezel, Tom Perrine, Jens Petersen, Daniel Pfeiffer, Fred Pierresteguy, Christian Plaunt, Francesco A. Potorti, Michael D. Prange, Ashwin Ram, Eric S. Raymond, Paul Reilly, Edward M. Reingold, Rob Riepel, Roland B. Roberts, John Robinson, Danny Roozendaal, William Rosenblatt, Guillermo J. Rozas, Ivar Rummelhoff, Wolfgang Rupprecht, James B. Salem, Masahiko Sato, William Schelter, Ralph Schleicher, Gregor Schmid, Michael Schmidt, Ronald S. Schnell, Philippe Schnoebelen, Stephen Schoef, Randal Schwartz, Manuel Serrano, Stanislav Shalunov, Mark Shapiro, Richard Sharman, Olin Shivers, Espen Skoglund, Rick Sladkey, Lynn Slater, Chris Smith, David Smith, Paul D. Smith, William Sommerfeld, Michael Staats, Sam Steingold, Ake Stenhoff, Peter Stephenson, Jonathan Stigelman, Steve Strassman, Jens T. Berger Thielemann, Spencer Thomas, Jim Thompson, Masanobu Umeda, Neil W. Van Dyke, Ulrik Vieth, Geoffrey Voelker, Johan Vromans, Barry Warsaw, Morten Welinder, Joseph Brian Wells, Rodney Whitby, Ed Wilkinson, Mike Williams, Steven A. Wood, Dale R. Worley, Felix S. T. Wu, Tom Wurgler, Eli Zaretskii, Jamie Zawinski, Ian T. Zimmermann, Reto Zimmermann, and Neal Ziring.

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) 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 this service 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 make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. 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.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will in-

dividually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program," below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification.") Each licensee is addressed as "you."

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute 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 and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such in-

teractive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.
6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), 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 distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy

both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
9. The Free Software Foundation may publish revised and/or new versions of the 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 a version number of this License which applies to it and "any later version," you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.
10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, 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.
12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE 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.

END OF TERMS AND CONDITIONS

Comment appliquer ces termes à votre nouveau programme

Si vous développez un nouveau programme, et si vous désirez qu'il soit d'une utilité la plus grande possible pour le public, le meilleur moyen est d'en faire un logiciel libre, que chacun pourra redistribuer et modifier sous ces termes.

Pour cela, attachez les notifications suivantes au programme. Il est plus prudent de les attacher au début de chaque fichier source pour faire comprendre plus sûrement l'absence de garantie ; et chaque fichier devrait avoir au moins une ligne de "copyright" et un pointeur vers le lieu où la notification complète peut être trouvée.

> *une ligne pour donner le nom du programme et une idée de ce qu'il fait.*
> Copyright (C) 19*yy* *nom de l'auteur*
>
> 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 2
> 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, write to the Free Software Foundation, Inc.,
> 59 Temple Place, Suite 330, Boston, MA 02111-1307, USA.

Ajoutez aussi comment vous contacter par courrier électronique et postal.

Si le programme est interactif, faites-lui afficher une courte notification de ce style lorsqu'il démarre en mode interactif :

> Gnomovision version 69, Copyright (C) 20*yy* *nom de l'auteur*
> Gnomovision 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.

Les commandes hypothétiques '`show w`' et '`show c`' devraient montrer les passages appropriés de la General Public License. Bien sûr, les commandes que vous utilisez peuvent être appelées autrement que '`show w`' et '`show c`' ; elles peuvent aussi bien être basées sur un clic de souris ou une entrée de menu — ce qui convient le mieux à votre programme.

Vous devriez aussi faire signer à votre employeur (si vous travaillez comme programmeur) ou votre école (s'il y a lieu) un "renoncement de copyright"

pour le programme, si nécessaire. Voici un exemple, en changeant les noms :

```
Yoyodyne, Inc., déclare par la présente renoncer à tout copyright sur le
programme ‘Gnomovision’ (qui fait des passes aux compilateurs)
écrit par Pierre Bidouille.

signature de Jean Gagne, 1er Avril 1989
Jean Gagne, Président du Vice
```

Cette General Public License ne permet pas d'incorporer votre programme dans des programmes propriétaires. Si votre programme est une bibliothèque de sous-routines, vous pourriez considérer plus utile de permettre de lier des applications propriétaires avec votre bibliothèque. Si c'est ce que vous voulez faire, utilisez la GNU Library General Public License au lieu de cette Licence.

Appendix A GNU Free Documentation License

Version 1.1, March 2000

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written 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 noncommercially. 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 that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you."

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. (For example, 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.

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 "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and 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 has been designed to thwart or discourage subsequent modification by readers is not Transparent. 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 designed for human modification. Opaque formats include PostScript, PDF, 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 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.

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 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 publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. 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 less than five).
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", and 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. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, 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 as "Endorsements" or to conflict in title with any Invariant Section.

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.

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, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they 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 quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around 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 provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

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 http://www.gnu.org/copyleft/.

    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.

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 1.1
> or any later version published by the Free Software Foundation;
> with the Invariant Sections being *list their titles*, with the
> Front-Cover Texts being *list*, and with the Back-Cover Texts being *list*.
> A copy of the license is included in the section entitled "GNU
> Free Documentation License."

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being *list*"; likewise for Back-Cover Texts.

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.

Introduction

Ce manuel traite d'Emacs, l'incarnation selon GNU de l'éditeur avancé, auto-documenté, personnalisable, extensible et visuel en temps réel. (Le 'G' de GNU n'est pas muet.)

Nous disons qu'Emacs est un éditeur *visuel* car, normalement, le texte étant édité est visible à l'écran et mis à jour automatiquement lorsque vous tapez vos commandes. See Chapter 1 [Screen], page 23.

Nous l'appelons un éditeur en *temps réel* car l'affichage est mis à jour très fréquemment, généralement après chaque caractère ou paire de caractères tapés. Ce qui minimise la quantité d'informations que vous devez garder en mémoire lorsque vous éditez. See Chapter 4 [Basic Editing], page 41.

Nous qualifions Emacs d'*avancé* car il fournit des facilités au delà de la simple insertion et suppression : contrôle de sous-processus, indentation automatique de programmes, visualisation de deux fichiers ou plus à la fois, édition de texte formaté, traitement en terme de caractères, mots, lignes, phrases, paragraphes et pages, mais aussi expressions et commentaires pour différents langages de programmation.

Auto-documenté veut dire qu'à tout moment, vous pouvez taper le caractère spécial `Control-h`, pour connaître vos différentes options. Vous pouvez aussi l'utiliser pour savoir ce que fait telle commande, ou pour trouver toutes les commandes se rapportant à un sujet. See Chapter 7 [Help], page 67.

Personnalisable veut dire que vous pouvez changer les définitions des commandes d'Emacs en un rien de temps. Par exemple, si vous utilisez un langage de programmation pour lequel les commentaires commencent par '`<**`' et finissent par '`**>`', vous pouvez indiquer aux commandes de manipulation de commentaires d'Emacs d'utiliser ces chaînes (see Section 22.7 [Comments], page 291). Un autre genre de personnalisation est le réarrangement du jeu de commandes. Par exemple, si vous préférez que les quatre commandes de déplacement élémentaire (haut, bas, gauche et droite) se trouvent sur des touches placées en diamant sur le clavier, vous pouvez rattacher les touches ainsi. See Chapter 31 [Customization], page 455.

Extensible veut dire que vous pouvez aller plus loin que la simple personnalisation et écrire des commandes totalement nouvelles, sous forme de programmes en langage Lisp exécutables par le propre interpréteur Lisp d'Emacs. Emacs est un système "extensible en ligne", ce qui veut dire qu'il est formé d'un grand nombre de fonctions s'appelant entre elles, chacune d'elles pouvant être redéfinie au milieu d'une session d'édition. A peu près n'importe quelle partie d'Emacs peut être remplacée sans faire une copie séparée d'Emacs en entier. La plupart des commandes d'édition d'Emacs sont déjà écrites en Lisp ; les quelques exceptions auraient pu être écrites en Lisp mais ont été écrites en C pour des raisons d'efficacité. Bien que seul un programmeur puisse écrire une extension, tout le monde peut ensuite

l'utiliser. Si vous voulez apprendre la programmation en Emacs Lisp, nous recommandons *Introduction to Emacs Lisp* de Robert J. Chassel, également publié par la Free Software Foundation.

Exécuté sous le système X Window, Emacs fournit ses propres menus et des raccourcis commodes aux boutons de souris. Mais Emacs peut fournir beaucoup des avantages d'un système de fenêtrage sur un terminal texte. Par exemple, vous pouvez parcourir ou éditer plusieurs fichiers à la fois, déplacer du texte entre fichiers et éditer des fichiers tout en exécutant des commandes shell.

1 L'organisation de l'écran

Sur un terminal texte, l'affichage d'Emacs occupe tout l'écran. Sous le système X Window, Emacs crée ses propres fenêtres X. Nous utilisons le terme *cadre* pour un écran texte entier ou une fenêtre X entière utilisés par Emacs. Emacs utilise ces deux types de cadres de la même manière pour afficher votre édition. Emacs démarre normalement avec un seul cadre, mais vous pouvez créer des cadres additionnels si vous le désirez. See Chapter 17 [Frames], page 203.

Lorsque vous démarrez Emacs, le cadre entier excepté la première et la dernière ligne est réservé au texte que vous éditez. Cette zone est appelée la *fenêtre*. La première ligne est une *barre de menus*, et la dernière ligne est une *zone de répercussion* ou *fenêtre de mini-tampon* où des prompts apparaissent et où vous pouvez entrer vos réponses. Voir plus loin pour plus d'informations sur ces lignes spéciales.

Vous pouvez subdiviser la grande fenêtre de texte horizontalement ou verticalement en plusieurs fenêtres de texte, chacune d'elle pouvant être utilisée pour un fichier différent (see Chapter 16 [Windows], page 195). Dans ce manuel, le mot "fenêtre" se réfère toujours aux subdivisions d'un cadre d'Emacs.

La fenêtre où se trouve le curseur est la *fenêtre selectionnée*, dans laquelle l'édition a lieu. La plupart des commandes d'Emacs s'appliquent implicitement au texte de la fenêtre sélectionnée (cependant les commandes souris opèrent généralement dans la fenêtre où vous cliquez, selectionnée ou non). Les autres fenêtres affichent leurs propres textes pour référence seulement, à moins que/jusqu'à ce que vous les sélectionniez. Si vous utilisez plusieurs cadres sous le système X Window, donner le focus à un cadre particulier sélectionne une fenêtre dans ce cadre.

Chaque dernière ligne de fenêtre est une *ligne de mode*, qui décrit ce qui se passe dans cette fenêtre. Elle apparaît en inversion vidéo, si le terminal le permet, et son contenu commence par '`--:--  *scratch*`' au démarrage d'Emacs. La ligne de mode affiche des informations sur la situation comme quel tampon est affiché au dessus de lui dans la fenêtre, quels modes majeurs et mineurs sont utilisés, et si le tampon contient des changements non enregistrés.

1.1 Point

A l'intérieur d'Emacs, le curseur du terminal montre l'endroit où les commandes d'édition prendront effet. Cet endroit est appelé *point*. Beaucoup de commandes Emacs déplacent le point à travers le texte, vous pouvez ainsi éditer à différents endroits. Vous pouvez aussi placer le point avec le bouton 1 de la souris.

Alors que le curseur apparaît comme pointant *sur* un caractère, vous devez imaginer le point *entre* deux caractères ; il pointe *avant* le caractère apparaissant sous le curseur. Par exemple, si votre texte ressemble à ‘`frob`’ avec le curseur sur le ‘`b`’, le point se trouve entre le ‘`o`’ et le ‘`b`’. Si vous insérez le caractère ‘`!`’ à cette position, le résultat est ‘`fro!b`’, avec le point entre le ‘`!`’ et le ‘`b`’. Par conséquent, le curseur reste sur le ‘`b`’, comme précédemment.

Il arrive que l'on dise le “curseur” alors que l'on devrait dire “le point”, ou que l'on parle de commandes qui déplacent le point comme commande de “déplacement du curseur”.

Les terminaux ont un seul curseur, et lorsque des saisies sont en cours, il doit apparaître là où la saisie a lieu. Cela ne veut pas dire que le point se déplace. C'est seulement qu'Emacs n'a pas le moyen de vous montrer la location du point, sauf lorsque le terminal est au repos.

Si vous éditez plusieurs fichiers sous Emacs, chacun dans son propre tampon, chaque tampon a sa propre position du point. Un tampon qui n'est pas actuellement affiché se rappelle la position de son point, au cas où vous voudriez le réafficher plus tard.

Lorsqu'il y a plusieurs fenêtres dans un cadre, chaque fenêtre a sa propre position du point. Le curseur montre l'emplacement du point dans la fenêtre sélectionnée. C'est aussi la manière de savoir quelle fenêtre est sélectionnée. Si un même tampon apparaît dans plus d'une fenêtre, chaque fenêtre a sa propre position du point dans ce tampon.

Lorsqu'il y a plusieurs cadres, chacun d'eux peut afficher un curseur. Le curseur dans le cadre sélectionné est plein ; le curseur de chacun des autres cadres apparaît comme un rectangle vide, et apparaît dans la fenêtre qui serait sélectionnée si vous donnez le focus à ce cadre.

Le terme “point” vient du caractère ‘`.`’, qui était la commande TECO (le langage dans lequel la version originale d'Emacs fut écrite) pour accéder à la valeur maintenant appelée “point.”

1.2 La zone de répercussion

La ligne en bas du cadre (sous la ligne de mode) est la *zone de répercussion*. Elle est utilisée pour afficher de petites quantités de texte en diverses occasions.

Répercuter veut dire afficher les caractères que vous tapez. En dehors d'Emacs, le système d'exploitation répercute normalement toutes vos saisies. Emacs contrôle la répercussion différemment.

Les commandes à caractère unique ne sont pas répercutées sous Emacs, et les commandes à caractères multiples sont répercutées seulement si vous faites une pause en les tapant. Dès que vous faites une pause de plus d'une seconde au milieu d'une commande, Emacs répercute tous les autres car-

actères de la commande. Ceci est fait pour vous *souffler* le reste de la commande. Une fois que la répercussion a commencé, la suite de la commande se répercute dès que vous la tapez. Ce comportement est étudié pour donner aux utilisateurs assurés une réponse rapide, et aux utilisateurs plus hésitants un maximum de réaction. Vous pouvez changer ce comportement en changeant la valeur d'une variable (see Section 11.12 [Display Custom], page 115).

Si une commande ne peut être exécutée, elle peut afficher un *message d'erreur* dans la zone de répercussion. Les messages d'erreur sont accompagnés d'une sonnerie ou d'un flash de l'écran. De plus, les caractères tapés précédemment sont abandonnés lorsqu'une erreur arrive.

Certaines commandes affichent des messages d'information dans la zone de répercussion. Ces messages ressemblent plus à des messages d'erreur, mais ne sont pas annoncés par une sonnerie et les entrées précédentes ne sont pas perdues. Parfois le message vous dit ce que la commande a fait, lorsque ce n'est pas évident à la vue du texte que vous tapez. Parfois, le seul comportement d'une commande est d'afficher un message d'information—par exemple, `C-x =` affiche un message décrivant le caractère placé après le point, la position du point et sa colonne courante dans la fenêtre. Les commandes qui prennent un certain temps affichent aussi des messages finissant par '`...`' pendant qu'elles s'exécutent, et '`done`' lorsqu'elles ont fini.

Les messages d'information de la zone de répercussion sont enregistrés dans un tampon de l'éditeur appelé '`*Messages*`'. (Nous n'avons pas encore expliqué les tampons ; voir Chapter 15 [Buffers], page 185 pour plus d'informations.) Si vous ratez un message qui appraît brièvement à l'écran, vous pouvez vous reporter au tampon '`*Messages*`' pour le voir de nouveau. (Les messages de progression successifs sont fréquemment réduits en un seul dans ce tampon.)

La taille de '`*Messages*`' est limitée à un certain nombre de lignes. La variable `message-log-max` spécifie ce nombre de lignes. Une fois que le tampon atteint cette taille, chaque ligne ajoutée à la fin supprime une ligne au début. See Section 31.2 [Variables], page 457, pour savoir commencer positionner les variables comme `message-log-max`.

La zone de répercussion est aussi utilisée pour afficher le *mini-tampon*, une fenêtre utilisée pour lire les arguments d'une commande, comme le nom d'un fichier à éditer. Quand le mini-tampon est utilisé, la zone de répercussion commence avec une chaîne de prompt généralement terminée par ':' ; de plus, le curseur apparaît sur cette ligne car c'est la fenêtre sélectionnée. Vous pouvez toujours sortir du mini-tampon en tapant `C-g`. See Chapter 5 [Minibuffer], page 55.

1.3 La ligne de mode

Chaque dernière ligne d'une fenêtre de texte est une *ligne de mode*, qui décrit ce qui arrive dans cette fenêtre. Lorsqu'il n'y a qu'une seule fenêtre de texte, la ligne de mode apparaît juste au dessus de la zone de répercussion ; c'est l'avant-dernière ligne du cadre. La ligne de mode commence et finit avec des tirets. Pour un affichage en texte seulement, la ligne de mode est en inversion vidéo si le terminal le permet ; pour un affichage graphique, la ligne de mode a une apparence de boîte 3D pour aider à la différencier.

La ligne de mode ressemble normalement à :

-cs:ch tampon (majeur mineur)--ligne--pos------

Elle donne des informations sur le tampon affiché dans la fenêtre : le nom du tampon, quels modes majeur et mineur sont utilisés, si le texte du tampon a été modifié, et quelle portion du tampon vous examinez.

ch contient deux étoiles '`**`' si le texte du tampon a été édité (le tampon est "modifié"), ou '`--`' si le tampon n'a pas été édité. Pour un tampon en lecture seule, c'est '`%*`' si le tampon est modifié, et '`%%`' sinon.

tampon est le nom du *tampon* de la fenêtre. Dans la plupart des cas, c'est le nom du fichier que vous éditez. See Chapter 15 [Buffers], page 185.

Le tampon affiché dans la fenêtre sélectionnée (celle où apparaît le curseur) est aussi le tampon sélectionné d'Emacs, c'est-à-dire celui où a lieu l'édition. Lorsque nous parlons de ce que fait une commande "au tampon", nous parlons du tampon couramment sélectionné.

ligne est '`L`' suivi de la ligne courante du point. C'est visible lorsque le mode de Numérotation de Ligne est actif (il l'est par défaut). Vous pouvez aussi afficher le numéro de colonne courante, en activant le mode de Numérotation de Colonne (qui n'est pas activé par défaut, car assez lent). See Section 11.10 [Optional Mode Line], page 113.

pos précise s'il reste du texte au dessus du haut de la fenêtre, ou en dessous du bas. Si votre tampon est assez petit pour être entièrement visible dans la fenêtre, *pos* est '`all`'. Sinon, c'est '`Top`' si vous regardez le début du tampon, '`Bot`' si vous regardez la fin, ou '*nn*`%`', où *nn* est le pourcentage du tampon au dessus du haut de la fenêtre.

majeur est le nom du *mode majeur* en application dans le tampon. A chaque instant, chaque tampon est dans un et seulement un des modes majeurs possibles. Les modes majeurs disponibles incluent le mode Fondamental (le moins spécialisé), le mode Texte, le mode Lisp, le mode C, le mode Texinfo, et beaucoup d'autres. See Chapter 19 [Major Modes], page 235, pour des détails sur les différences entre les modes, et comment en sélectionner un.

Certains modes majeurs affichent des informations supplémentaires après leur nom. Par exemple, les tampons Rmail affichent le numéro de message courant et le nombre total de messages. Les tampons de Compilation et de Shell affichent l'état du sous-processus.

mineur est une liste des *modes mineurs* activés à l'instant même dans le tampon de la fenêtre sélectionnée. Par exemple, '`Fill`' veut dire que le mode Auto Fill est actif. '`Abbrev`' veut dire que le mode Word Abbrev est actif. '`Ovwrt`' veut dire que le mode Overwrite est actif. See Section 31.1 [Minor Modes], page 455, pour plus d'informations. '`Narrow`' veut dire que l'édition du tampon affiché est réduite à une portion seulement de son texte. Ce n'est pas réellement un mode mineur, mais y ressemble. See Section 30.9 [Narrowing], page 443. '`Def`' veut dire qu'une macro clavier est en train d'être définie. See Section 31.3 [Keyboard Macros], page 470.

De plus, si Emacs est dans un niveau d'édition récursif, des crochets ('`[...]`') apparaissent autour des parenthèses qui entourent les noms de modes. Si Emacs est dans un niveau d'édition récursif à l'intérieur d'un autre, des crochets doubles apparaissent, etc. Etant donné que les niveaux d'édition récursifs affectent Emacs globalement, et non seulement un tampon, les crochets apparaissent dans chacune des lignes de mode ou dans aucune d'elles. See Section 30.13 [Recursive Edit], page 447.

Les terminaux sans fenêtrage peuvent seulement afficher un seul cadre Emacs à la fois (see Chapter 17 [Frames], page 203). Sur de tels terminaux, la ligne de mode affiche le nom du cadre sélectionné, après *ch*. Le nom du cadre initial est '`F1`'.

cs indique le système de codage utilisé pour le fichier que vous éditez. Un tiret indique la situation par défaut : pas de conversion de code, à l'exception de la conversion de fin-de-ligne si le contenu du fichier le nécessite. '`=`' veut dire pas de conversion du tout. Les conversions de code non triviales sont représentées par des lettres diverses — par exemple, '`1`' se réfère à l'ISO Latin-1. See Section 18.6 [Coding Systems], page 223, pour plus d'informations. Si vous utilisez une méthode d'entrée, une chaîne du style '*i*`>`' est ajoutée au début de *cs* ; *i* identifie la méthode d'entrée. (Certaines méthodes d'entrée affichent '`+`' ou '`@`' plutôt que '`>`'.) See Section 18.4 [Input Methods], page 221.

Lorsque vous utilisez un terminal texte (et non pas un système de fenêtres), *cs* utilise trois caractères pour décrire, respectivement, le système de codage pour l'entrée au clavier, le système de codage pour la sortie sur le terminal, et le système de codage pour le fichier que vous éditez.

Lorsque les caractères multi-octets ne sont pas activés, *s* n'appraît pas du tout. See Section 18.2 [Enabling Multibyte], page 219.

Les deux-points après *cs* peuvent se transformer en une autre chaîne dans certaines circonstances. Emacs utilise des caractères newline pour séparer les lignes d'un tampon. Certains fichiers utilisent des conventions différentes pour séparer les lignes : soit carriage-return linefeed (la convention MS-DOS), soit juste carriage-return (la convention Macintosh). Si le fichier du tampon utilise carriage-return linefeed, les deux-points se changent soit en barre oblique inverse ('`\`') soit en '`(DOS)`', selon le système d'exploitation. Si le fichier utilise seulement carriage-return, les deux-points se changent soit en barre oblique ('`/`') soit en '`(Mac)`'. Sur certains systèmes, Emacs affiche

‘(Unix)’ plutôt que les deux-points même pour les fichiers utilisant newline pour séparer les lignes.

Vous pouvez personnaliser l'affichage de la ligne de mode pour chacun des formats de fin de ligne en définissant chacune des variables `eol-mnemonic-unix`, `eol-mnemonic-dos`, `eol-mnemonic-mac` et `eol-mnemonic-undecided` comme une chaîne qui vous semble appropriée. See Section 31.2 [Variables], page 457, pour savoir comment définir des variables.

See Section 11.10 [Optional Mode Line], page 113, pour les possibilités d'ajouter d'autres informations pratiques à la ligne de mode, comme le numéro de colonne courante du point, l'heure, et si du courrier est arrivé pour vous.

La ligne de mode est sensible à la souris ; lorsque vous déplacez la souris sur différentes parties de la ligne, Emacs affiche un texte d'aide vous indiquant ce que produirait un clic à cet endroit. See Section 17.6 [Mode Line Mouse], page 208.

1.4 La Barre de Menu

Chaque cadre d'Emacs a normalement une *barre de menu* tout en haut que vous pouvez utiliser pour exécuter certaines opérations classiques. Il n'est pas nécessaire de les lister ici, vous pouvez les voir plus facilement par vous-même.

Lorsque vous utilisez un système de fenêtrage, vous pouvez utiliser la souris pour choisir une commande dans la barre de menu. Une flèche pointant vers la droite, après un élément de menu, indique que cet élément conduit à un menu subsidiaire ; ‘...’ à la fin veut dire que la commande va demander des arguments avant de s'exécuter.

Pour voir le nom de commande complet et la documentation pour un élément de menu, tapez `C-h k`, puis sélectionnez l'élément dans la barre de menu avec la souris, de la manière habituelle (see Section 7.1 [Key Help], page 70).

Sur les terminaux texte sans souris, vous pouvez utiliser la barre de menu en tapant `M-‘` ou ⟨F10⟩ (qui lancent la commande `tmm-menubar`). Cette commande lance un mode dans lequel vous pouvez sélectionner un élément de menu à partir du clavier. Un choix prévisionnel apparaît dans la zone de répercussion. Vous pouvez utilisez les touches de curseur haut et bas pour vous déplacer dans le menu. Lorsque vous êtes sur le bon élément, tapez ⟨RET⟩ pour le sélectionner.

Chaque élément de menu a aussi une lettre ou un chiffre assigné qui désigne cet élément ; c'est généralement l'initiale d'un mot dans le nom de l'élément. Cette lettre ou ce chiffre est séparé du nom de l'élément par ‘=>’. Vous pouvez taper la lettre ou le chiffre de l'élément pour sélectionner ce dernier.

Certaines des commandes dans la barre de menu ont aussi un raccourci clavier ; dans ce cas, le menu liste un raccourci clavier équivalent entre parenthèses après l'élément lui-même.

2 Caractères, Touches et Commandes

Ce chapitre explique les jeux de caractères utilisés par Emacs pour l'entrée de commandes et pour le contenu de fichiers, et explique aussi les concepts de *touches* et de *commandes*, fondamentaux pour comprendre comment Emacs interprète vos entrées au clavier et à la souris.

2.1 Types d'entrées utilisateur

GNU Emacs utilise une extension du jeu de caractères ASCII pour l'entrée au clavier ; il accepte aussi en entrée des événements de type autre que des caractères, comme les touches de fonction et des actions sur les boutons de souris.

Le jeu de caractères ASCII consiste en 128 codes de caractères. Certains de ces caractères sont des symboles graphiques assignés, comme '`a`' et '`=`' ; les autres sont des caractères de contrôle, comme `Control-a` (habituellement écrit `C-a` pour simplifier). `C-a` tient son nom du fait que vous l'obtenez en gardant la touche ⟨CTRL⟩ appuyée lorsque vous pressez la touche `a`.

Certains caractères de contrôle ASCII ont des noms spéciaux, et la plupart des terminaux ont des touches spéciales avec lesquelles vous pouvez les obtenir : par exemple, ⟨RET⟩, ⟨TAB⟩, ⟨DEL⟩ et ⟨ESC⟩. Le caractère espace sera par la suite souvent référencé par ⟨SPC⟩, bien qu'il soit à strictement parler un caractère graphique dont le graphique est vierge. Certains claviers ont une touche "linefeed" qui est un pseudonyme pour `C-j`.

Emacs étend le jeu de caractères ASCII avec des milliers de caractères imprimables supplémentaires (See Chapter 18 [International], page 219), des caractères de contrôle additionnels, et quelques modificateurs de plus qui peuvent être combinés avec n'importe quel caractère.

Sur des terminaux ASCII, seuls 32 caractères de contrôle sont possibles. Ce sont les variantes de contrôle des lettres et de '`@[]\^_`'. De plus, la touche shift est sans signification pour les caractères de contrôle : `C-a` et `C-A` sont les mêmes caractères, et Emacs ne peut pas les distinguer.

Mais le jeu de caractères d'Emacs a de la place pour des variantes de contrôle de tous les caractères imprimables, et pour faire la distinction entre `C-a` et `C-A`. Le système X Window rend possible l'entrée de tous ces caractères. Par exemple, `C--` (c'est-à-dire Control-Moins) et `C-5` sont des commandes Emacs significatives sous X.

Une autre extension du jeu de caractères d'Emacs est l'ajout de bits modificateurs. Un seul bit modificateur est généralement utilisé ; il est appelé Meta. Chaque caractère a une variante Meta ; comme par exemple `Meta-a` (normalement écrit `M-a` pour simplifier), `M-A` (un caractère différent de `M-a`, bien que ces deux caractères aient des significations identiques sous Emacs),

`M-`⟨RET⟩, et `M-C-a` ; logiquement parlant, l'ordre dans lequel les touches modificatrices ⟨CTRL⟩ et ⟨META⟩ sont mentionnées n'a pas d'importance.

Certains terminaux ont une touche ⟨META⟩, vous permettant de taper les caractères Meta en gardant cette touche appuyée. Ainsi, `Meta-a` est obtenu en gardant ⟨META⟩ appuyé tout en pressant `a`. La touche ⟨META⟩ fonctionne comme la touche ⟨SHIFT⟩. Une telle touche n'est toutefois pas toujours appelée ⟨META⟩, cette fonction étant souvent une option spéciale pour une touche avec une autre utilité première.

S'il n'y a pas de touche `META`, vous pouvez toujours taper les caractères Meta en utilisant une séquence de deux caractères commençant par ⟨ESC⟩. Ainsi, pour entrer `M-a`, vous pouvez taper ⟨ESC⟩ `a`. Pour entrer `C-M-a`, vous devriez taper ⟨ESC⟩ `C-a`. ⟨ESC⟩ est aussi permis sur les terminaux ayant une touche ⟨META⟩, au cas où vous auriez pris l'habitude de l'utiliser.

Le système X Window fournit plusieurs autres touches modificatrices pouvant être appliquées à n'importe quel caractère d'entrée. Ils sont appelés ⟨SUPER⟩, ⟨HYPER⟩ et ⟨ALT⟩. Nous écrirons '`s-`', '`H-`' et '`A-`' pour dire qu'un caractère utilise ces modificateurs. Ainsi, `s-H-C-x` est une abbréviation de `Super-Hyper-Control-x`. Tous les terminaux X ne fournissent pas actuellement des touches pour ces modificateurs — en fait, beaucoup de terminaux ont une touche appelée ⟨ALT⟩ qui est en réalité une touche ⟨META⟩. Les raccourcis clavier standards d'Emacs n'incluent aucun caractère avec ces modificateurs. Mais vous pouvez leur assigner une signification en personnalisant Emacs.

Les entrées au clavier incluent des touches du clavier qui ne sont pas des caractères du tout : par exemple les touches de fonction et les touches du curseur. Les boutons de souris sont aussi en dehors de la gamme des caractères. Vous pouvez modifier ces événements avec les touches modificatrices ⟨CTRL⟩, ⟨META⟩, ⟨SUPER⟩, ⟨HYPER⟩ et ⟨ALT⟩, exactement comme pour les caractères du clavier.

Les entrées caractères et les entrées non-caractère sont collectivement appelées *événements d'entrée*. See section "Input Events" in *The Emacs Lisp Reference Manual*, pour plus d'informations. Si vous ne programmez pas en Lisp, mais désirez simplement redéfinir la signification de quelques événements caractères ou non-caractères, voyez Chapter 31 [Customization], page 455.

Les terminaux ASCII ne peuvent pas en réalité envoyer à l'ordinateur autre chose que des caractères ASCII. Ces terminaux utilisent une séquence de caractères pour représenter chaque touche de fonction. Mais ceci est invisible à l'utilisateur d'Emacs, car les routines d'entrée au clavier reconnaissent ces séquences spéciales et les convertissent en événements de touche de fonction avant que d'autres parties d'Emacs puissent les voir.

2.2 Touches

Une *séquence de touches* (*touche*, pour simplifier) est une séquence d'événements d'entrée étant significative en tant qu'unité — en tant que "commande simple." Certaines séquences de commandes d'Emacs sont juste un caractère ou un événement ; par exemple, `C-f` seulement est suffisant pour avancer d'un caractère. Mais Emacs a aussi des commandes que l'on invoque avec deux événements ou plus.

Si une séquence d'événements est suffisante pour invoquer une commande, c'est une *touche complète*. Des exemples de touches complètes sont `C-a`, `X`, ⟨RET⟩, ⟨NEXT⟩ (une touche de fonction), ⟨DOWN⟩ (une touche de curseur), `C-x C-f` et `C-x 4 C-f`. Si elle n'est pas assez longue pour être complète, nous l'appellerons une *touche préfixe*. Les exemples précédents montrent que `C-x` et `C-x 4` sont des touches préfixe. Chaque séquence de touches est soit une touche complète, soit une touche préfixe.

La plupart des caractères simples constituent des touches complètes dans les raccourcis clavier standard d'Emacs. Certains d'entre eux sont des touches préfixe. Une touche préfixe se combine avec l'événement d'entrée suivant pour former une séquence de touches plus longue, qui peut elle-même être soit complète, soit préfixe. Par exemple, `C-x` est une touche préfixe, et `C-x` et l'événement d'entrée suivant se combinent pour former une séquence de touches de deux caractères. La plupart de ces séquences de touches sont complètes, comme `C-x C-f` et `C-x b`. Certaines, comme `C-x 4` et `C-x r`, sont elles-mêmes préfixe, conduisant à des séquences de touches de trois caractères. Il n'y a pas de limite à la longueur d'une séquence de touches, mais en pratique on utilise rarement des séquences plus longues que quatre événements.

Par contraste, vous ne pouvez pas ajouter des événements à une touche complète. Par exemple, la séquence de deux caractères `C-f C-k` n'est pas une touche, car `C-f` est complète à elle seule. Il est impossible de donner à `C-f C-k` une signification indépendante. `C-f C-k` forme deux séquences de touches, et non une.

Pour tout dire, les touches préfixe sous Emacs sont `C-c`, `C-h`, `C-x`, `C-x` ⟨RET⟩, `C-x @`, `C-x a`, `C-x n`, `C-x r`, `C-x v`, `C-x 4`, `C-x 5`, `C-x 6`, ⟨ESC⟩, `M-g` et `M-j`. Mais cette liste n'est pas fixée concrètement ; c'est juste du point de vue des raccourcis clavier standard d'Emacs. Si vous presonnalisez Emacs, vous pouvez définir de nouvelles touches préfixe, ou en éliminer... See Section 31.4 [Key Bindings], page 474.

Si vous définissez ou éliminez des touches préfixe, vous changez le jeu de séquences de touches possibles. Par exemple, si vous redéfinissez `C-f` comme un préfixe, `C-f C-k` devient automatiquement une touche (complète, à moins que vous ne la définissiez aussi comme préfixe). Inversement, si vous supprimez la définition du préfixe `C-x 4`, alors `C-x 4 f` (ou `C-x 4` *n'importe quoi*) n'est plus une touche.

Le fait de taper le caractère d'aide (`C-h` ou ⟨F1⟩) après un caractère préfixe affiche une liste des commandes commençant par ce préfixe. Il ex-

iste quelques caractères préfixe pour lesquels `C-h` ne marche pas — pour des raisons historiques, elles ont des significations différentes pour `C-h` qu'il serait difficile de changer. Mais ⟨F1⟩ devrait marcher pour tous les caractères préfixe.

2.3 Touches et Commandes

Ce manuel est rempli de passages vous disant ce que font certaines touches. Mais Emacs n'assigne pas de signification aux touches directement. Au lieu de cela, Emacs assigne une signification à des *commandes*, puis donne une signification aux touches en les *liant* à des commandes.

Chaque commande a un nom choisi par un programmeur. Ce nom est habituellement composé de quelques mots anglais séparés par des tirets ; par exemple, `next-line` ou `forward-word`. Une commande a aussi une *définition de fonction* qui est un programme Lisp ; c'est ce qui permet à la commande de faire ce qu'elle fait. En Lisp Emacs, une commande est un type spécial de fonction Lisp ; qui spécifie comment lire ses arguments et comment être appelée interactivement. Pour plus d'informations sur les commandes et les fonctions, voir section "What Is a Function" in *The Emacs Lisp Reference Manual*. (La définition que nous utilisons dans ce manuel est légèrement simplifiée.)

Les liens entre les touches et les commandes sont enregistrés dans diverses tables appelées *tables de touches*. See Section 31.4.1 [Keymaps], page 474.

Lorsque nous disons que "`C-n` déplace verticalement d'une ligne vers le bas", nous dissimulons une distinction hors de propos pour l'usage ordinaire mais vitale pour comprendre comment personnaliser Emacs. C'est la commande `next-line` qui est programmée pour déplacer verticalement d'une ligne vers le bas. `C-n` a cet effet *car* elle est liée à cette commande. Si vous changez le lien de `C-n` vers la commande `forward-word`, alors `C-n` déplacera d'un mot vers l'avant. Changer les liens des touches est une méthode classique de personnalisation.

Dans la suite de ce manuel, nous ignorons habituellement cette subtilité, pour garder les choses simples. Pour donner les informations nécessaires à la personnalisation, nous indiquons le nom de la commande qui fait effectivement le travail entre parenthèses après la touche qui l'exécute. Par exemple, nous dirons que "La commande `C-n` (`next-line`) déplace le point verticalement vers le bas," pour dire que `next-line` est une commande qui déplace le point verticalement vers le bas et `C-n` est une touche qui est par défaut liée à cette commande.

Maintenant que nous sommes sur le sujet de la personnalisation, il est temps de vous parler de *variables*. Fréquemment, la description d'une commande dira : "Pour changer ça, modifiez la variable `toto`." Une variable est un nom utilisé pour se rappeler une valeur. La plupart des variables documentées dans ce manuel existent simplement pour faciliter la personnalisa-

tion : une commande ou une autre partie d'Emacs consulte cette variable et réagit différemment selon la valeur à laquelle vous l'avez fixée. À moins que vous ne soyez intéressé par la personnalisation, vous pouvez ignorer les informations sur les variables. Lorsque vous serez prêt à vous y intéresser, lisez les informations de base sur les variables, et alors les informations sur les variables individuelles prendront tout leur sens. See Section 31.2 [Variables], page 457.

2.4 Jeu de Caractères pour le Texte

Le texte, dans les tampons d'Emacs, est une séquence d'octets de huit bits. Chaque octet peut contenir un unique caractère ASCII. Les caractères ASCII de contrôle (de code octal 000 à 037, et 0177) aussi bien que les caractères ASCII imprimables (codes 040 à 0176) sont permis ; toutefois des caractères de contrôle non ASCII ne peuvent pas apparaître dans un tampon. Les autres modificateurs utilisés par l'entrée au clavier, comme Meta, ne sont de même pas permis dans les tampons.

Certains caractères de contrôle ont un rôle particulier dans le texte, et ont des noms spéciaux. Par exemple, la caractère newline (code octal 012) est utilisé dans le tampon pour terminer une ligne, et le caractère tab (code octal 011) est utilisé pour indenter le texte à la colonne d'arrêt de tabulation suivante (normalement toutes les huit colonnes). See Section 11.11 [Text Display], page 115.

Des caractères imprimables non ASCII peuvent aussi apparaître dans les tampons. Lorsque les caractères multi-octets sont permis, vous pouvez utiliser n'importe quel caractère imprimable non ASCII supporté par Emacs. Leurs codes démarrent à 256, 0400 en octal, et chacun d'eux est représenté par une séquence de deux octets ou plus. See Chapter 18 [International], page 219. Les caractères simple-octet de codes 128 à 255 peuvent aussi apparaître dans des tampons multi-octets.

Si vous ne permettez pas les caractères multi-octets, vous ne pourrez utiliser qu'un alphabet de caractères non ASCII, qui tiennent tous sur un octet. Ils utilisent les codes de 0200 à 0377 (en octal). See ⟨undefined⟩ [Single-Byte Character Support], page ⟨undefined⟩.

3 Démarrer et quitter Emacs

La manière usuelle d'invoquer Emacs est depuis le shell la commande `emacs`. Emacs efface l'écran puis affiche un message d'aide initial et une notice de copyright. Certains systèmes d'exploitation ignorent toute frappe lors du démarrage d'Emacs ; et ne donnent à Emacs aucun moyen d'y remédier. Il est alors recommandé d'attendre qu'Emacs ait effacé l'écran avant de commencer à taper votre première commande d'édition.

Si vous exécutez Emacs depuis une fenêtre shell sous X Window, démarrez-le en arrière-plan avec `emacs&`. Ainsi, Emacs ne bloque pas la fenêtre shell, et vous pourrez l'utiliser pour exécuter d'autres commandes shell tandis qu'Emacs s'exécute dans sa propre fenêtre X. Vous pouvez commencer à taper des commandes Emacs dès que vous avez dirigé votre entrée clavier vers le cadre d'Emacs.

Lorsqu'Emacs démarre, il crée un tampon appelé '`*scratch*`'. C'est le tampon avec lequel vous démarrez. Le buffer '`*scratch*`' utilise le mode Lisp Interaction ; vous pouvez l'utiliser pour saisir des expressions Lisp et les évaluer, ou vous pouvez ignorer cette possibilité et simplement y griffoner. (Vous pouvez spécifier un mode majeur différent pour ce tampon en initialisant la variable `initial-major-code` dans votre fichier d'initialisation. See Section 31.7 [Init File], page 486.)

Il est possible de spécifier des fichiers à visiter, des fichiers Lisp à charger, et des fonctions à appeler, en passant à Emacs des arguments dans la ligne de commande du shell. See Appendix B [Command Arguments], page 507. Mais nous ne recommandons pas de le faire. Cette possibilité existe principalement pour une compatibilité avec d'autres éditeurs.

Beaucoup d'autres éditeurs sont prévus pour être redémarrés chaque fois que vous voulez éditer. Vous éditez un fichier puis quittez l'éditeur. La prochaine fois que vous voulez éditer soit un autre, soit le même fichier, vous devez démarrer l'éditeur de nouveau. Avec ces éditeurs, il est sensé d'utiliser un argument en ligne de commande pour dire quel fichier éditer.

Mais démarrer un nouvel Emacs chaque fois que vous voulez éditer un fichier différent n'a pas de sens. Tout d'abord, cela risque d'être fâcheusement lent. D'autre part, ce serait ne pas profiter de la capacité d'Emacs à visiter plus d'un fichier lors d'une session d'édition unique. Et cela vous ferait perdre les contextes accumulés, comme les registres, l'historique d'annulation, et le contenu du presse-papiers.

La manière recommandée d'utiliser Emacs est de le démarrer une seule fois, juste après vous être logués, et de faire toutes vos éditions dans la même session Emacs. Chaque fois que vous voulez éditer un nouveau fichier, vous le visitez avec la session existante d'Emacs, qui contient éventuellement plusieurs autres fichiers prêts à être édités. Normalement, vous ne quittez

Emacs que lorsque vous êtes prêts à vous déloguer. See Chapter 14 [Files], page 143, pour plus d'informations sur la manière de visiter plusieurs fichiers.

3.1 Quitter Emacs

Il existe deux commandes pour sortir d'Emacs car il existe deux types de sortie : *suspendre* Emacs et *détruire* Emacs.

Suspendre veut dire arrêter Emacs temporairement et rendre le contrôle à son processus parent (habituellement un shell), vous permettant de revenir plus tard à vos éditions dans la même session d'Emacs, avec les mêmes tampons, le même contenu du presse-papiers, le même historique d'annulation, etc. C'est la manière habituelle de sortir d'Emacs.

Détruire Emacs veut dire détruire la tâche Emacs. Vous pourrez redémarrer Emacs plus tard, mais vous obtiendrez un Emacs tout frais ; il n'y a aucun moyen de revenir à une session d'édition après qu'elle ait été détruite.

`C-z` Suspendre Emacs (`suspend-emacs`) ou iconifier un cadre (`iconify-or-deiconify-frame`).

`C-x C-c` Détruire Emacs (`save-buffers-kill-emacs`).

Pour suspendre Emacs, tapez `C-z` (`suspend-emacs`). Ceci vous renvoie au shell depuis lequel vous avez invoqué Emacs. Vous pourrez revenir à Emacs avec la commande shell `%emacs`, pour les shells les plus courants.

Pour les systèmes ne supportant pas la suspension de programmes, `C-z` démarre un shell fils qui communique directement avec le terminal. Emacs attend que vous quittiez ce sous-shell. (Le moyen de faire cela est certainement `C-d` ou `exit`, mais cela dépend du shell que vous utilisez.) Le seul moyen sur ces systèmes de revenir au shell depuis lequel vous avez lancé Emacs (pour vous déloguer, par exemple), est de détruire Emacs.

La suspension échoue également si vous lancez Emacs depuis un shell qui ne supporte pas la suspension de programmes, bien que le système lui-même le supporte. Dans un tel cas, vous pouvez initialiser la variable `cannot-suspend` à une valeur différente de `nil` pour forcer `C-z` à démarrer un shell fils. (On pourrait aussi décrire le shell parent d'Emacs comme "inférieur" pour empêcher de supporter correctement le contrôle des tâches, mais c'est une histoire de goût.).

Lorsqu'Emacs communique directement avec un serveur X et crée sa propre fenêtre X dédiée, `C-z` a une signification différente. Suspendre une application qui utilise sa propre fenêtre X ne serait ni sensé, ni utile. Au lieu de ça, `C-z` exécute la commande `iconify-or-deiconify-frame`, qui ferme temporairement le cadre d'Emacs sélectionné (see Chapter 17 [Frames], page 203). Revenir à une fenêtre shell se fait grâce au gestionnaire de fenêtres.

Pour détruire Emacs, tapez `C-x C-c` (`save-buffers-kill-emacs`). Une touche à deux caractères est utilisée pour rendre plus difficile sa saisie. Cette commande propose d'abord de sauvegarder les tampons de visite de fichier modifiés. Si vous ne les sauvegardez pas tous, il redemande une confirmation par `yes` avant de détruire Emacs, car tout changement non sauvegardé sera perdu à jamais. De plus, si des sous-processus sont toujours en cours d'exécution, `C-x C-c` vous demande de confirmer leur destruction, car la destruction d'Emacs entraînera la destruction immédiate de ces sous-processus.

Il n'existe aucun moyen de redémarrer une session Emacs une fois que vous l'avez détruite. Vous pouvez, cependant, vous arranger pour qu'Emacs sauvegarde certaines informations sur la session, comme quels fichiers ont été visités, le moment auquel vous le détruisez, ainsi la prochaine fois que vous démarrerez Emacs, il essaiera de visiter les mêmes fichiers et ainsi de suite. See Section 30.12 [Saving Emacs Sessions], page 446.

Le système d'exploitation attend habituellement après certains caractères spéciaux dont la signification est de détruire ou de suspendre le programme que vous exécutez. **Cette possibilité du système d'exploitation est annihilée lorsque vous êtes sous Emacs.** Les significations de `C-z` et `C-x C-c` comme touches sous Emacs sont inspirées de l'utilisation de `C-z` et `C-c` sur la plupart des systèmes d'exploitation pour stopper ou détruire un programme, mais c'est la seule relation avec le système d'exploitation. Vous pouvez personnaliser ces touches pour exécuter la commande de votre choix (see Section 31.4.1 [Keymaps], page 474).

4 Commandes d'édition élémentaires

Nous donnons maintenant les fondements sur la manière d'entrer du texte, faire des corrections, et enregistrer ce texte dans un fichier. Si tout cela est nouveau pour vous, vous pourriez apprendre plus facilement avec le tutoriel d'Emacs apprendre-en-pratiquant. Pour utiliser ce tutoriel, démarrez Emacs et tapez `Control-h t` (`help-with-tutorial`).

Pour effacer l'écran et le rafficher, tapez `C-l` (`recenter`).

4.1 Insérer du texte

Pour insérer des caractères imprimables dans le texte que vous éditez, il suffit de les taper. Ceci insère les caractères que vous tapez dans le tampon à la position du curseur (en fait, à la position du *point* ; see Section 1.1 [Point], page 23). Le curseur avance, et le texte se trouvant après le curseur avance aussi. Si le texte dans le tampon est '`BIDULE`', le curseur se trouvant avant le '`D`', et que vous tapez `XX`, vous obtenez '`BIXXDULE`', le curseur se trouvant toujours avant le '`D`'.

Pour *supprimer* le texte que vous venez d'entrer, utilisez la touche large ⟨DEL⟩, ⟨BACKSPACE⟩ ou ⟨DELETE⟩ qui est à proximité au dessus de la touche ⟨RET⟩ ou ⟨ENTER⟩. C'est la touche que vous utilisez normalement, en dehors d'Emacs, pour effacer le dernier caractère que vous avez tapé. Indépendamment du texte apparaissant sur cette touche, Emacs la reconnaît toujours comme ⟨DEL⟩, et c'est ainsi que nous l'appellerons dans ce manuel.

La touche ⟨DEL⟩ supprime le caractère *avant* le curseur. Comme conséquence, le curseur et tous les caractères après lui reculent. Si vous tapez un caractère imprimable puis tapez ⟨DEL⟩, les deux frappes s'annulent.

Sur la plupart des ordinateurs, Emacs reconnaît automatiquement quelle touche doit être ⟨DEL⟩, et la configure de cette manière. Mais dans certains cas, spécialement avec des terminaux texte, vous devrez indiquer à Emacs quelle touche utiliser pour cela. Si la touche large non loin au dessus de la touche ⟨RET⟩ ou ⟨ENTER⟩ n'efface pas en arrière, vous devez faire cela. See Section 32.2.1 [DEL Gets Help], page 493.

Un grand nombre de claviers ont une touche ⟨BACKSPACE⟩ non loin au dessus de la touche ⟨RET⟩ ou ⟨ENTER⟩, et une touche ⟨DELETE⟩ autre part. Dans ce cas, la touche ⟨BACKSPACE⟩ est ⟨DEL⟩, et la touche ⟨DELETE⟩ fait autre chose — elle supprime "en avant," supprimant le caractère après le point, celui en dessous du curseur, comme `C-d` (voir plus bas).

Pour terminer une ligne et commencer à taper sur la suivante, tapez ⟨RET⟩. Ceci insère un caractère newline dans le tampon. Si le point se trouve au milieu d'une ligne, ⟨RET⟩ coupe la ligne en deux. Taper ⟨DEL⟩ lorsque le curseur se trouve en début de ligne a pour effet de supprimer le caractère newline précédent, donc de joindre cette ligne avec la ligne précédente.

Emacs peut couper les lignes automatiquement lorsqu'elles deviennent trop longues, si vous utilisez un mode mineur spécial appelé mode de *Remplissage automatique*. See Section 21.5 [Filling], page 248, pour savoir comment utiliser le mode de Remplissage Automatique.

Si vous préférez que le texte entré remplace (écrive par-dessus) le texte existant plutôt que de le pousser vers la droite, vous pouvez lancer le mode Réécriture (Overwrite), un mode mineur. See Section 31.1 [Minor Modes], page 455.

L'insertion directe marche pour les caractères imprimables et pour ⟨SPC⟩, mais les autres caractères agissent comme des commandes d'édition et ne s'insèrent pas d'eux-mêmes. Si vous désirez insérer un caractère de contrôle ou un caractère dont le code est supérieur à l'octal 200, vous devez le *citer* en tapant précédemment le caractère `Control-q` (`quoted-insert`). (Ce nom de caractère est normalement écrit `C-q`, pour simplifier.) Il y a deux façons d'utiliser `C-q` :

- `C-q` suivi d'un caractère non graphique (même `C-g`) insére ce caractère.
- `C-q` suivi d'une séquence de chiffres octaux insére le caractère dont le code octal est celui spécifié. Vous pouvez utiliser n'importe quel nombre de chiffres octaux ; le premier caractère qui n'est pas un chiffre termine la séquence. Si le caractère terminal est ⟨RET⟩, il sert seulement à terminer la séquence ; tout autre caractère terminal est inséré après la fin de la séquence. (L'utilisation de séquences octales est inhibée en mode Réécriture non-binaire ordinaire, pour vous donner une manière d'insérer des chiffres plutôt que de réécrire avec.)

Lorsque les caractères multi-octets sont utilisés, si vous spécifiez un code dans l'intervalle 0200 à 0377 octal, `C-q` assume que vous avez l'intention d'utiliser un jeu de caractères ISO 8859-*n*, et convertit le code spécifié en code de caractère Emacs correspondant. See Section 18.2 [Enabling Multibyte], page 219. Vous choisissez *quel* jeu de caractère ISO 8859 utiliser en choisissant votre environnement de langue (see Section 18.3 [Language Environments], page 220).

Pour utiliser le système décimal ou hexadécimal plutôt que l'octal, mettez la variable `read-quoted-char-radix` à 10 ou 16. Si la base est supérieure à 10, un certain nombre de lettres, en commençant par `a`, sont utilisées comme chiffres pour entrer un code de caractère. Un argument numérique à `C-q` spécifie le nombre de copies du caractère cité à insérer (see Section 4.10 [Arguments], page 52).

Information de personnalisation : ⟨DEL⟩ dans la plupart des modes exécute la commande `delete-backward-char` ; ⟨RET⟩ exécute la commande `newline`, et les caractères imprimables auto-insérables exécutent la commande `self-insert`, qui insère le caractère qui a été utilisé pour l'invoquer. Certains modes majeurs relient ⟨DEL⟩ à d'autres commandes.

4.2 Changer l'emplacement du point

Pour aller plus loin que la simple insertion de caractères, vous devez savoir comment déplacer le point (see Section 1.1 [Point], page 23). La manière la plus simple est d'utiliser les touches du curseur, ou de cliquer le bouton gauche de la souris à l'endroit où vous voulez le placer.

Il existe aussi des caractères de contrôle et des meta-caractères pour déplacer le curseur. Certains sont équivalents aux touches de curseur (cela remonte au temps où les terminaux n'avaient pas de touches de curseur, et sont utiles sur ce type de terminal). Les autres font des choses plus sophistiquées.

`C-a` Déplace le point en début de ligne (`beginning-of-line`).

`C-e` Déplace le point en fin de ligne (`end-of-line`).

`C-f` Avance d'un caractère (`forward-char`). La touche de curseur droite fait la même chose.

`C-b` Recule d'un caractère (`backward-char`). La touche de curseur gauche a le même effet.

`M-f` Avance d'un mot (`forward-word`).

`M-b` Recule d'un mot (`backward-word`).

`C-n` Descend d'une ligne, verticalement (`next-line`). Cette commande essaie de garder une position horizontale inchangée : si vous commencez en milieu de ligne, vous finissez en milieu de ligne. La touche de curseur vers le bas fait la même chose.

`C-p` Remonte d'une ligne, verticalement (`previous-line`). La touche de curseur vers le haut a le même effet.

`M-r` Déplace le point sur la marge gauche, centré verticalement dans la fenêtre (`move-to-window-line`). Le texte ne bouge pas dans la fenêtre.

Un argument numérique indique sur quelle ligne placer le point. Les lignes sont comptées à partir du haut de la fenêtre (zéro pour la première). Un argument négatif indique un numéro de ligne en comptant à partir du bas (-1 pour la ligne du bas).

`M-<` Déplace le point au début du tampon (`beginning-of-buffer`). Avec un argument numérique n, déplace de $n/10$ de la distance au début. See Section 4.10 [Arguments], page 52, pour plus d'informations sur les arguments numériques.

`M->` Déplace le point à la fin du tampon (`end-of-buffer`).

`C-v` Fait défiler l'affichage d'un écran en avant, et déplace le point si nécessaire pour le placer sur l'écran (`scroll-up`). Ceci ne

déplace pas toujours le point, mais c'est en général pour cela qu'elle est utilisée. Si votre clavier a une touche ⟨PAGEDOWN⟩, elle fait la même chose. Les commandes de défilement sont décrites plus en détail dans Section 11.6 [Scrolling], page 109.

`M-v` Fait défiler d'un écran en arrière, et déplace le point si nécessaire pour le placer sur l'écran (`scroll-down`). Ceci ne déplace pas toujours le point, mais est généralement utilisé pour cela. La touche ⟨PAGEUP⟩ a le même effet.

`M-x goto-char`
Lit un nombre *n* et déplace le point à la position *n* du tampon. La position 1 est le début du tampon.

`M-x goto-line`
Lit un nombre *n* et déplace le point à la ligne *n*. La ligne 1 est la première ligne du tampon.

`C-x C-n` Utilise la colonne courante du point comme *colonne de destination semi-permanente* pour `C-n` et `C-p` (`set-goal-column`). Après ça, ces commandes déplacent toujours le point vers cette colonne pour chaque ligne de destination, ou aussi près que possible étant donné le contenu de la ligne. Cette colonne de destination reste active tant qu'elle n'est pas annulée.

`C-u C-x C-n`
Annule la colonne de destination. Après ça, `C-n` et `C-p` essaient de nouveau de garder la même position horizontale, que d'ordinaire.

Si vous mettez la variable `track-eol` à une valeur non `nil`, alors `C-n` et `C-p`, utilisés en fin de ligne, déplacent à la fin de la ligne de destination. Normalement, `track-eol` est `nil`. See Section 31.2 [Variables], page 457, pour voir comment redéfinir les variables comme `track-eol`.

`C-n` renvoie généralement une erreur lorsque vous l'utilisez sur la dernière ligne du tampon (exactement comme `C-p` renvoie une erreur sur la première ligne). Mais si vous définissez la variable `next-line-add-newlines` à une valeur non `nil`, `C-n` sur la dernière ligne crée une ligne additionnelle à la fin et s'y déplace.

4.3 Effacer du texte

⟨DEL⟩ Efface le caractère avant le point (`delete-backward-char`).

`C-d` Efface le caractère après le point (`delete-char`).

⟨DELETE⟩
⟨BACKSPACE⟩
: Une de ces touches, celle large placée au dessus de la touche ⟨RET⟩ ou ⟨ENTER⟩, supprime le caractère avant le point, comme ⟨DEL⟩. Si cette touche est ⟨BACKSPACE⟩, et que votre clavier a aussi ⟨DELETE⟩, alors ⟨DELETE⟩ supprime en avant, comme `C-d`.

`C-k` : Détruit la fin de la ligne (`kill-line`).

`M-d` : Détruit jusqu'à la fin du mot suivant (`kill-word`).

`M-`⟨DEL⟩ : Détruit jusqu'au début du mot précédent (`backward-kill-word`).

Vous savez déjà que la touche ⟨DEL⟩ supprime le caractère avant le point (ou avant le curseur). Une autre touche, `Control-d` (`C-d`, pour simplifier), supprime le caractère après le point (c'est-à-dire le caractère placé sous le curseur). Ceci déplace le reste du texte de la ligne vers la gauche. Si vous tapez ⟨C-d⟩ en fin de ligne, cette ligne et la ligne suivante ne forment plus qu'une.

Pour effacer un morceau de texte plus important, utilisez la touche `k`, qui détruit une ligne à la fois. Si vous tapez `C-k` en début ou en milieu de ligne, il détruit tout le texte se trouvant entre le point et la fin de ligne. Si vous tapez `C-k` en fin de ligne, il joint cette ligne avec la ligne suivante.

See Section 9.1 [Killing], page 85, pour des moyens plus flexibles de détruire du texte.

4.4 Annuler des changements

Vous pouvez annuler tous les changements faits récemment dans le texte du tampon, dans une certaine limite. Chaque tampon enregistre ses changements individuellement, et la commande d'annulation s'applique toujours au tampon courant. Habituellement, chacune des commandes d'édition crée une entrée séparée dans l'enregistrement des annulations, mais certaines commandes comme `query-replace` créent plusieurs entrées, et des commandes très simples comme `self-insert` sont souvent regroupées pour rendre l'annulation moins assommante.

`C-x u` : Annule un groupe de changements — habituellement, une commande (`undo`).

`C-_` : Même chose.

`C-u C-x u` : Annule un groupe de changements dans la région.

La commande `C-x u` ou `C-_` est le moyen d'annuler. La première fois que vous appelez cette commande, elle annule le dernier changement. Le

point se déplace à l'endroit où il se trouvait avant la commande qui a fait le changement.

Des répétitions consécutives de `C-_` ou `C-x u` annulent les changements précédents, jusqu'à la limite des informations d'annulation disponibles. Si tous les changements sauvegardés ont été annulés, la commande d'annulation imprime un message d'erreur et ne fait rien d'autre.

Toute commande différente d'une commande d'annulation termine la séquence de commandes d'annulation. À partir de ce moment, les commandes d'annulation précédentes deviennent des commandes ordinaires, qui peuvent elles-mêmes être annulées. Ainsi, pour rétablir des changements que vous auriez annulés, tapez `C-f` ou une autre commande qui terminera en douceur la séquence d'annulation, puis tapez d'autres commandes d'annulation.

Ordinairement l'annulation s'applique à tous les changements effectués dans le tampon courant. Vous pouvez aussi utiliser l'*annulation sélective*, limitée à la région courante. Pour cela, spécifiez la région de votre choix, puis lancez la commande `undo` avec un argument préfixe (la valeur n'a pas d'importance) : `C-u C-x u` ou `C-u C-_`. Ceci annule le changement le plus récent dans la région. Pour annuler des changements supplémentaires dans la même région, répétez la commande `undo` (l'argument préfixe n'est pas nécessaire). Dans le mode de Marque Transitoire, toute utilisation de `undo` lorsqu'une région est active effectue une annulation sélective ; l'argument préfixe n'est pas nécessaire.

Si vous remarquez qu'un tampon a été modifié accidentellement, le meilleur moyen de le rétablir est de taper `C-_` répétitivement jusqu'à ce que les étoiles disparaissent du début de la ligne de mode. À ce moment, toutes les modifications faites ont été annulées. Lorsqu'une commande d'annulation fait disparaître les étoiles de la ligne de mode, le contenu du tampon est identique à ce qu'il était lorsque le fichier a été chargé ou enregistré pour la dernière fois.

Si vous ne savez plus si vous avez changé le tampon délibérément, tapez `C-_` une fois. Lorsque vous annulez le dernier changement, vous devriez voir plus facilement si c'était un changement intentionnel. Si c'était un accident, laissez-le annulé. S'il était délibéré, rétablissez le changement comme décrit précédemment.

Tous les tampons n'enregistrent pas les informations d'annulation. Les tampons dont le nom commence par un espace ne le font pas ; ces tampons sont utilisés en interne par Emacs et sont des extensions pour garder des textes que l'utilisateur n'a besoin ni de voir ni d'éditer.

Vous ne pouvez pas annuler un simple déplacement du curseur ; seuls des changements sur le contenu du tampon sauvegardent des informations d'annulation. Par ailleurs, certaines commandes de déplacement du curseur placent la marque, donc si vous utilisez ces commandes de temps en temps,

vous pouvez revenir sur vos pas en parcourant la pile des marques (see Section 8.5 [Mark Ring], page 81).

Lorsque les informations d'annulation pour un tampon deviennent trop importantes, Emacs abandonne les informations les plus anciennes de temps en temps (durant la collecte des ordures). Vous pouvez spécifier le volume d'informations sauvegardées en initialisant deux variables : `undo-limit` et `undo-strong-limit`. Ces valeurs sont exprimées en octets.

La variable `undo-limit` représente une limite souple : Emacs garde assez de données d'annulation pour remplir cet espace, et peut-être le dépasser, mais ne garde plus de données pour les commandes suivantes. Sa valeur par défaut est 20000. La variable `undo-strong-limit` représente une limite stricte : la commande qui fait dépasser les données de cette limite est elle aussi abandonnée. Sa valeur par défaut est 30000.

En regard des valeurs de ces variables, le changement le plus récent n'est jamais abandonné. Il n'y a donc aucun danger qu'une collecte d'ordure arrivant juste après un important changement vous empêche d'annuler ce dernier.

La raison pour laquelle la commande `undo` a deux touches, `C-x u` et `C-_`, définies pour l'exécuter, est qu'elle mérite d'avoir une touche simple, mais qu'il est difficile sur certains claviers de taper `C-_`. `C-x u` est une alternative que vous pouvez taper sans souci sur n'importe quel terminal.

4.5 Fichiers

Les commandes décrites plus haut sont suffisantes pour créer ou altérer un texte dans un tampon d'Emacs ; les commandes d'Emacs plus avancées rendent juste les choses plus faciles. Mais pour garder un texte en permanence, vous devez le sauvegarder dans un *fichier*. Les fichiers sont des unités de texte nommées qui sont stockées pour vous par le système d'exploitation, et accessibles plus tard par leur nom. Pour examiner ou utiliser le contenu d'un fichier d'une manière quelconque, comme l'éditer avec Emacs, vous devez spécifier son nom.

Considérez un fichier appelé '`/usr/rms/toto.c`'. Sous Emacs, pour commencer à éditer ce fichier, tapez :

```
C-x C-f /usr/rms/toto.c ⟨RET⟩
```

Ici, le fichier est donné en *argument* à la commande `C-x C-f` (`find-file`). Cette commande utilise le *mini-tampon* pour lire l'argument, et vous tapez ⟨RET⟩ pour terminer la saisie de l'argument (see Chapter 5 [Minibuffer], page 55).

Emacs répond à la commande en *visitant* le fichier : ouverture d'un tampon, copie du contenu du fichier dans le tampon, puis affichage du tampon pour que vous puissiez l'éditer. Si vous altérez le texte, vous pouvez *sauvegarder* le nouveau texte dans le fichier en tapant `C-x C-s` (`save-buffer`).

Ceci rend les changements permanents en copiant le contenu altéré du tampon dans le fichier '`/usr/rms/toto.c`'. Avant que vous sauvegardiez, les changements sont seulement effectifs dans Emacs, et le fichier '`toto.c`' n'est pas altéré.

Pour créer un fichier, visitez juste le fichier avec `C-x C-f` comme s'il existait. Ceci crée un tampon vide dans lequel vous pouvez insérer le texte que vous voulez placer dans le fichier. Le fichier est effectivement créé lorsque lorsque vous sauvegardez le tampon avec `C-x C-s`.

Bien sûr, il y a encore beaucoup à apprendre sur les fichiers. See Chapter 14 [Files], page 143.

4.6 Aide

Si vous oubliez ce que fait une touche, vous pouvez le retrouver avec la touche d'aide, qui est `C-h` (ou ⟨F1⟩, qui est un pseudonyme de `C-h`). Tapez `C-h k` suivi de la touche pour laquelle vous voulez de l'aide ; par exemple, `C-h k C-n` vous dit ce que fait la touche `C-n`. `C-h` est une touche préfixe ; `C-h k` est juste une de ses sous-commandes (la commande `describe-key`). Les autres sous-commandes de `C-h` fournissent différents types d'aide. Tapez `C-h` deux fois pour obtenir une description de toutes les possibilités d'aide. See Chapter 7 [Help], page 67.

4.7 Lignes vierges

Voici des commandes et des techniques spéciales pour insérer et supprimer des lignes vierges.

`C-o` Insére une ou plusieurs lignes vierges après le curseur (`open-line`).

`C-x C-o` Supprime toutes les lignes vierges consécutives sauf une (`delete-blank-lines`).

Lorsque vous voulez insérer une nouvelle ligne de texte avant une ligne existante, vous pouvez le faire en tapant la nouvelle ligne de texte, suivi de ⟨RET⟩. Néanmoins, il est plus facile de voir ce que vous faites si vous insérez d'abord une ligne vierge, puis insérez le texte désiré dans cette ligne. Ceci est facile à faire en utilisant `C-o` (`open-line`), qui insère un caractère newline après le point, mais laisse le point avant le newline. Après `C-o`, tapez le texte pour la nouvelle ligne. `C-o T O T O` a le même effet que `T O T O` ⟨RET⟩, excepté la position finale du point.

Vous pouvez insérer plusieurs lignes vierges en tapant `C-o` plusieurs fois, ou en lui donnant un argument numérique pour lui dire combien de lignes insérer. See Section 4.10 [Arguments], page 52, pour savoir comment. Si

vous avez un préfixe de remplissage, alors la commande `C-o` insère le préfixe de remplissage dans la nouvelle ligne, lorsque vous l'utilisez en début de ligne. See Section 21.5.3 [Fill Prefix], page 251.

Un moyen facile de se débarrasser de lignes vierges en trop est d'utiliser la commande `C-x C-o` (`delete-blank-lines`). `C-x C-o` au milieu d'une suite de lignes vierges supprime toutes ces lignes sauf une. `C-x C-o` sur une ligne vierge solitaire supprime cette ligne vierge. Lorsque le point est sur une ligne non vierge, `C-x C-o` supprime toutes les lignes vierges suivant cette ligne non vierge.

4.8 Lignes de continuation

Si vous ajoutez trop de caractères à une ligne sans la couper avec ⟨RET⟩, la ligne croît pour occuper deux lignes (ou plus) sur l'écran. Sur des terminaux graphiques, Emacs indique le bouclage de lignes avec de petites flèches coudées dans les bordures gauche et droite de la fenêtre. Sur des terminaux texte, Emacs affiche un caractère '\' dans la marge droite d'une ligne d'écran si ce n'est pas le dernier de la ligne de texte. Ce caractère '\' indique que le ligne d'écran suivante n'est pas réellement une ligne distincte dans le texte, mais seulement une *continuation* d'une ligne trop longue pour rentrer dans l'écran. La continuation est aussi appelée *bouclage de ligne*.

Losque le bouclage de ligne arrive avant un caractère plus large qu'une colonne, certaines colonnes à la fin de la ligne d'écran précédente peuvent être "vides." Dans ce cas, Emacs affiche des caractères '\' supplémentaires dans les colonnes "vides," juste avant le caractère '\' indiquant la continuation.

Il est parfois utile qu'Emacs insère des caractères newline automatiquement lorsqu'une ligne devient trop longue. La continuation ne fait pas cela. Utilisez le mode Auto Fill (see Section 21.5 [Filling], page 248) si c'est ce que vous désirez obtenir.

Comme alternative à la continuation, Emacs peut afficher les lignes longues par *troncature*. Cela veut dire que tous les caractères ne tenant pas dans la largeur de l'écran ou de la fenêtre n'apparaissent pas du tout. Ils restent dans le tampon, mais sont temporairement invisibles. Sur des terminaux texte, '`*`' dans la dernière colonne vous indique que la ligne a été tronquée. Sur des systèmes à fenêtrage, une petite flèche droite dans la marge droite de la fenêtre indique une ligne tronquée.

La troncature est effective à la place de la continuation lorsque le défilement horizontal est utilisé, et optionnellement dans toutes les fenêtres côte-à-côte (see Chapter 16 [Windows], page 195). Vous pouvez autoriser ou interdire la troncature pour un tampon particulier avec la commande `M-x toggle-truncate-lines`.

See Section 11.12 [Display Custom], page 115, pour des variables additionnelles qui affectent l'affichage du texte.

4.9 Informations sur la position du curseur

Voici des commandes qui permettent d'obtenir des informations sur la taille et la position de parties du tampon, et pour compter des lignes.

`M-x what-page`
: Affiche le numéro de page du point, et le numéro de ligne dans la page.

`M-x what-line`
: Affiche le numéro de ligne dans le tampon.

`M-x line-number-mode`
: Active/désactive l'affichage automatique du numéro de ligne courant.

`M-x column-number-mode`
: Active/désactive l'affichage automatique du numéro de ligne courant ou de colonne courante. See Section 11.10 [Optional Mode Line], page 113.

`M-=`
: Affiche le nombre de lignes dans la région courante (`count-lines-region`). See Chapter 8 [Mark], page 77, pour des informations sur la région.

`C-x =`
: Affiche le code du caractère placé après le point, la position du point, et la colonne du point (`what-cursor-position`).

`M-x hl-line-mode`
: Permet ou interdit la mise en surbrillance de la ligne courante.

Il existe deux commandes qui marchent avec des numéros de ligne. `M-x what-line` calcule le numéro de ligne courant et l'affiche dans la zone de répercussion. Pour aller à un numéro de ligne donné, utilisez `M-x goto-line` ; il vous demande un numéro de ligne. Ces numéros de ligne commencent à un pour la première ligne du tampon.

Vous pouvez aussi voir le numéro de ligne dans la ligne de mode ; See Section 1.3 [Mode Line], page 26. Si vous restreignez le tampon, le numéro de ligne dans la ligne de mode est relatif à la portion accessible (see Section 30.9 [Narrowing], page 443). Par contraste, `what-line` affiche aussi bien le numéro de ligne relatif à la région restreinte que le numéro de ligne relatif au tampon entier.

Par contraste, `M-x what-page` compte les pages depuis le début du fichier, compte les lignes dans la page, et affiche les deux nombres. See Section 21.4 [Pages], page 247.

Tant que nous sommes dans ce sujet, nous devons aussi mentionner `M-=` (`count-lines-region`), qui affiche le nombre de lignes dans la région (see Chapter 8 [Mark], page 77). See Section 21.4 [Pages], page 247, pour la commande `C-x l` qui compte les lignes dans la page courante.

La commande `C-x =` (`what-cursor-position`) peut être utilisée pour obtenir la colonne où se trouve le curseur, et d'autres informations diverses sur le point. Elle affiche une ligne dans la zone de répercussion ressemblant à :

```
Char: c (0143, 99, 0x63)  point=25900 of 31877(81%)  column 53
```

(En fait, ceci est la sortie produite lorsque le point est avant le mot '`column`' dans l'exemple.)

Les quatre valeurs après '`Char:`' décrivent le caractère qui suit le point, d'abord en l'imprimant, puis en donnant son code caractère en octal, décimal et hexadécimal. Pour un caractère multi-octet non ASCII, elles sont suivies de '`ext`' et de la représentation du caractère, en hexadécimal, dans le système de codage du tampon, si ce système de codage encode le caractère correctement et en un seul octet (see Section 18.6 [Coding Systems], page 223). Si le codage du caractère est plus long qu'un octet, Emacs affiche '`ext ...`'.

'`point=`' est suivi de la position du point exprimée en nombre de caractères. Le début du tampon est compté en position 1, la position un caractère plus loin en 2, etc. Le suivant, plus grand, est le nombre total de caractères dans le tampon. Ensuite vient, entre parenthèses, la position exprimée en pourcentage de la taille totale.

'`column`' est suivi de la position horizontale du point, en colonnes depuis le bord gauche de la fenêtre.

Si le tampon a été restreint, rendant une partie de texte au début et à la fin temporairement inaccessible, `C-x =` affiche du texte additionnel décrivant la partie couramment accessible. Par exemple, il pourrait afficher ceci :

```
Char: C (0103, 67, 0x43)  point=252 of 889(28%) <231 - 599>  column 0
```

où les deux nombres supplémentaires donnent la plus petite et la plus grande position que le point peut prendre. Les caractères entre ces deux positions sont les caractères accessibles. See Section 30.9 [Narrowing], page 443.

Si le point est à la fin du tampon (ou à la fin de la partie accessible), la sortie de `C-x =` ne décrit pas un caractère après le point. La sortie ressemblerait à :

```
point=26957 of 26956(100%)  column 0
```

`C-u C-x =` affiche des informations supplémentaires sur le caractère, à la place des coordonnées dans le tampon et de la colonne : le nom du jeu de caractères et les codes identifiant le caractère dans ce jeu de caractères ; les caractères ASCII sont identifiés comme appartenant au jeu de caractères `ASCII`. De plus, le codage complet du caractère, même s'il tient sur plus d'un simple octet, est affiché après '`ext`'. Voici un exemple pour un caractère A Latin-1 avec un accent grave dans un tampon dont le système de codage est iso-2022-7bit[1] :

```
Char: À (04300, 2240, 0x8c0, ext ESC , A @) (latin-iso8859-1 64)
```

[1] Sur les terminaux supportant les caractères Latin-1, le caractère affiché après '`Char:`' est affiché comme le glyphe de A avec un accent grave.

4.10 Arguments numériques

En mathématiques et en informatique, le mot *argument* veut dire “donnée fournie à une fonction ou une opération”. Vous pouvez donner à n'importe quelle commande Emacs un *argument numérique* (aussi appelé *argumennt préfixe*). Certaines commandes interprètent l'argument comme un nombre de répétitions. Par exemple, `C-f` avec un argument de dix avance de dix caractères plutôt que d'un. Avec ces commandes, aucun argument est équivalent à un argument égal à 1. Des arguments négatifs indiquent à la plupart de ces commandes de déplacer ou d'agir dans la direction opposée.

Si votre clavier a une touche ⟨META⟩, le meilleur moyen de spécifier un argument numérique est de taper des chiffres et/ou un signe moins tout en gardant la touche ⟨META⟩ appuyée. Par exemple,

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

devrait descendre le curseur de cinq lignes. Les caractères `Meta-1`, `Meta-2`, etc. et `Meta--`, sont des touches reliées aux commandes (`digit-argument` et `negative-argument`) qui sont définies pour contribuer à un argument pour la prochaine commande. Les chiffres et `-` modifiés par Control, ou Control et Meta, spécifient aussi des arguments numériques.

Un autre moyen de spécifier un argument est d'utiliser la commande `C-u` (`universal-argument`) suivie des chiffres de l'argument. Avec `C-u`, vous pouvez taper les chiffres de l'argument sans avoir à garder de touche modificatrice enfoncée ; `C-u` marche sur tous les terminaux. Pour taper un argument négatif, tapez un signe moins après `C-u`. Un signe moins sans chiffre veut normalement dire -1.

`C-u` suivi d'un caractère autre qu'un chiffre ou que le signe moins a une signification particulière de “multiplication par quatre”. Il multiplie l'argument pour la prochaine commande par quatre. `C-u` deux fois multiplie celui-ci par seize. Ainsi, `C-u C-u C-f` avance de seize caractères. C'est un bon moyen d'avancer “rapidement”, puisqu'il avance d'environ 1/5 de la ligne pour une taille usuelle de l'écran. D'autres combinaisons utiles sont `C-u C-n`, `C-u C-u C-n` (descend d'une bonne fraction de l'écran), `C-u C-u C-o` (insère “un tas” de lignes vierges), et `C-u C-k` (détruit quatre lignes).

Certaines commandes utilisent le fait qu'elles aient un argument, et pas leur valeur. Par exemple, la commande `M-q` (`fill-paragraph`) sans argument remplit le texte ; avec un argument, elle justifie aussi le texte. (See Section 21.5 [Filling], page 248, pour plus d'informations sur `M-q`.) `C-u` seul est un moyen pratique de donner un argument à une telle commande.

Certaines commandes utilisent la valeur de l'argument comme nombre de répétitions, mais font quelque chose de particulier lorsqu'il n'y a pas d'argument. Par exemple, la commande `C-k` (`kill-line`) avec l'argument *n* détruit *n* lignes, incluant leur caractère newline terminal. Mais `C-k` sans argument est spécial : il détruit le texte jusqu'au caractère newline suivant,

ou, si le point est déjà en fin de ligne, il détruit le caractère newline lui-même. Ainsi, deux commandes `C-k` sans argument peuvent détruire une ligne non vierge, exactement comme `C-k` avec un argument de 1. (See Section 9.1 [Killing], page 85, pour plus d'informations sur `C-k`.)

Certaines commandes comprennent un `C-u` seul différemment d'un argument ordinaire. Certaines autres peuvent traiter un argument comprenant le signe moins seul différemment d'un argument de −1. Ces cas inhabituels sont décrits en temps utile ; ils le sont toujours pour des raisons de facilité d'usage de la commande.

Vosu pouvez utiliser un argument numérique pour insérer plusieurs copies d'un caractère. C'est évident sauf lorsque le caractère est un chiffre ; par exemple, `C-u 6 4 a` insère 64 copies du caractère '`a`'. Mais ceci ne marche pas pour insérer des chiffres ; `C-u 6 4 1` spécifie un argument de 641, mais n'insère rien. Pour séparer le chiffre à insérer de l'argument, tapez un autre `C-u` ; par exemple, `C-u 6 4 C-u 1` insère 64 copies du caractère '`1`'.

Nous utilisons le terme "argument préfixe" aussi bien que "argument numérique" pour insister sur le fait que vous tapez l'argument avant la commande, et pour distinguer ces arguments de ceux du mini-tampon qui viennent après la commande.

4.11 Répéter une Commande

La commande `C-x z` (`repeat`) fournit une autre méthode pour répéter plusieurs fois une commande Emacs. Cette commande répète la commande Emacs précédente, quelle qu'elle soit. La répétition d'une commande utilise les mêmes arguments qui étaient utilisés auparavant ; elle ne lit pas de nouveaux arguments à chaque fois.

Pour répéter la commande plus d'une fois, tapez des `z` additionnels : chaque `z` répète la commande une fois de plus. La répétition se termine lorsque vous tapez un caractère différent de `z`, ou pressez un bouton souris.

Par exemple, supposez que vous tapez `C-u 2 0 C-d` pour supprimer 20 caractères. Vous pouvez répéter cette commande (ainsi que son argument) trois fois de plus, pour supprimer un total de 80 caractères, en tapant `C-x z z z`. Le premier `C-x z` répète la commande une fois, et chaque `z` suivant la répète une fois de plus.

5 Le mini-tampon

Le *mini-tampon* est une facilité utilisée par les commandes d'Emacs pour lire des arguments plus complexes qu'un simple nombre. Les arguments dans le mini-tampon peuvent être des noms de fichiers, des noms de tampons, des noms de fonctions Lisp, des noms de commandes Emacs, des expressions Lisp, et bien d'autres choses encore, selon la commande qui lit l'argument. Vous pouvez utiliser les commandes d'édition usuelles dans le mini-tampon pour éditer l'argument.

Lorsque le mini-tampon est en usage, il apparaît dans la zone de répercussion, et le curseur du terminal s'y déplace. Le début de la ligne du mini-tampon affiche un *prompt* qui indique quel type d'entrée vous devez donner, et comment il doit être utilisé. Ce prompt est souvent dérivé du nom de la commande à laquelle est destiné l'argument. Le prompt se termine normalement par '`:`'.

Parfois, un *argument par défaut* apparaît entre parenthèses après les deux-points ; il fait aussi partie du prompt. Le défaut sera utilisé comme valeur d'argument si vous entrez un argument vide (par exemple en tapant juste ⟨RET⟩). Par exemple, les commandes qui lisent un nom de tampon indiquent toujours un argument par défaut, qui est le nom du tampon qui sera utilisé si vous tapez juste ⟨RET⟩.

Le moyen le plus simple de rentrer un argument dans le mini-tampon est de taper le texte que vous désirez, terminé par ⟨RET⟩ qui vous fait sortir du mini-tampon. Vous pouvez annuler la commande qui demande un argument, et sortir du mini-tampon, en tapant `C-g`.

Le mini-tampon utilisant l'espace écran de la zone de répercussion, il peut entrer en conflit avec d'autres utilisations que fait Emacs de la zone de répercussion. Voici comment Emacs résout ces conflits :

- Si une commande signale une erreur alors que vous êtes dans le mini-tampon, ceci n'annule pas le mini-tampon. Cependant, la zone de répercussion étant nécessaire à l'affichage du message d'erreur, le mini-tampon est caché pour un instant. Il réapparaît après quelques secondes, ou dès que vous tapez quelque chose.
- Si vous utilisez dans le mini-tampon une commande dont l'usage est d'afficher un message dans la zone de répercussion, comme `C-x =`, le message est affiché normalement, et le mini-tampon est caché pour un instant. Il réapparaît après quelques secondes, ou dès que vous tapez quelque chose.
- La répercussion des frappes de touches n'est pas effective lorsque le mini-tampon est utilisé.

5.1 Mini-tampons pour des noms de fichiers

Parfois, le mini-tampon, quand il apparaît, contient déjà du texte. Par exemple, lorsque vous devez entrer un nom de fichier, le mini-tampon, quand il apparaît, contient le *répertoire par défaut*, qui finit avec une barre oblique. C'est pour vous informer dans quel répertoire Emacs cherchera le fichier si vous n'en spécifiez pas un.

Par exemple, le mini-tampon pourrait démarrer avec ce contenu :

```
Find File: /u2/emacs/src/
```

où '`Find File: `' est le prompt. Taper `buffer.c` spécifie le fichier '`/u2/emacs/src/buffer.c`'. Pour trouver des fichiers dans des répertoires voisins, utilisez `..` ; ainsi, si vous tapez `../lisp/simple.el`, vous obtiendrez le fichier nommé '`/u2/emacs/lisp/simple.el`'. Alternativement, vous pouvez effacer avec `M-`⟨DEL⟩ les noms de répertoire que vous ne voulez pas (see Section 21.1 [Words], page 243).

Si vous ne voulez pas du répertoire par défaut, vous pouvez l'effacer avec `C-a C-k`. Mais vous n'avez pas besoin de l'effacer ; vous pouvez simplement l'ignorer. Insérez un nom de fichier absolu, démarrant avec une barre oblique ou un tilde, après le répertoire par défaut. Par exemple, pour spécifier le fichier '`/etc/printcap`', insérez juste ce nom, ce qui donne au mini-tampon le contenu :

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

GNU Emacs donne une signification particulière à une double barre oblique (qui n'est normalement pas une chose utile à écrire). Cela veut dire : "ignorer tout ce qui se trouve avant la seconde barre oblique." Ainsi, '`/u2/emacs/src/`' est ignoré dans l'exemple précédent, et vous obtenez le fichier '`/etc/termcap`'.

Si vous mettez `insert-default-directory` à `nil`, le répertoire par défaut n'est pas inséré dans le mini-tampon. Dans ce cas, le mini-tampon, quand il apparaît, est vide. Mais le nom que vous tapez, s'il est relatif, est toujours interprété par rapport au même répertoire par défaut.

5.2 Éditer dans le mini-tampon

Le mini-tampon est un tampon Emacs (quoiqu'un peu particulier), et les commandes Emacs usuelles sont disponibles pour éditer le texte d'un argument.

⟨RET⟩ dans le mini-tampon étant défini pour quitter ce dernier, vous ne pouvez pas l'utiliser pour insérer un caractère newline dans le mini-tampon. Pour cela, tapez `C-o` ou `C-q C-j`. (Rappelez-vous que le caractère newline n'est autre que le caractère control-J.)

Le mini-tampon a sa propre fenêtre qui a toujours sa place à l'écran, mais agit comme s'il n'y était pas lorsque le mini-tampon n'est pas actif.

Lorsque le mini-tampon est actif, sa fenêtre est comme les autres ; vous pouvez passer à une autre fenêtre avec `C-x o`, éditer du texte dans les autres fenêtres ou même peut-être visiter d'autres fichiers, avant de retourner au mini-tampon pour y entrer l'argument. Vous pouvez couper du texte dans une autre fenêtre, retourner dans le mini-tampon, puis coller le texte pour l'utiliser dans l'argument. See Chapter 16 [Windows], page 195.

Il y a cependant quelques restrictions dans l'utilisation de la fenêtre du mini-tampon. Vous ne pouvez changer de tampon dans celle-ci — le mini-tampon et sa fenêtre sont attachés en permanence. De plus, vous ne pouvez ni partager ni détruire la fenêtre du mini-tampon. Mais vous pouvez la rendre plus grande, de la manière habituelle, avec `C-x ^`.

La fenêtre du mini-tampon s'agrandit verticalement autant que nécessaire pour contenir le texte que vous placez dans le mini-tampon si `resize-mini-windows` est non `nil`. Si `resize-mini-windows` est `t`, la fenêtre est toujours redimensionnée pour contenir le texte qu'il affiche. Si `resize-mini-windows` est le symbole `grow-only`, la fenêtre est élargie seulement, jusqu'à ce qu'il devienne de nouveau vide : à ce moment il retrouve sa taille normale.

La variable `max-mini-window-height` contrôle la hauteur maximale pour redimensionner la fenêtre du mini-tampon : un nombre à virgule flottante spécifie une fraction de la hauteur du cadre ; un entier spécifie le nombre maximal de lignes ; `nil` indique de ne pas redimensionner automatiquement la fenêtre du mini-tampon. La valeur par défaut est 0.25.

Si une fois dans le mini-tampon vous lancez une commande qui affiche un texte d'aide dans une autre fenêtre, vous pouvez utiliser la commande `C-M-v` à partir du mini-tampon pour faire défiler le texte d'aide. Et ceci jusqu'à ce que vous quittiez le mini-tampon. Cette facilité est particulièrement utile si un mini-tampon de complétion vous donne une liste de complétions possibles. See Section 16.3 [Other Window], page 197.

Emacs interdit normalement la plupart des commandes utilisant le mini-tampon tant que le mini-tampon est actif. Cette règle permet de ne pas dérouter les utilisateurs novices avec des mini-tampons récursifs. Si vous désirez pouvoir utiliser de telles commandes à partir du mini-tampon, mettez la variable `enable-recursive-minibuffers` à une valeur non `nil`.

5.3 Complétion

Pour certains types d'arguments, vous pouvez utiliser la *complétion* pour entrer la valeur de l'argument. Complétion veut dire que vous tapez une partie de l'argument, puis Emacs remplit la suite et l'affiche, ou en remplit autant qu'il peut le déterminer à partir de la partie que vous avez tapée.

Lorsque la complétion est disponible, certaines touches (⟨TAB⟩, ⟨RET⟩ et ⟨SPC⟩) sont redéfinies pour compléter le texte présent dans le mini-tampon en une chaîne plus longue qu'il représente, en le comparant à un ensemble

d'*alternatives de complétion* fournies par la commande lisant l'argument. `?` est définie pour afficher une liste de complétions possibles pour ce que vous avez inséré.

Par exemple, lorsque `M-x` utilise le mini-tampon pour lire un nom de commande, il fournit une liste de tous les noms de commandes d'Emacs complets. Les touches de complétion comparent le texte du mini-tampon avec tous les noms de commandes, trouvent tous les caractères additionnels déduits de ceux déjà présents dans le mini-tampon, et rajoutent ces caractères à ceux que vous avez donnés. C'est pourquoi il est possible de taper `M-x ins` ⟨SPC⟩ `b` ⟨RET⟩ plutôt que `M-x insert-buffer` ⟨RET⟩ (par exemple).

La casse est normalement significative pour la complétion, car elle est significative dans la plupart des noms que vous pouvez compléter (noms de tampons, noms de fichiers et noms de commandes). Ainsi, '`tot`' ne se complète pas en '`Toto`'. La complétion ignore les distinctions de casse pour certains arguments dans lesquels la casse n'importe pas.

5.3.1 Exemple de complétion

Un exemple concret pourra aider ici. Si vous tapez `M-x au` ⟨TAB⟩, ⟨TAB⟩ cherche des alternatives (dans ce cas, des noms de commandes) commençant par '`au`'. Il y en a plusieurs, dont `auto-fill-mode` et `auto-save-mode` — mais elles sont toutes identiques jusqu'à `auto-`, donc le '`au`' dans le mini-tampon se change en '`auto-`'.

Si vous tapez de nouveau ⟨TAB⟩ immédiatement, il y a différentes possibilités pour le prochain caractère (il pourrait être choisi parmi '`cfilrs`') donc aucun caractère n'est ajouté ; au lieu de ça, ⟨TAB⟩ affiche une liste de toutes les complétions possibles dans une autre fenêtre.

Si vous tapez `f` ⟨TAB⟩, ce ⟨TAB⟩ voit '`auto-f`'. Le seul nom de commande commençant ainsi est `auto-fill-mode`, la complétion remplit donc le mini-tampon avec ça. Vous avez maintenant '`auto-fill-mode`' dans le mini-tampon, après avoir juste tapé `au` ⟨TAB⟩ `f` ⟨TAB⟩. Notez que ⟨TAB⟩ a cet effet car elle est reliée, dans le mini-tampon, à la commande `minibuffer-complete` lorsque la complétion est disponible.

5.3.2 Commandes de complétion

Voici une liste des commandes de complétion définies dans le mini-tampon lorsque la complétion est disponible :

⟨TAB⟩ Complète le texte dans le mini-tampon autant que possible (`minibuffer-complete`).

⟨SPC⟩ Complète le texte du mini-tampon, mais sans aller plus loin qu'un mot (`minibuffer-complete-word`).

⟨RET⟩ Envoie le texte du mini-tampon comme argument, en complétant avant si possible comme décrit ci-dessus (`minibuffer-complete-and-exit`).

`?` Affiche une liste de toutes les complétions possibles du texte dans le mini-tampon (`minibuffer-list-completions`).

⟨SPC⟩ complète comme ⟨TAB⟩, mais ne va jamais plus loin que le prochain tiret ou espace. Si vous avez '`auto-f`' dans le mini-tampon et tapez ⟨SPC⟩, la complétion trouvée est '`auto-fill-mode`', mais la complétion s'arrête après '`fill-`'. Ce qui donne '`auto-fill-`'. Un autre ⟨SPC⟩ à ce moment complète jusqu'à '`auto-fill-mode`'. ⟨SPC⟩ dans le mini-tampon lorsque la complétion est disponible exécute la commande `minibuffer-complete-word`.

Voici quelques commandes que vous pouvez utiliser pour choisir une complétion dans une fenêtre affichant une liste de complétions :

`Mouse-2` Cliquer le bouton 2 de la souris sur une complétion parmi une liste de complétions possibles choisit cette complétion (`mouse-choose-completion`). Vous utilisez normalement cette commande lorsque le point est dans le mini-tampon ; mais vous devez cliquer sur la liste des complétions, pas dans le mini-tampon lui-même.

⟨PRIOR⟩
`M-v` Taper ⟨PRIOR⟩, ⟨PAGE-UP⟩ ou `M-v` depuis le mini-tampon sélectionne la fenêtre montrant le tampon de la liste des complétions (`switch-to-completions`). C'est un préalable à l'utilisation des commandes suivantes. (Sélectionner cette fenêtre de la manière usuelle a le même effet, mais cette manière est plus commode.)

⟨RET⟩ Taper ⟨RET⟩ *depuis le tampon de la liste des complétions* choisit la complétion où se trouve le point ou juste après (`choose-completion`). Pour utiliser cette commande, vous devez d'abord vous déplacer vers la fenêtre affichant la liste des complétions.

⟨RIGHT⟩ Taper la touche de curseur droite ⟨RIGHT⟩ *depuis le tampon de la liste des complétions* déplace le point sur la complétion suivante (`next-completion`).

⟨LEFT⟩ Taper la touche de curseur gauche ⟨LEFT⟩ *depuis le tampon de la liste des complétions* déplace le point vers le début du tampon, sur la complétion précédente (`previous-completion`).

5.3.3 Complétion stricte

⟨RET⟩ peut compléter le contenu du mini-tampon de trois manières différentes, selon l'usage de l'argument.

- La complétion *stricte* est utilisée lorsque donner un argument autre qu'une des alternatives connues n'a pas de sens. Par exemple, lorsque `C-x k` demande un nom de tampon à détruire, il est absurde de donner autre chose qu'un nom de tampon existant. En complétion stricte, ⟨RET⟩ refuse de sortir du mini-tampon si le texte entré ne peut être complété en une et une seule des alternatives possibles.
- La complétion *prudente* est similaire à la complétion stricte, sauf que ⟨RET⟩ sort du mini-tampon seulement si le texte est déjà une des alternatives possibles, n'ayant pas besoin d'être complétée. Si le texte n'est pas complet, ⟨RET⟩ ne sort pas, mais complète le texte. S'il le complète entièrement, un second ⟨RET⟩ sortira du mini-tampon.

  La complétion prudente est utilisée pour lire des noms de fichiers qui doivent déjà exister.
- La complétion *tolérante* est utilisée lorsque toute chaîne est sensée, la liste des alternatives de complétion étant juste un guide. Par exemple, lorsque `C-x C-f` lit un nom de fichier à visiter, n'importe quel nom est permis, au cas où vous voudriez créer un nouveau fichier. En complétion tolérante, ⟨RET⟩ prend le texte du mini-tampon exactement comme il le trouve, sans le compléter.

Les commandes de complétion affichent une liste de toutes les complétions possibles dans une fenêtre lorsqu'il existe plus d'une alternative pour le prochain caractère. Sinon, taper ⟨?⟩ demande explicitement une telle liste. Si la liste des complétions est longue, vous pouvez la faire défiler avec `C-M-v` (see Section 16.3 [Other Window], page 197).

5.3.4 Options de complétion

Lorsque la complétion est faite sur des noms de fichiers, certains noms de fichiers sont normalement ignorés. La variable `completion-ignored-extensions` contient une liste de chaînes ; un fichier dont le nom finit par une de ces chaînes est ignoré comme complétion possible. La valeur standard de cette variable a plusieurs éléments, dont `".o"`, `".elc"`, `".dvi"` et `"~"`. L'effet est que, par exemple, '`toto`' peut se compléter en '`toto.c`' même si '`toto.o`' existe aussi. Cependant, si *toutes* les complétions possibles finissent par des chaînes "ignorées", elles ne sont pas ignorées. Les extensions ignorées ne s'appliquent pas aux listes de complétions — celles-ci mentionnent toujours toutes les complétions possibles.

Normalement, une commande de complétion qui trouve que le prochain caractère est indéterminé affiche automatiquement une liste de toutes les complétions possibles. Si la variable `completion-auto-help` est mise à `nil`, ce comportement est désaxctivé, et vous devez taper `?` pour afficher les complétions possibles.

Le mode Complétion Partielle implémente un type de complétion plus puissant qui peut compléter plusieurs mots en parallèle. Par exemple, il peut compléter l'abréviation du nom de commande `p-b` en `print-buffer`, car aucune autre commande ne commence avec deux mots dont les initiales sont '`p`' et '`b`'.

La complétion partielle de répertoires dans les noms de fichiers utilise '`*`' pour indiquer les places pour la complétion ; ainsi, '`/u*/b*/f*`' pourrait se compléter en '`/usr/bin/foo`'.

Pour activer ce mode, utilisez la commande `M-x partial-completion-mode`, ou personnalisez l'option `partial-completion-mode`. Ceci relie les commandes de complétion partielle à ⟨TAB⟩, ⟨SPC⟩, ⟨RET⟩, et `?`. Les commandes de complétion usuelle sont disponibles sur `M-`⟨TAB⟩, `M-`⟨SPC⟩, `M-`⟨RET⟩ et `M-?`.

Une autre fonction du mode Complétion Partielle est d'étendre `find-file` de telle sorte que '`<`*inclus*`>`' désigne le fichier appelé *inclus* dans un répertoire quelconque du chemin `PC-include-file-path`. Si vous réglez `PC-disable-includes` sur autre chose que `nil`, cette fonction est désactivée.

Le mode Icomplete présente un affichage constamment mis à jour qui vous indique quelles complétions sont disponibles pour le texte que vous avez entré jusque-là. Utilisez `M-x icomplete-mode` pour lancer ou arrêter ce mode mineur.

5.4 Historique du mini-tampon

Chaque argument que vous entrez dans le mini-tampon est sauvegardé dans une *liste historique du mini-tampon* ; vous pouvez donc l'utiliser plus tard pour un autre argument. Des commandes spéciales chargent le texte d'un argument précédemment entré dans le mini-tampon. Elles effacent l'ancien contenu du mini-tampon, vous pouvez donc penser à elles comme se déplaçant dans l'historique des arguments précédents.

⟨UP⟩
`M-p` Se déplacer sur l'argument précédent sauvegardé dans l'historique du mini-tampon (`previous-history-element`).

⟨DOWN⟩
`M-n` Se déplacer sur l'argument suivant sauvegardé dans l'historique du mini-tampon (`next-history-element`).

`M-r` *expression-rationnelle* ⟨RET⟩
Se déplacer sur un argument précédent, dans l'historique du mini-tampon, correspondant à *expression-rationnelle* (`previous-matching-history-element`).

`M-s` *expression-rationnelle* ⟨RET⟩
: Se déplacer sur un argument suivant, dans l'historique du mini-tampon, correspondant à *expression-rationnelle* (`next-matching-history-element`).

La manière la plus simple de réutiliser un argument sauvegardé dans la liste historique est de se déplacer dans la liste un élément à la fois. Une fois dans le mini-tampon, tapez `M-p` ou la flèche de curseur vers le haut (`previous-history-element`) pour "vous déplacer" sur l'entrée dans le mini-tampon précédente, et utilisez `M-n` ou la flèche de curseur vers le bas (`next-history-element`) pour "vous déplacer" sur l'entrée suivante.

L'entrée précédente que vous rapportez de l'historique remplace entièrement le contenu du mini-tampon. Pour l'utiliser comme argument, quittez le mini-tampon de manière classique avec ⟨RET⟩. Vous pouvez aussi éditer le texte avant de le réutiliser ; cela ne modifie pas l'élément de l'historique sur lequel vous vous êtes déplacé, mais votre nouvel argument va en fin d'historique comme il se doit.

Pour beaucoup d'arguments, il y a une valeur "par défaut". Dans certains cas, les commandes d'historique du mini-tampon connaissent la valeur par défaut. Vous pouvez alors insérer la valeur par défaut dans le mini-tampon en utilisant `M-n` pour vous déplacer "dans le futur" de l'historique. Nous espérons un jour rendre cette possibilité disponible toutes les fois où le mini-tampon a une valeur par défaut.

Il existe aussi des commandes qui recherchent en avant ou en arrière dans l'historique ; elles cherchent des éléments de l'historique qui correspondent à une expression rationnelle que vous aurez spécifiée dans le mini-tampon. `M-r` (`previous-matching-history-element`) recherche des éléments plus anciens dans l'historique, alors que `M-s` (`next-matching-history-element`) recherche dans les éléments plus récents. Par dispense spéciale, ces commandes peuvent utiliser le mini-tampon pour lire leurs arguments alors que vous êtes déjà dans le mini-tampon lorsque vous les appelez. Comme avec la recherche incrémentale, une lettre majuscule dans une expression rationnelle rend la recherche sensible à la casse.

Toutes les opérations dans le mini-tampon sont enregistrées dans une liste historique, mais il y a différentes listes historiques selon les types d'arguments. Par exemple, il y a une liste pour les noms de fichiers, utilisée par toutes les commandes attendant un nom de fichier. (Comme caractéristique spéciale, cette liste historique enregistre le nom de fichier absolu, ni plus ni moins, même si ce n'est pas de cette manière que vous avez entré le nom de fichier.)

Il y a plusieurs autres listes historiques très spécifiques, dont une pour les noms de commandes lues par `M-x`, une pour les noms de tampons, une pour les arguments de commandes comme `query-replace`, et une pour les commandes de compilation lues par `compile`. Finalement, il y a une liste historique "divers" que la plupart des arguments du mini-tampon utilisent.

La variable `history-length` spécifie la longueur maximale d'une liste historique ; une fois qu'une liste atteint cette longueur, l'élément le plus ancien est supprimé à chaque fois qu'un élément est rajouté. Si la valeur de `history-length` est `t`, cependant, la liste n'est pas limitée en longueur et les éléments ne sont jamais supprimés.

5.5 Répéter les commandes du mini-tampon

Toute commande utilisant le mini-tampon au moins une fois est enregistrée dans une liste historique spéciale, avec ses arguments ; vous pouvez ainsi réutiliser la commande entière. En particulier, tout usage de `M-x` y est enregistré, puisque `M-x` utilise le mini-tampon pour lire un nom de commande.

`C-x` ⟨ESC⟩ ⟨ESC⟩
: Réexécute une commande récente du mini-tampon (`repeat-complex-command`).

`M-x list-command-history`
: Affiche l'historique complet des commandes, montrant toutes les commandes que `C-x` ⟨ESC⟩ ⟨ESC⟩ peut répéter, les plus récentes d'abord.

`C-x` ⟨ESC⟩ ⟨ESC⟩ est utilisé pour exécuter à nouveau une commande récente utilisant le mini-tampon. Sans argument, il répète la dernière commande. Un argument numérique spécifie quelle commande répéter ; 1 veut dire la dernière, et des nombres supérieurs spécifient des commandes plus anciennes.

`C-x` ⟨ESC⟩ ⟨ESC⟩ fonctionne en transformant la commande précédente en une expression Lisp puis en entrant dans un mini-tampon initialisé avec le texte de cette expression. Si vous tapez juste ⟨RET⟩, la commande est répétée comme précédemment. Vous pouvez aussi changer la commande en éditant l'expression Lisp. L'expression que vous soumettez alors est celle qui sera exécutée. La commande répétée est ajoutée en tête de l'historique des commandes, à moins qu'elle ne soit identique à la commande la plus récemment exécutée.

Même si vous ne comprenez pas la syntaxe Lisp, vous comprendrez certainement la commande affichée pour être répétée. Si vous ne changez rien au texte, elle sera répétée exactement comme précédemment.

Une fois dans le mini-tampon pour `C-x` ⟨ESC⟩ ⟨ESC⟩, vous pouvez utiliser les commandes d'historique du mini-tampon (`M-p`, `M-n`, `M-r`, `M-s` ; see Section 5.4 [Minibuffer History], page 61) pour vous déplacer dans la liste historique des commandes enregistrées. Après avoir trouvé la commande précédente désirée, vous pouvez éditer son expression puis la soumettre avec ⟨RET⟩ comme d'habitude.

La liste des commandes précédentes utilisant le mini-tampon est sauvegardée sous forme de liste Lisp dans la variable `command-history`. Chaque élément est une expression Lisp qui décrit une commande et ses arguments. Les programmes Lisp peuvent exécuter de nouveau une commande en appelant `eval` sur l'élément de `command-history`.

6 Exécuter une commande par son nom

Chaque commande Emacs a un nom que vous pouvez utiliser pour l'exécuter. Les commandes qui sont utilisées couramment ou qui doivent être tapées rapidement sont aussi reliées à des touches — pour la commodité d'utilisation. Vous pouvez les appeler par leur nom si vous ne vous rappelez plus des touches. D'autres commandes Emacs n'ayant pas à être tapées rapidement ne sont pas reliées à des touches ; le seul moyen de les appeler est par leur nom. See Section 31.4 [Key Bindings], page 474, pour savoir comment relier des commandes à des touches.

Par convention, un nom de commande consiste en un ou plusieurs mots, séparés par des tirets ; par exemple, `auto-fill-mode` ou `manual-entry`. L'utilisation de mots anglais rend le nom de commande plus facile à retenir qu'une touche faite de caractères obscurs, bien que ce soit plus de caractères à taper.

La façon d'exécuter une commande par son nom est de commencer par `M-x`, de taper le nom de la commande, et de finir avec ⟨RET⟩. `M-x` utilise le mini-tampon pour lire le nom de la commande. ⟨RET⟩ sort du mini-tampon et exécute la commande. La chaîne '`M-x`' apparaît au début du mini-tampon comme *invite* pour vous rappeler d'entrer le nom d'une commande à exécuter. See Chapter 5 [Minibuffer], page 55, pour plus d'informations sur les possibilités du mini-tampon.

Vous pouvez utiliser la complétion pour entrer le nom de la commande. Par exemple, vous pouvez invoquer la commande `forward-char` par son nom en tapant

```
M-x forward-char ⟨RET⟩
```

ou

```
M-x forw ⟨TAB⟩ c ⟨RET⟩
```

Notez que `forward-char` est la commande que vous invoquez avec la touche `C-f`. Vous pouvez exécuter n'importe quelle commande Emacs par son nom avec `M-x`, qu'elle soit ou non reliée à une touche.

Si vous tapez `C-g` alors qu'Emacs attend le nom de la commande, vous annulez la commande `M-x` et sortez du mini-tampon, revenant à l'endroit d'où vous venez.

Pour passer un argument numérique à la commande que vous invoquez avec `M-x`, spécifiez l'argument numérique avant `M-x`. `M-x` passe son argument à la commande qu'elle exécute. La valeur de l'argument apparaît dans l'invite pendant qu'Emacs attend le nom de la commande.

Si la commande que vous tapez est reliée à une touche, Emacs le mentionne dans la zone de répercussion, deux secondes après que la commande finisse (si vous ne tapez rien avant). Par exemple, si vous tapez `M-x forward-word`, le message dit que vous pouvez exécuter la même commande

plus facilement en tapant `M-f`. Vous pouvez vous débarrasser de ces messages en mettant `suggest-key-bindings` à `nil`.

En temps normal, lorsque nous décrivons dans ce manuel une commande qui est exécutée par son nom, nous omettons le ⟨RET⟩ nécessaire pour terminer le nom. Nous parlons donc de `M-x auto-fill-mode` plutôt que de `M-x auto-fill-mode` ⟨RET⟩. Nous mentionnons ⟨RET⟩ seulement s'il est nécessaire de souligner sa présence, par exemple lorsque nous montrons une commande suivie de ses arguments.

`M-x` fonctionne en exécutant la commande `execute-extended-command`, qui a la responsabilité de lire le nom d'une autre commande et de l'invoquer.

7 Aide

Emacs fournit des possibilités d'aide avancées accessibles par un simple caractère, `C-h`. `C-h` est une touche préfixe uniquement utilisée pour des commandes d'affichage de documentation. Les caractères que vous pouvez taper après `C-h` sont appelés *options d'aide*. `C-h` lui-même peut être une option d'aide ; c'est ainsi que vous demandez de l'aide sur la façon d'utiliser `C-h`. Pour annuler, tapez `C-g`. La touche de fonction ⟨F1⟩ est équivalente à `C-h`.

`C-h C-h` (`help-for-help`) affiche une liste des options d'aide possibles, chacune accompagnée d'une brève description. Avant de taper une option d'aide, vous pouvez utiliser ⟨ESPACE⟩ ou ⟨DEL⟩ pour vous déplacer dans la liste.

`C-h` ou ⟨F1⟩ veut dire "à l'aide" dans d'autres contextes également. Par exemple, au milieu de `query-replace`, elle décrit les options disponibles pour agir sur la correspondance courante. Après une touche préfixe, elle affiche une liste des options possibles. (Certaines touches préfixe ne supportent pas `C-h`, car elles la définissent autrement, mais toutes supportent ⟨F1⟩.)

La plupart des tampons d'aide utilisent un mode majeur, le mode Aide, qui vous laisse vous déplacer avec ⟨ESPACE⟩ et ⟨DEL⟩. Il offre aussi des hyperliens vers de l'aide supplémentaire comme des références croisées, des noeuds Info, des tampons de personnalisation et ainsi de suite. See Section 7.6 [Help Mode], page 74.

Si vous recherchez une fontionnalité particulière, mais ne savez pas exactement où elle est documentée, et n'êtes même pas sûr du nom exact de la commande ou de l'option correspondante, nous recommandons d'essayer ces méthodes. Habituellement, le mieux est de commencer avec une commande apropos, puis de rechercher dans l'index du manuel, et enfin de regarder dans la FAQ et dans les mots-clés du paquetage.

`C-h a` *sujet* ⟨RET⟩
: Recherche les commandes dont le nom correspond à *sujet*, qui doit être une expression rationnelle. (see Section 12.5 [Regexps], page 125). Parcourez le tampon ouvert par Emacs, pour trouver ce que vous cherchez. See Section 7.3 [Apropos], page 71.

`M-x apropos` ⟨RET⟩ *sujet* ⟨RET⟩
: Fonctionne comme `C-h a`, mais recherche aussi dans les options utilisateur et les autres variables, pour le cas où la fonctionnalité que vous cherchez est contrôlée par une option et non par une commande. See Section 7.3 [Apropos], page 71.

`M-x apropos-documentation` ⟨RET⟩ *sujet* ⟨RET⟩
: Recherche dans les *chaînes de documentation* (les courtes descriptions) de toutes les variables et fonctions (pas leurs noms)

une correspondance de *sujet*, une expression rationnelle. See Section 7.3 [Apropos], page 71.

`C-h i m emacs` ⟨RET⟩ `i` *sujet* ⟨RET⟩
: Recherche *sujet* dans les index du manuel en ligne d'Emacs. S'il y a plusieurs correspondances, Emacs affiche la première. Vous pouvez alors presser `,` pour vous déplacer sur les correspondances suivantes, jusqu'à trouver ce que vous cherchez.

`C-h i m emacs` ⟨RET⟩ `s` *sujet* ⟨RET⟩
: Similaire, mais recherche *sujet* (qui peut être une expression rationnelle) dans le *texte* du manuel plutôt que dans ses index.

`C-h F`
: Ouvre la FAQ Emacs, où vous pouvez utiliser les commandes de recherche usuelles (see Chapter 12 [Search], page 119) por trouver l'information.

`C-h p`
: Finalement, vous pouvez essayer de rechercher le paquetage adéquat en utilisant des mots-clés appropriés pour la fonctionnalité recherchée. See Section 7.4 [Library Keywords], page 73.

Voici un résumé des commandes d'aide définies.

`C-h a` *expression-rationnelle* ⟨RET⟩
: Affiche une liste de commandes dont le nom correspond à *expression-rationnelle* (`apropos-command`).

`C-h b`
: Affiche un tableau de tous les raccourcis claviers actifs, dans cet ordre : raccourcis du mode mineur, raccourcis du mode majeur, raccourcis globaux. (`describe-bindings`)

`C-h c` *touche*
: Affiche le nom de la commande exécutée par *touche* (`describe-key-briefly`). Ici `c` veut dire "caractère". Pour plus d'informations sur *touche*, utilisez `C-h k`.

`C-h f` *fonction* ⟨RET⟩
: Affiche la documentation sur la fonction Lisp appelée *fonction* (`describe-function`). Les commandes étant des fonctions Lisp, vous pouvez donner un nom de commande.

`C-h h`
: Affiche le fichier '`hello`', qui montre des exemples de divers jeux de caractères.

`C-h i`
: Exécute Info, le programme pour parcourir les fichiers de documentation (`info`). Le manuel Emacs complet est disponible en ligne avec Info.

`C-h k` *touche*
: Affiche le nom et la documentation de la commande exécutée par *touche* (`describe-key`).

`C-h l`
: Affiche une description des 100 derniers caractères que vous avez tapés (`view-lossage`).

`C-h m` Affiche la documentation sur le mode majeur courant (`describe-mode`).

`C-h n` Affiche la documentation sur les nouveautés d'Emacs, les plus récentes d'abord (`view-emacs-news`).

`C-h P` Affiche la documentation sur les problèmes connus d'Emacs et l'éventuelle façon de les contourner (`view-emacs-problems`).

`C-h p` Trouve des paquetages par mot-clé (`finder-by-keyword`).

`C-h s` Affiche le contenu courant de la table de syntaxe (`describe-syntax`). See Section 31.6 [Syntax], page 485.

`C-h t` Entre dans le tutoriel interactif d'Emacs (`help-with-tutorial`).

`C-h v` *var* ⟨RET⟩
Affiche la documentation sur la variable Lisp *var* (`describe-variable`).

`C-h w` *commande* ⟨RET⟩
Affiche quelles touches exécutent la commande appelée *commande* (`where-is`).

`C-h C` *codage* ⟨RET⟩
Décrit le système de codage *codage* (`describe-coding-system`).

`C-h C` ⟨RET⟩
Décrit les systèmes de codage actuellement utilisés.

`C-h I` *méthode* ⟨RET⟩
Décrit une méthode d'entrée (`describe-input-method`).

`C-h L` *env-langage* ⟨RET⟩
Affiche des informations sur le jeu de caractères, les systèmes de codage et méthodes d'entrée utilisés par l'environnement de langage *env-langage* (`describe-language-environment`).

`C-h C-c` Affiche les conditions de copie de GNU Emacs.

`C-h C-d` Affiche des informations sur la manière d'obtenir de nouvelles versions de GNU Emacs.

`C-h C-f` *fonction* ⟨RET⟩
Entre dans Info et va au noeud documentant la fonction Emacs *fonction* (`Info-goto-emacs-command-node`).

`C-h C-k` *touche*
Entre dans Info et va au noeud documentant la séquence de touches *touche* (`Info-goto-emacs-key-command-node`).

`C-h C-p` Affiche des informations sur le Projet GNU.

`C-h` ⟨TAB⟩ *symbole* ⟨RET⟩
: Affiche la documentation Info sur le symbole *symbole* selon le langage de programmation que vous utilisez (`info-lookup-symbol`).

7.1 Documentation sur une Touche

Les options les plus élémentaires pour `C-h` sont `C-h c` (`describe-key-briefly`) et `C-h k` (`describe-key`). `C-h c` *touche* affiche dans la zone de répercussion le nom de la commande à laquelle *touche* est reliée. Par exemple, `C-h c C-f` affiche '`forward-char`'. Étant donné que les noms des commandes sont choisis de façon à décrire leur effet, c'est un bon moyen de savoir rapidement ce que fait *touche*.

`C-h k` *touche* est similaire mais donne plus d'informations : il affiche la chaîne de documentation de la commande en plus de son nom. Le contenu étant trop grand pour la zone de répercussion, il est affiché dans une fenêtre.

`C-h c` et `C-h k` fonctionnent pour n'importe quel type de séquences de touches, dont les touches de fonction et les événements souris.

7.2 Aide par un Nom de Commande ou de Variable

`C-h f` (`describe-function`) attend le nom d'une fonction Lisp dans le mini-tampon, puis affiche la chaîne de documentation de cette fonction dans une fenêtre. Les commandes étant des fonctions Lisp, vous pouvez utiliser cette touche pour obtenir la documentation d'une commande dont vous connaissez le nom. Par exemple,

```
C-h f auto-fill-mode ⟨RET⟩
```

affiche la documentation de `auto-fill-mode`. C'est le seul moyen d'obtenir la documentation d'une commande qui n'est reliée à aucune touche (et que vous exécutez en utilisant `M-x`).

`C-h f` est aussi utile pour des fonctions Lisp que vous envisagez d'utiliser dans un programme Lisp. Par exemple, si vous venez d'écrire l'expression `(make-vector lg)`, pour vérifier que vous utilisez correctement `make-vector`, tapez `C-h f make-vector` ⟨RET⟩. Puisque `C-h f` accepte tout nom de fonction, et pas seulement les noms de commandes, il est possible que certaines de vos abréviations favorites qui marchent avec `M-x` ne marchent plus avec `C-h f`. Une abréviation peut être unique parmi les noms de commandes mais ne plus l'être quand on ajoute les noms de fonctions à la liste des possibilités.

Le nom de la fonction à décrire par `C-h f` a une valeur par défaut, qui est utilisée si vous tapez ⟨RET⟩ en laissant le mini-tampon vide. La valeur par

défaut est la fonction appelée par l'expression Lisp la plus profonde contenant le point, *à condition* que ce soit le nom d'une fonction Lisp définie et valide. Par exemple, si le point se trouve juste après le texte '`(make-vector (car x)`', la liste la plus profonde contenant le point est celle commençant par '`(make-vector`', le comportement par défaut est donc de décrire la fonction `make-vector`.

`C-h f` est aussi utile pour seulement vérifier qu'un nom de fonction est bien orthographié. Si `C-h f` mentionne un nom du tampon comme défaut, c'est que ce nom doit être une fonction Lisp définie. Si c'est tout ce que vous vouliez savoir, tapez juste `C-g` pour annuler la commande `C-h f` et revenir à l'édition.

`C-h w` *commande* ⟨RET⟩ vous indique quelles touches sont reliées à *commande*. Elle affiche une liste de touches dans la zone de répercussion. Si elle dit que cette commande n'est reliée à aucune touche, vous devez utiliser `M-x` pour l'exécuter. `C-h w` exécute la commande `where-is`.

`C-h v` (`describe-variable`) est similaire à `C-h f` mais décrit des variables Lisp, plutôt que des fonctions Lisp. La valeur par défaut est le symbole Lisp autour ou avant le point, mais seulement si c'est le nom d'une variable Lisp connue. See Section 31.2 [Variables], page 457.

Les tampons d'aide décrivant des variables ou des fonctions définies en Lisp ont normalement un hyperlien vers la définition Lisp, si vous avez installé les fichiers sources de Lisp. Si vous connaissez Lisp, ceci fournit la documentation ultime. Si vous ne connaissez pas Lisp, vous devriez l'apprendre. Si vous traitez Emacs comme un fichier objet, alors vous *utilisez* seulement Emacs. Pour une réelle intimité avec Emacs, vous devez lire le code source.

7.3 À Propos

Vous pouvez poser un autre style de question plus sophistiqué, comme : "Quelles sont les commandes pour travailler avec les fichiers ?" Pour poser cette question, tapez `C-h a file` ⟨RET⟩, qui affiche une liste de tous les noms de commandes contenant '`file`', dont `copy-file`, `find-file`, etc. Avec chaque nom de commande apparaît une description concise de la manière d'utiliser la commande, et les touches à utiliser pour l'invoquer. Par exemple, elle pourrait dire que vous pouvez invoquer la commande `find-file` en tapant `C-x C-f`. Le `a` dans `C-h a` veut dire "À propos". `C-h a` exécute la commande `apropos-command`. Cette commande vérifie normalement seulement les commandes (fonctions interactives) ; si vous spécifiez un argument préfixe, elle vérifie aussi parmi les fonctions non interactives.

`C-h a` ne recherchant que les fonctions dont les noms contiennent la chaîne spécifiée, vous devez faire preuve d'ingéniosité dans le choix de la chaîne. Si vous cherchez des commandes pour couper le texte précédent (NdT : kill backwards en anglais) et que `C-h a kill-backwards` ne révèle rien, ne

désespérez pas. Essayez seulement `kill`, ou seulement `backwards`, ou seulement `back`. Persistez. Notez aussi que vous pouvez utiliser une expression rationnelle pour plus de flexibilité. (see Section 12.5 [Regexps], page 125).

Voici un ensemble d'arguments à passer à `C-h a` qui couvre un grand nombre de classes de commandes Emacs. En vous donnant une idée des conventions de nom, cet ensemble devrait aussi vous servir à développer une technique pour trouver des chaînes `apropos`.

> 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.

Pour lister toutes les variables utilisateur qui correspondent à une expression rationnelle, utilisez la commande `M-x apropos-variable`. Cette commande affiche seulement les variables utilisateur et les options de personnalisation par défaut ; si vous spécifiez un argument préfixe, elle parcourt toutes les variables.

Pour obtenir une liste de *tous* les symboles contenant une correspondance pour une expression rationnelle, et pas seulement ceux définis comme commandes, utilisez la commande `M-x apropos` plutôt que `C-h a`. Cette commande ne regarde pas parmi les raccourcis clavier par défaut ; spécifiez un argument numérique pour qu'elle le fasse.

La commande `apropos-documentation` est semblable à `apropos` à ceci près qu'elle recherche une correspondance de chaînes dans les documentations des symboles aussi bien que dans leurs noms.

La commande `apropos-value` est semblable à `apropos` à ceci près qu'elle recherche une correspondance de chaînes parmi les valeurs des symboles. Cette commande ne parcourt pas les définitions des fonctions ou les listes de propriétés par défaut ; spécifiez un argument numérique pour qu'elle le fasse.

Si la variable `apropos-do-all` est non `nil`, les commandes suivantes fonctionnent toutes comme si elles avaient reçu un argument préfixe.

Si vous voulez plus d'informations sur la définition d'une fonction, d'une variable ou sur la propriété d'un symbole listé dans le tampon Apropos, vous pouvez cliquer dessus avec `Souris-2` ou vous y déplacer et taper ⟨RET⟩.

7.4 Recherche par mot-clé de bibliothèques Lisp

La commande `C-h p` vous permet de rechercher par mot-clé parmi les bibliothèques Lisp d'Emacs. Voici une liste partielle de mots-clés que vous pouvez utiliser :

> abbrev — abréviations, raccourcis, macros.

bib — processeur de bibliographies `bib`.
c — langages C et C++.
calendar — calendrier et gestion du temps.
comm — communications, réseaux, accès distants à des fichiers.
data — support pour éditer des fichiers de données.
docs — support pour la documentation Emacs.
emulations — émulations d'autres éditeurs.
extensions — extensions du langage Emacs Lisp.
faces — support pour utiliser les faces (polices et couleurs ; see Section 11.1 [Faces], page 103).
frames — support pour les cadres Emacs et les systèmes de fenêtres.
games — jeux, blagues et divertissements.
hardware — support pour interfacer du matériel exotique.
help — support pour des systèmes d'aide en ligne.
hypermedia — support pour liens hyper-textes, et autres types de média.
i18n — internationalisation et support de jeux de caractères alternatifs.
internal — code interne Emacs, build process, défauts.
languages — modes spécialisés pour éditer des langages de programmation.
lisp — support pour l'utilisation de Lisp (dont Emacs Lisp).
local — bibliothèques locales à votre site.
maint — aide à la maintenance pour le groupe de développement Emacs.
mail — modes pour le courrier électronique.
matching — recherche et correspondance de chaînes.
news — support pour la lecture et l'envoi de news.
non-text — support pour l'édition de fichiers non texte.
oop — support pour la programmation orientée objet.
outlines — hierarchical outlining.
processes — processus, sous-shell, compilation, contrôle de travaux.
terminals — support de terminaux.
tex — support du processeur TEX.
tools — outils de programmation.
unix — interfaces/assistants pour, ou émulation de, fonctionnalités Unix.
vms — support de VMS.
wp — traitement de texte.

7.5 Aide pour le Support de Langues Étrangères

Vous pouvez utiliser la commande `C-h L` (`describe-language-environment`) pour trouver un environnement de langue donné. See Section 18.3 [Language Environments], page 220. Elle vous dit pour quelles langues on utilise cet environnement de langue, liste les jeux de caractères, les systèmes de codage et les méthodes d'entrée proposés par cet environnement. Il montre aussi du texte en exemple pour illustrer les scripts.

La commande `C-h h` (`view-hello-file`) affiche le fichier '`etc/HELLO`', qui montre comment dire "bonjour" en différentes langues.

La commande `C-h I` (`describe-input-method`) affiche des informations à propos des méthodes d'entrée ; soit pour une méthode d'entrée donnée, ou par défaut pour la méthode d'entrée actuellement utilisée. See Section 18.4 [Input Methods], page 221.

La commande `C-h C` (`describe-coding-system`) affiche des informations sur les systèmes de codage ; soit pour un système de codage donné, soit pour celui actuellement utilisé. See Section 18.6 [Coding Systems], page 223.

7.6 Commandes du Mode Aide

Les tampons d'aide donnent accès aux commandes du mode Vue (see Section 14.10 [Misc File Ops], page 181), plus un certain nombre de commandes spéciales propres aux tampons d'aide.

⟨SPC⟩ Défilement vers l'avant.

⟨DEL⟩
⟨BS⟩ Défilement vers l'arrière. Sur certains claviers, cette touche est connue comme ⟨BS⟩ ou ⟨backspace⟩.

⟨RET⟩ Suit la référence croisée sur laquelle se trouve le point.

⟨TAB⟩ Déplace le point vers la référence croisée suivante.

`S-`⟨TAB⟩ Déplace le point vers la référence croisée précédente.

`Mouse-2` Suit la référence croisée sur laquelle vous cliquez.

Lorsqu'un nom de commande (see Chapter 6 [Running Commands by Name], page 65) ou un nom de variable (see Section 31.2 [Variables], page 457) apparaît dans la documentation, il apparaît normalement entre des paires de guillemets simples. Vous pouvez cliquer sur ce nom avec `Mouse-2`, ou y déplacer le point et taper ⟨RET⟩, pour voir la documentation de cette commande ou variable. Utilisez `C-c C-b` pour revenir sur vos pas.

Il existe des commandes pour déplacer le point vers des références croisées dans le texte d'aide. ⟨TAB⟩ (`help-next-ref`) déplace le point vers la référence croisée suivante. `S-`⟨TAB⟩ déplace le point vers la référence croisée précédente (`help-previous-ref`).

7.7 Autres Commandes d'Aide

`C-h i` (`info`) exécute le programme Info, utilisé pour naviguer dans des fichiers de documentation structurés. Le manuel Emacs entier est disponible sous Info. À long terme, toute la documentation du système GNU y sera

disponible. Tapez `h` après être entré dans Info pour exécuter un tutoriel sur la façon d'utiliser Info.

Si vous spécifiez un argument numérique, `C-h i` demande le nom d'un fichier de documentation. De cette manière, vous pouvez parcourir un fichier qui n'a pas d'entrée dans le menu Info de plus haut niveau. C'est aussi pratique lorsque vous voulez obtenir de la documentation rapidement et connaissez le nom exact du fichier.

Il y a deux commandes d'aide spéciales pour accéder à la documentation d'Emacs par Info. `C-h C-f` *fonction* ⟨RET⟩ lance Info et va directement à la documentation de la fonction Emacs *fonction*. `C-h C-k` *touche* lance Info et va directement à la documentation de la touche *touche*. Ces deux touches exécutent les commandes `Info-goto-emacs-command-node` et `Info-goto-emacs-key-command-node`.

En éditant un programme, si vous avez une version Info du manuel du langage de programmation, vous pouvez utiliser la commande `C-h C-i` pour trouver la documentation sur un symbole (mot-clé, fonction ou variable). Les détails sur la manière dont marche cette fonction dépendent du mode majeur.

Si quelque chose d'étrange arrive, et que vous n'êtes pas sûr des commandes que vous avez tapées, vous pouvez utiliser `C-h l` (`view-lossage`). `C-h l` affiche les 100 derniers caractères de commande que vous avez tapés. Si vous voyez des commandes que vous ne connaissez pas, vous pouvez utiliser `C-h c` pour découvrir ce qu'elles font.

Emacs a de nombreux modes majeurs, chacun d'eux redéfinissant certaines touches et changeant quelques propriétés d'édition. `C-h m` (`describe-mode`) affiche la documentation sur le mode majeur courant, qui décrit normalement toutes les commandes modifiées par ce mode.

`C-h b` (`describe-bindings`) et `C-h s` (`describe-syntax`) présentent d'autres informations sur le mode Emacs courant. `C-h b` affiche une liste de tous les raccourcis clavier actuellement activés ; les raccourcis locaux définis par le mode majeur courant en premier, puis les raccourcis globaux (see Section 31.4 [Key Bindings], page 474). `C-h s` affiche le contenu de la table de syntaxe, avec des explications sur la syntaxe de chaque caractère (see Section 31.6 [Syntax], page 485).

Vous pouvez obtenir une liste similaire pour une touche préfixe particulière en tapant `C-h` après la touche préfixe. (Cela ne marche pas pour certaines touches préfixe : celles qui fournissent leur propre raccourci pour `C-h`. L'une d'entre elles est ⟨ESC⟩, car ⟨ESC⟩ `C-h` est `C-M-h`, qui marque un defun.

Les autres options de `C-h` affichent des fichiers divers d'informations utiles. `C-h C-w` affiche les détails sur l'absence complète de garantie pour GNU Emacs. `C-h n` (`view-emacs-news`) affiche le fichier '`emacs/etc/NEWS`', qui contient des informations sur les nouveautés d'Emacs, classées chronologiquement. `C-h F` (`view-emacs-FAQ`) affiche la "Foire Aux Ques-

tions" sur Emacs. `C-h t` (`help-with-tutorial`) affiche le tutoriel Emacs apprendre-en-pratiquant. `C-h C-c` (`describe-copying`) affiche le fichier '`emacs/etc/COPYING`', qui vous donne les conditions sous lesquelles vous pouvez distribuer des copies d'Emacs. `C-h C-d` (`describe-distribution`) affiche le fichier '`emacs/etc/DISTRIB`', vous disant comment vous procurer la dernière version d'Emacs. `C-h C-p` (`describe-project`) affiche diverses informations sur le Projet GNU. `C-h P` (`view-emacs-problems`) affiche le fichier '`emacs/etc/PROBLEMS`', qui liste les problèmes connus avec Emacs dans diverses situations et les solutions ou contournements possibles dans la plupart des cas.

7.8 Aide sur le Texte Actif et Tooltips

Lorsqu'une région de texte est "active", et que vous pouvez la sélectionner avec la souris ou avec une touche comme `RET`, elle a souvent un texte d'aide associé. Les zones de la ligne de mode en sont un exemple. Cette aide sera normalement affichée dans la zone d'écho lorsque vous déplacez le point sur le texte actif. Dans un système de fenêtres, vous pouvez afficher le texte d'aide comme "bulle d'aide". See Section 17.18 [Tooltips], page 217.

8 La Marque et la Région

Beaucoup de commandes Emacs opèrent sur une partie contiguë arbitraire du tampon courant. Pour indiquer à une telle commande sur quel texte opérer, vous positionnez *la marque* à une extrémité de ce texte, et déplacez le point à l'autre extrémité. Le texte compris entre le point et la marque est appelé *la région*. Quand vous avez défini une région, Emacs la met en surbrillance si vous lancez le mode de Marque Transitoire. (see Section 8.2 [Transient Mark], page 78).

Vous pouvez déplacer le point ou la marque pour ajuster les limites de la région. Peu importe lequel est le premier chronologiquement, ou lequel vient en premier dans le texte. Une fois que la marque a été positionnée, elle reste à cette place jusqu'à ce que vous ne la placiez à un autre endroit. Chaque tampon d'Emacs a sa propre marque, ainsi lorsque vous revenez dans un tampon précédemment sélectionné, sa marque est restée au même endroit qu'avant.

Beaucoup de commandes insérant du texte, comme `C-y` (`yank`) et `M-x insert-buffer`, positionnent le point et la marque aux extrémités du texte inséré. La région contient ainsi le texte venant d'être inséré.

À part délimiter la région, la marque peut aussi servir à se rappeler un endroit où vous voulez revenir plus tard. Pour rendre cette fonction plus utile, chaque tampon enregistre les 16 emplacements précédents de la marque dans la *pile des marques*.

8.1 Positionner la marque

Voici quelques commandes pour positionner la marque :

`C-`⟨SPC⟩ Place la marque là où est le point (`set-mark-command`).

`C-@` Même chose.

`C-x C-x` Échange la marque et le point (`exchange-point-and-mark`).

`Drag-Mouse-1`
Place le point et la marque aux extrémités du texte parcouru.

`Mouse-3` Place la marque là où est le point, et déplace le point là où vous cliquez (`mouse-save-then-kill`).

Par exemple, supposez que vous vouliez convertir une partie du tampon en majuscules, avec la commande `C-x C-u` (`upcase-region`), qui opère sur le texte dans la région. Vous pouvez d'abord aller au début du texte à mettre en capitales, taper `C-`⟨SPC⟩ pour y placer la marque, aller à la fin et taper `C-x C-u`. Ou bien, vous pouvez placer la marque à la fin du texte, aller au début, puis taper `C-x C-u`.

La manière la plus courante de placer la marque est d'utiliser la commande `C-`⟨SPC⟩ (`set-mark-command`). Elle place la marque à l'endroit du point. Vous pouvez alors déplacer le point, laissant la marque là où elle est.

Il y a deux façons de placer la marque avec la souris. Vous pouvez déplacer la souris sur une zone de texte avec le bouton 1 appuyé ; le point est alors placé là où vous relâchez le bouton, et la marque à l'autre extrémité. Ou bien vous pouvez cliquer le bouton 3 de la souris qui place la marque sur le point (comme `C-`⟨SPC⟩) puis déplace le point (comme `Mouse-1`). Ces deux méthodes copient la région dans le presse-papiers en plus de placer la marque ; ce qui donne un comportement compatible à d'autres applications fenêtrées, mais si vous ne voulez pas modifier le presse-papiers, vous devez utiliser les commandes clavier pour placer la marque. See Section 17.1 [Mouse Commands], page 203.

Les terminaux classiques ne possédant qu'un seul curseur, Emacs n'a pas la possibilité de vous rappeler l'endroit où se trouve la marque. Vous devez vous en souvenir. La solution la plus courante à ce problème est de placer la marque et de l'utiliser rapidement, avant d'oublier où elle se trouve. Alternativement, vous pouvez voir où se trouve la marque avec la commande `C-x C-x` (`exchange-point-and-mark`) qui échange les positions de la marque et du point. Le contenu de la région est inchangé, mais le curseur et le point sont maintenant là où était auparavant la marque. Dans le mode de Marque Transitoire, cette commande réactive la marque.

`C-x C-x` est aussi utile lorsque vous êtes satisfait de la position du point mais voulez déplacer l'autre extrémité de la région (où se trouve la marque) ; faites `C-x C-x` pour placer le point à cette extrémité de la région, puis déplacez-le. Une seconde utilisation de `C-x C-x`, si nécessaire, place la marque à la nouvelle position et ramène le point à sa position d'origine.

Pour plus de facilités vous permettant d'aller aux marques précédemment placées, voir Section 8.5 [Mark Ring], page 81.

Il n'y a pas de caractère `C-`⟨SPC⟩ en ASCII ; lorsque vous tapez ⟨SPC⟩ tout en gardant ⟨CTRL⟩ appuyé, vous obtenez sur la plupart des terminaux classiques le caractère `C-@`. Cette touche est reliée à `set-mark-command`. Mais à moins que vous ne soyez assez malchanceux pour avoir un terminal où vous n'obtenez pas `C-@` en tapant `C-`⟨SPC⟩, vous devez penser à ce caractère comme à `C-`⟨SPC⟩. Sous X, `C-`⟨SPC⟩ est un caractère distinct, mais est toujours relié à `set-mark-command`.

8.2 Mode de Marque Transitoire

Sur un terminal qui supporte les couleurs, Emacs peut mettre en surbrillance la région courante. Mais normalement il ne le fait pas. Pourquoi non ?

Mettre la région en surbrillance lorsqu'elle existe n'est pas désirable sous Emacs, car une fois que vous avez placé une marque, il y a *toujours* une région (dans ce tampon). Et mettre en surbrillance tout le temps serait nuisible. C'est pour cette raison qu'Emacs met en surbrillance une région seulement immédiatement après que vous l'ayez sélectionnée avec la souris.

Vous pouvez utiliser la mise en surbrillance de la région avec le mode Marque Transitoire. C'est un mode d'opération plus rigide dans lequel la région "reste" temporairement seulement, vous devrez donc indiquer une région pour chaque commande qui en utilise une. Dans le mode de Marque Transitoire, la plupart du temps aucune région n'est définie ; dans ce cas il est commode de mettre en surbrillance la région lorsqu'elle existe, et ce comportement n'est pas ennuyeux.

Pour lancer le mode de Marque Transitoire, tapez `M-x transient-mark-mode`. Cette commande démarre ou arrête le mode, vous pouvez alors répéter la commande pour sortir du mode.

Voici les détails du mode de Marque Transitoire :

- Pour placer la marque, tapez `C-`⟨SPC⟩ (`set-mark-command`). La marque est alors rendue active ; en déplaçant le point, vous verrez la région mise en surbrillance grandir et diminuer.
- Les commandes souris pour spécifier la marque la rendent elles aussi active. De même pour les commandes clavier pour spécifier une région, dont `M-@`, `C-M-@`, `M-h`, `C-M-h`, `C-x C-p` et `C-x h`.
- Lorsque la marque est active, vous pouvez exécuter des commandes qui opèrent sur la région, comme copier, indenter, ou écrire dans un fichier.
- Un changement dans le tampon, comme l'insertion ou la suppression d'un caractère, désactive la marque. a veut dire que toute commande suivante opérant sur une région échouera et refusera de s'exécuter. Vous pouvez rendre la région à nouveau active en tapant `C-x C-x`.
- Les commandes comme `M->` et `C-s` qui "laissent la marque derrière eux" en plus de leur fonction première n'activent pas la nouvelle marque. Vous pouvez activer la nouvelle région en exécutant `C-x C-x` (`exchange-point-and-mark`).
- `C-s` lorsque la marque est active n'altère pas la marque.
- Quitter avec `C-g` désactive la marque.
- Certaines commandes opèrent sur la région lorsqu'elle est active. Par exemple, `C-x u` dans le mode de Marque Transitoire opère sur la région lorsqu'une région existe. En dehors du mode de Marque Transitoire, vous devez taper `C-u C-x u` si vous désirez opérer sur la région. See Section 4.4 [Undo], page 45. Les autres commandes opérant de cette manière sont identifiées comme telles dans leur propre documentation.

La mise en surbrillance utilise la face de `région` ; vous pouvez personnaliser la façon dont la région est mise en valeur en changeant cette face. See Section 31.2.2.3 [Face Customization], page 463.

Lorsque différentes fenêtres contiennent un même tampon, elles peuvent avoir des régions différentes, car elles peuvent avoir différentes valeurs pour le point (bien qu'elles partagent toutes une même position de la marque). Ordinairement, seule la fenêtre sélectionnée met sa région en surbrillance (see Chapter 16 [Windows], page 195). Si la variable `highlight-nonselected-windows` est non-`nil`, alors toutes les fenêtres met sa région en surbrillance (à condition que le mode de Marque Transitoire soit actif et que la marque soit active).

Lorsque le mode de Marque Transitoire est désactivé, chaque commande positionnant la marque l'active aussi, et elle n'est jamais désactivée.

Si la variable `mark-even-if-inactive` est non-`nil` dans le mode de Marque Transitoire, les commandes peuvent utiliser la marque et la région même si elles sont inactives. La mise en surbrillance apparaît et disparaît comme d'habitude avec le mode de Marque Transitoire, mais la marque n'est pas réellement désactivée lorsque la mise en surbrillance disparaît.

Le mode de Marque Transitoire est parfois aussi appelé "mode Zmacs" car l'éditeur Zmacs sur la Machine Lisp du MIT gérait le point d'une manière similaire.

8.3 Opérer sur la Région

Une fois que vous avez une région et que la marque est active, voici quelques moyens d'opérer sur la région :

- Coupez-la avec `C-w` (see Section 9.1 [Killing], page 85).
- Sauvez-la dans un registre avec `C-x r s` (see Chapter 10 [Registers], page 97).
- Sauvez-la dans un tampon ou un fichier (see Section 9.3 [Accumulating Text], page 92).
- Changez la casse avec `C-x C-l` ou `C-x C-u` (see Section 21.6 [Case], page 254).
- Indentez-la avec `C-x` ⟨TAB⟩ ou `C-M-\` (see Chapter 20 [Indentation], page 239).
- Alignez le texte avec `M-x fill-region` (see Section 21.5 [Filling], page 248).
- Imprimez-la avec `M-x print-region` (see Section 30.5 [Hardcopy], page 438).
- Évaluez-la comme code Lisp avec `M-x eval-region` (see Section 23.8 [Lisp Eval], page 340).

La plupart des commandes opérant sur le texte de la région contiennent le mot `region` dans leur nom.

8.4 Commandes pour Marquer des Objets Texte

Voici les commandes pour placer le point et la marque autour d'un objet texte comme un mot, une liste, un paragraphe ou une page.

`M-@` Place la marque après la fin du mot suivant (`mark-word`). Cette commande, ainsi que la suivante, ne déplace pas le point.

`C-M-@` Place la marque après la fin d'expression Lisp suivante (`mark-sexp`).

`M-h` Place la région autour du paragraphe courant (`mark-paragraph`).

`C-M-h` Place la région autour du defun Lisp courant (`mark-defun`).

`C-x h` Place la région autour du tampon entier (`mark-whole-buffer`).

`C-x C-p` Place la région autour de la page courante (`mark-page`).

`M-@` (`mark-word`) place la marque à la fin du mot suivant, alors que `C-M-@` (`mark-sexp`) la place à la fin de l'expression Lisp suivante. Ces commandes acceptent des arguments, comme `M-f` et `C-M-f`.

D'autres commandes placent aussi bien le point que la marque, pour délimiter un objet du tampon. Par exemple, `M-h` (`mark-paragraph`) déplace le point au début du paragraphe qui contient ou suit le point, et place la marque à la fin de ce paragraphe (see Section 21.3 [Paragraphs], page 246). Elle prépare la région pour que vous puissiez indenter, changer la casse, ou couper un paragraphe entier.

`C-M-h` (`mark-defun`) place de manière similaire le point avant et la marque après le defun courant ou suivant (see Section 22.4 [Defuns], page 277). `C-x C-p` (`mark-page`) place le point avant la page courante, et la marque à la fin. (see Section 21.4 [Pages], page 247). La marque est placée après le délimiteur de la page courante (pour l'inclure), alors que le point est placé après le délimiteur de la page précédente (pour l'exclure). Un argument numérique spécifie une page suivante (si positif) ou précédente (si négatif) plutôt que la page courante.

Finalement, `C-x h` (`mark-whole-buffer`) place le tampon entier dans la région, en plaçant le point au début et la marque à la fin.

Dans le mode de Marque Transitoire, toutes ces commandes activent la marque.

8.5 La Pile des Marques

En plus de délimiter une région, la marque peut aussi servir pour se rappeler d'un endroit auquel vous voulez revenir plus tard. Pour rendre cette possibilité plus intéressante, chaque tampon se souvient des 16 locations

précédentes de la marque, dans la *pile des marques*. Les commandes qui placent la marque placent aussi l'ancienne marque dans cette pile. Pour retourner à un endroit marqué, utilisez `C-u C-`⟨SPC⟩ (ou `C-u C-@`) ; c'est la commande `set-mark-command` à laquelle on passe un argument. Elle déplace le point à l'endroit où était la marque, et replace la marque selon la pile des marques précédentes. Ainsi, la répétition de cette commande déplace le point à toutes les anciennes marques présentes dans la pile, une par une. Les positions des marques que vous passez ainsi ne sont pas perdues ; elles sont placées en fin de pile.

Chaque tampon a sa propre pile des marques. Toutes les commandes d'édition utilisent la pile des marques du tampon courant. En particulier, `C-u C-`⟨SPC⟩ reste toujours dans le même tampon.

Un grand nombre de commandes déplaçant le point sur de longues distances, comme `M-<` (`beginning-of-buffer`), commencent par placer la marque et sauver l'ancienne dans la pile des marques. Ceci pour vous permettre de revenir plus tard à l'endroit initial plus facilement. Les commandes de recherche placent la marque si elles déplacent le point. Vous pouvez voir lorsqu'une commande place la marque, car elle affiche '`Mark Set`' dans la zone de répercussion.

Si vous voulez revenir au même endroit encore et encore, la pile des marques peut ne pas être assez appropriée. Dans ce cas, vous pouvez enregistrer la position dans un registre pour le récupérer ultérieurement. (see Section 10.1 [RegPos], page 97).

La variable `mark-ring-max` spécifie le nombre maximal d'entrées à garder dans la pile des marques. Si le nombre maximal d'entrées est atteint et qu'une nouvelle est empilée, la dernière de la liste est abandonnée. Répéter `C-u C-`⟨SPC⟩ fait un cycle parmi les positions actuellement dans la pile.

La variable `mark-ring` contient la pile des marques elle-même, en tant que liste d'objets marque, la marque la plus récente en premier. Cette variable est locale à chaque tampon.

8.6 La Pile des Marques Globale

En plus de la pile des marques ordinaire, spécifique à chaque tampon, Emacs a une *pile des marques globale* unique. Elle enregistre une séquence de tampons dans lesquels vous avez placé la marque, pour que vous puissiez revenir à ces tampons.

Placer la marque crée toujours une entrée dans la pile des marques du tampon courant. Si vous avez changé de tampon depuis la précédente pose d'une marque, la nouvelle position de la marque crée de plus une entrée dans la pile des marques globale. Le résultat est que la pile des marques globale enregistre une séquence de tampons dans lesquels vous avez été, et, pour chaque tampon, un endroit où vous avez placé la marque.

La commande `C-x C-⟨SPC⟩` (`pop-global-mark`) saute au tampon et à la position de la dernière entrée dans la pile des marques globale. Elle boucle aussi dans la pile, ainsi des utilisations successives de `C-x C-⟨SPC⟩` vous déplacent dans des tampons de plus en plus anciens.

9 Couper et Déplacer du Texte

Couper veut dire effacer du texte et le copier dans le *presse-papiers*, à partir duquel on peut le récupérer en le *collant*.

La manière habituelle de déplacer ou de recopier du texte sous Emacs est de couper ce texte puis de le coller ailleurs, en un ou plusieurs endroits. Ceci est très sûr car Emacs sauvegarde plusieurs coupes récentes, pas seulement la dernière. C'est souple, car les mêmes commandes permettant d'effacer des unités syntaxiques permettent aussi de déplacer ces unités. Mais il y a d'autres moyens de copier du texte pour des usages spécifiques.

Emacs a un seul presse-papiers pour tous les tampons, ce qui permet de couper du texte dans un tampon et de le coller dans un autre.

9.1 Supprimer et Couper

La plupart des commandes qui effacent du texte le sauvegardent dans le presse-papiers pour que vous puissiez le déplacer ou le copier à d'autres endroits du tampon. Ces commandes sont connues comme des commandes de *coupe*. Les autres commandes effaçant du texte n'enregistrent pas ce texte dans le presse-papiers ; on les appelle des commandes de *suppression*. (Cette distinction est faite seulement pour l'effacement de texte dans le tampon.) Si vous faites une commande de coupe ou de suppression par erreur, vous pouvez utiliser la commande `C-x u` (`undo`) pour l'annuler. (see Section 4.4 [Undo], page 45).

Vous ne pouvez pas couper du texte en lecture seule, car ce type de texte ne permet aucune sorte de modification. Mais certains utilisateurs préfèrent utiliser les commandes de coupe pour copier du texte en lecture seule dans le presse-papier, sans pour autant le modifier. Si vous définissez la variable `kill-read-only-ok` à une valeur non `nil`, les commandes de coupe fonctionneront différemment dans un tampon en lecture seule : elles vous déplacent à travers le texte, et le copient dans le presse-papier, sans pour autant le supprimer du tampon. Lorsque ceci arrive, un message dans la zone d'écho vous en tient informé.

Les commandes de suppression incluent `C-d` (`delete-char`) et ⟨DEL⟩ (`delete-backward-char`), qui suppriment seulement un caractère à la fois, et les commandes qui effacent seulement des espaces ou des caractères newline. Les commandes qui peuvent détruire une quantité significative de données non triviales sont généralement des commandes de coupe. Les commandes de coupe et de suppression se reconnaissent par les mots '`kill`' (pour copier) et '`delete`' (pour supprimer) dans leur nom et leur description.

Un grand nombre de systèmes de fenêtrage suivent la convention que l'insertion supprime le texte déjà sélectionné s'il y en a. Vous pouvez demander à Emacs de fonctionner de cette manière en activant le mode Supprime

Sélection, avec `M-x delete-selection-mode`, ou en utilisant la personnalisation. Un autre effet de ce mode est que ⟨DEL⟩, `C-d` et quelques autres touches, lorsqu'une sélection existe, vont effacer la sélection en entier. Il active aussi le mode de Marque Transitoire. (see Section 8.2 [Transient Mark], page 78).

9.1.1 Suppression

`C-d`
⟨Delete⟩ Supprime le caractère suivant (`delete-char`). Si votre clavier a une touche de fonction ⟨Delete⟩ (placée habituellement sur le pavé d'édition), Emacs la lie aussi à `delete-char`.

⟨DEL⟩
⟨BS⟩ Supprime le caractère précédent (`delete-backward-char`). Certains claviers appelent cette touche "touche backspace" et l'étiquettent avec une flèche vers la gauche : ⟨<-⟩.

`M-\` Supprime les espaces et tabulations autour du point (`delete-horizontal-space`).

`M-`⟨SPC⟩ Supprime les espaces et tabulations autour du point, laissant une espace (`just-one-space`).

`C-x C-o` Supprime les lignes vierges autour de la ligne courante (`delete-blank-lines`).

`M-^` Joint deux lignes en supprimant le caractère fin-de-ligne intermédiaire, ainsi que les caractères d'indentation qui le suivent (`delete-indentation`).

Les commandes de suppression les plus courantes sont `C-d` (`delete-char`) et ⟨DEL⟩ (`delete-backward-char`). `C-d` supprime le caractère après le point, celui sur lequel se trouve le curseur. Elles ne déplacent pas le point. ⟨DEL⟩ supprime le caractère avant le curseur, et recule le point. Vous pouvez effacer des fins-de-lignes comme tout autre caractère du tampon ; supprimer une fin-de-ligne joint deux lignes. `C-d` et ⟨DEL⟩ ne sont pas toujours des commandes de suppression ; lorsqu'on leur donne un argument, elles deviennent des commandes de coupe, car elles peuvent effacer plus d'un caractère.

Tous les claviers ont une touche large, étiquetée ⟨DEL⟩, ⟨BACKSPACE⟩, ⟨BS⟩ ou ⟨DELETE⟩, qui est un peu au dessus de la touche ⟨RET⟩ ou ⟨ENTER⟩ est qui est normalement utilisée pour effacer ce que vous venez de taper. Indépendamment du nom affiché sur la touche, elle est pour Emacs équivalente à ⟨DEL⟩ — ou devrait l'être.

Un grand nombre de claviers ont une touche ⟨BACKSPACE⟩ un peu au dessus de ⟨RET⟩ ou ⟨ENTER⟩, et une touche ⟨DELETE⟩ ailleurs. Dans ce cas, la

touche ⟨BACKSPACE⟩ est ⟨DEL⟩, et la touche ⟨DELETE⟩ est équivalente à `C-d` — ou devrait l'être.

Pourquoi précisons-nous "ou devrait l'être " ? Lorsqu'Emacs démarre en utilisant un système de fenêtrage, il détermine automatiquement quelle(s) touche(s) doit être équivalente à ⟨DEL⟩. Ainsi les touches ⟨BACKSPACE⟩ et/ou ⟨DELETE⟩ font normalement ce qu'il faut. Dans certains cas inhabituels, Emacs obtient de mauvaises informations de la part du système. Si ces touches ne font pas ce qu'elles devraient, vous devez dire à Emacs quelle touche utiliser pour ⟨DEL⟩. See Section 32.2.1 [DEL Gets Help], page 493.

Sur des terminaux texte seulement, Emacs ne peut pas dire où se trouve quelle touche, et suit donc un plan uniforme qui peut ou non convenir à votre clavier. Le plan uniforme est que le caractère ASCII ⟨DEL⟩ supprime, et que le caractère ASCII ⟨BS⟩ (backspace) demande de l'aide (identique à `C-h`). Si ce n'est pas le cas pour votre clavier, et que vous pensez que la touche qui devrait supprimer en arrière demande plutôt de l'aide, voyez Section 32.2.1 [DEL Gets Help], page 493.

Les autres commandes de suppression sont celles qui suppriment seulement des caractères vierges : espaces, tabulations et fins-de-ligne. `M-\` (`delete-horizontal-space`) supprime toutes les espaces et tabulations avant et après le point. `M-`⟨SPC⟩ (`just-one-space`) fait de même mais laisse un espace après le point, sans tenir compte du nombre d'espaces qui existaient précédemment (même zéro).

`C-x C-o` (`delete-blank-lines`) supprime toutes les lignes vierges suivant la ligne courante. Si la ligne courante est vierge, elle supprime toutes les lignes vierges précédentes également (en laissant une ligne vierge, la ligne courante).

`M-^` (`delete-indentation`) joint la ligne courante et la ligne précédente, en supprimant un fin-de-ligne et les espaces l'entourant, laissant habituellement un simple espace. See Chapter 20 [Indentation], page 239.

9.1.2 Couper par Lignes

`C-k` Coupe le reste de la ligne ou une ou plusieurs lignes (`kill-line`).

La commande de coupe la plus simple est `C-k`. En début de ligne, elle coupe tout le texte de la ligne, laissant celle-ci vierge. Sur une ligne vierge, elle coupe la ligne entière, caractère fin-de-ligne compris. Pour couper une ligne non vierge en entier, allez au début de cette ligne et tapez `C-k` deux fois.

Plus généralement, `C-k` coupe depuis le point jusqu'à la fin de la ligne, à moins que le point ne soit en fin de ligne. Dans ce cas elle coupe la fin-de-ligne suivant le point, raccrochant ainsi la ligne suivante à la ligne courante. Les espaces et tabulations que vous ne pouvez pas voir en fin de ligne sont

ignorées pour déterminer ce comportement, donc si le point semble être à la fin de la ligne, vous pouvez être sûr que `C-k` coupera la fin-de-ligne.

Lorsqu'on passe à `C-k` un argument positif, elle coupe autant de lignes et de fins-de-lignes (cependant, le texte de la ligne courante précédant le point est conservé). Avec un argument négatif $-n$, elle coupe n lignes précédant la ligne courante (ainsi que le texte de la ligne courante précédant le point). Ainsi, `C-u - 2 C-k` en début de ligne coupe les deux lignes précédentes.

`C-k` avec un argument égal à zéro coupe le texte de la ligne courante avant le point.

Si la variable `kill-whole-line` est non-`nil`, `C-k` en tout début d'une ligne coupe la ligne entière avec la fin-de-ligne. Cette varaible est `nil` par défaut.

9.1.3 Autres Commandes de Coupe

`C-w` Coupe la région (depuis le point jusqu'à la marque) (`kill-region`).

`M-d` Coupe un mot (`kill-word`). See Section 21.1 [Words], page 243.

`M-`⟨DEL⟩ Coupe le mot précédent (`backward-kill-word`).

`C-x` ⟨DEL⟩ Coupe depuis le début de la phrase (`backward-kill-sentence`). See Section 21.2 [Sentences], page 245.

`M-k` Coupe jusqu'à la fin de la phrase (`kill-sentence`).

`C-M-k` Coupe une sexp (`kill-sexp`). See Section 22.2 [Lists], page 274.

`M-z` *caractère*
Coupe jusqu'à la prochaine occurence de *caractère* (`zap-to-char`).

`C-w` (`kill-region`) est une commande de coupe très générale, qui coupe tout entre le point et la marque. Avec cette commande, vous pouvez couper une suite quelconque de caractères consécutifs, si vous placez d'abord la région autour de ces caractères.

Une manière pratique de couper est combinée avec la recherche : `M-z` (`zap-to-char`) lit un caractère et coupe depuis le point et jusqu'à la prochaine occurrence (incluse) de ce caractère dans le tampon. Un argument numérique joue le rôle de compteur de répétition. Un argument négatif indique de rechercher en arrière et de couper le texte avant le point.

D'autres unités syntaxiques peuvent être coupées : des mots, avec `M-`⟨DEL⟩ et `M-d` (see Section 21.1 [Words], page 243) ; des sexps, avec `C-M-k` (see Section 22.2 [Lists], page 274) ; et des phrases, avec `C-x` ⟨DEL⟩ et `M-k` (see Section 21.2 [Sentences], page 245).

Vous pouvez utiliser des commandes de coupe dans des tampons en lecture seule. Elles ne modifient pas le tampon, et sonnent pour vous en avertir,

mais copient le texte que vous avez essayé de couper dans le presse-papiers, pour que vous puissiez le coller dans d'autres tampons. La plupart des commandes de coupe déplacent le point à travers le texte qu'elles ont coupé de cette façon, pour que les commandes de coupe successives construisent une unique entrée dans le presse-papiers, comme d'habitude.

9.2 Coller

Coller veut dire réinsérer du texte précédemment coupé. La manière usuelle de déplacer ou de copier du texte est de le couper puis de le coller autre part, une ou plusieurs fois.

`C-y` Colle le dernier texte coupé (`yank`).

`M-y` Remplace le texte venant d'être collé avec un texte coupé précédemment (`yank-pop`).

`M-w` Enregistre la région comme dernier texte coupé, sans l'effacer (`kill-ring-save`).

`C-M-w` Ajoute la prochaine coupe à la dernière entrée du presse-papiers (`append-next-kill`).

9.2.1 Le Presse-Papiers

Tout le texte coupé est sauvegardé dans le *presse-papiers*, une liste de blocs de texte qui ont été coupés. Il y a un seul presse-papiers, partagé par tous les tampons, de sorte que vous puissiez couper du texte dans un tampon et le coller dans un autre tampon. C'est la manière habituelle de déplacer du texte d'un fichier à un autre. (See Section 9.3 [Accumulating Text], page 92, pour d'autres moyens.)

La commande `C-y` (`yank`) réinsère le texte coupé en dernier. Il place le curseur à la fin du texte, et place la marque au début du texte. See Chapter 8 [Mark], page 77.

`C-u C-y` laisse le curseur au début du texte, et place la marque à la fin. C'est ce qui arrive quand l'argument est spécifié juste par `C-u`. Tout autre argument, comme `C-u` suivi de chiffres, demande de coller une coupe antérieure. (see Section 9.2.3 [Earlier Kills], page 91).

Pour copier un bloc de texte, vous pouvez utiliser `M-w` (`kill-ring-save`), qui copie la région dans le presse-papiers, sans la retirer du tampon. C'est approximativement équivalent à `C-w` suivi de `C-x u`, exception faite que `M-w` n'altère pas l'historique des annulations et ne change pas temporairement l'écran.

9.2.2 Ajouter des Coupes

Normalement, chaque commande de coupe empile une nouvelle entrée dans le presse-papiers. Cependant, deux ou plusieurs commandes de coupe à la suite combinent leurs textes dans une seule entrée, pour qu'un seul `C-y` colle tout le texte d'un bloc, comme il l'était avant d'être coupé.

Dès lors, si vous désirez coller du texte d'un bloc, vous n'avez pas besoin de le couper entièrement en une seule commande ; vous pouvez le couper ligne par ligne, ou mot par mot, jusqu'à ce que vous l'ayez coupé entièrement, et vous pouvez alors le restituer d'un coup.

Les commandes qui coupent en avant depuis le point ajoutent le texte à la fin du texte précédemment coupé. Les commandes qui coupent en arrière à partir du point ajoutent le texte au début. De cette manière, une séquence mixte de commandes de coupe en avant ou en arrière placent tout le texte coupé dans une entrée sans tout mélanger. Des arguments numériques ne brisent pas la suite d'ajout des coupes. Par exemple, supposez que le tampon contienne ce texte :

```
Voici une ligne ⋆d'exemple de texte.
```

avec le point repésenté par ⋆. Si vous tapez `M-d` `M-`⟨DEL⟩ `M-d` `M-`⟨DEL⟩, coupant alternativement en avant et en arrière, vous finissez avec '`une ligne d'exemple de`' dans une entrée du presse-papiers, et '`Voici  texte.`' dans le tampon. (Notez le double espace, que vous pouvez effacer avec `M-`⟨SPC⟩ ou `M-q`.)

Un autre moyen de couper le même texte est de vous déplacer en arrière de deux mots avec `M-b M-b`, puis de couper les quatre mots d'un coup avec `C-u M-d`. Ceci produit exactement le même résultat dans le tampon et dans le presse-papiers. `M-f M-f C-u M-`⟨DEL⟩ coupe le même texte, mais cette fois en allant toujours en arrière ; encore une fois, le résultat est le même. Le texte dans le presse-papiers a toujours le même ordre qu'il avait dans le tampon avant que vous ne le coupiez.

Si une commande de coupe est séparée de la dernière commande de coupe par d'autres commandes (non juste des arguments numériques), elle démarre une nouvelle entrée dans le presse-papiers. Mais vous pouvez la forcer à ajouter en tapant d'abord la commande `C-M-w` (`append-next-kill`) juste avant. `C-M-w` indique à la prochaine commande, si c'est une commande de coupe, d'ajouter le texte qu'il coupe au dernier texte coupé, plutôt que de commencer une nouvelle entrée. Avec `C-M-w`, vous pouvez couper plusieurs morceaux séparés de texte et les accumuler pour les coller en un seul endroit.

Une commande de coupe après `M-w` n'ajoute pas au texte que `M-w` a copié dans le presse-papiers.

9.2.3 Coller des Coupes Antérieures

Pour retrouver du texte coupé qui n'est pas le plus récemment coupé, utilisez la commande `M-y` (`yank-pop`). Il prend le texte qui vient d'être collé et le remplace par celui d'une coupe antérieure. Ainsi, pour retrouver le texte de l'avant-dernière coupe, utilisez d'abord `C-y` pour coller la dernière coupe, puis utilisez `M-y` pour le remplacer par la coupe précédente. `M-y` n'est autorisé qu'après un `C-y` ou un autre `M-y`.

Vous pouvez voir `M-y` comme un pointeur sur la "dernière colle" qui pointe sur une entrée du presse-papiers. Chaque fois que vous coupez, le pointeur de "dernière colle" se déplace sur l'entrée nouvellement créée en haut de la pile. `C-y` colle l'entrée indiquée par le pointeur de "dernière colle". `M-y` déplace le pointeur de "dernière colle" sur une entrée différente, et le texte dans le tampon change en conséquence. Suffisamment de commandes `M-y` peuvent déplacer le pointeur sur n'importe quelle entrée de la pile. Au bout d'un moment, le pointeur peut atteindre la fin de la pile ; la commande `M-y` ramène alors celui-ci sur la première entrée.

`M-y` déplace le pointeur de "dernière colle" dans la pile, mais il ne change pas l'ordre des entrées dans celle-ci, qui va toujours de la coupe la plus récente vers la plus ancienne.

`M-y` peut prendre un argument numérique, qui lui indique de combien d'entrées avancer le pointeur de "dernière colle". Un argument négatif déplace le pointeur vers le haut de la pile ; à partir du haut de la pile, il est déplacé à la fin de la pile.

Une fois que le texte recherché est inséré dans le tampon, vous pouvez arrêter les commandes `M-y` et il restera en place. C'est juste une copie de l'entrée du presse-papiers, vous pouvez donc l'éditer dans le tampon sans modifier le contenu du presse-papiers. Aussi longtemps qu'une nouvelle coupe n'est pas faite, le pointeur de "dernière colle" reste à la même place, et la répétition de `C-y` collera une autre copie de cette même coupe.

Si vous connaissez le nombre de commandes `M-y` nécessaires pour retrouver le texte que vous désirez, vous pouvez coller ce texte en une seule étape en utilisant `C-y` avec un argument numérique. `C-y` avec un argument remonte la pile en arrière du nombre d'entrées spécifié et retrouve alors le texte dans le presse-papiers. Ainsi, `C-u 2 C-y` colle l'avant-dernier bloc de texte coupé. C'est équivalent à `C-y M-y`. `C-y` avec un argument numérique se déplace à partir du pointeur de "dernière colle", et place ce pointeur sur l'entrée qu'il colle.

La profondeur du presse-papiers est contrôlée par la variable `kill-ring-max` ; c'est le nombre maximal de blocs sauvegardés dans le presse-papiers.

Le contenu actuel du presse-papiers est stocké dans une variable appelée `kill-ring` ; vous pouvez examiner le contenu du presse-papiers avec la commande `C-h v kill-ring`.

9.3 Accumuler du Texte

Généralement, nous copions ou déplaçons du texte en le coupant puis en le collant, mais il existe d'autres méthodes commodes pour copier un bloc de texte en plusieurs endroits, ou pour copier plusieurs blocs de texte éparpillés en un seul endroit. Pour copier un bloc vers plusieurs endroits, stockez-le dans un registre (see Chapter 10 [Registers], page 97). Nous décrivons ici les commandes permettant d'accumuler des morceaux de texte éparpillés dans un tampon ou dans un fichier.

`M-x append-to-buffer`
: Ajoute la région derrière le point dans le tampon spécifié.

`M-x prepend-to-buffer`
: Ajoute la région devant le point dans le tampon spécifié.

`M-x copy-to-buffer`
: Copie la région dans le tampon spécifié, supprimant l'ancien contenu du tampon.

`M-x insert-buffer`
: Insère le contenu du tampon spécifié dans le tampon courant, au point.

`M-x append-to-file`
: Ajoute la région à la fin du fichier spécifié.

Pour accumuler du texte dans un tampon, utilisez la commande `M-x append-to-buffer`. Cette commande lit un nom de tampon, puis insère une copie de la région dans le tampon spécifié. Si vous spécifiez un tampon non existant, `append-to-buffer` le crée pour vous. Le texte est inséré à l'endroit du point dans ce tampon. Si vous êtes en train d'éditer ce tampon, le texte copié se retrouve au milieu du texte du tampon, à l'endroit où se trouvait le point.

Le point dans ce tampon est déplacé à la fin du texte copié, des utilisations successives de `append-to-buffer` accumulent donc le texte dans le tampon spécifié, dans l'ordre dans lequel il a été copié. Pour être précis, `append-to-buffer` n'ajoute pas toujours à la fin du tampon — il n'ajoute à la fin que si le point dans ce tampon est à la fin. Cependant, si `append-to-buffer` est la seule commande utilisée pour modifier un tampon, le point est toujours à la fin.

`M-x prepend-to-buffer` fonctionne comme `append-to-buffer` à l'exception que le point dans le tampon de destination est laissé avant le texte copié, de manière que des ajouts successifs ajoutent le texte dans l'ordre inverse. `M-x copy-to-buffer` est semblable à part que le tampon de destination est effacé, pour que ce tampon ne contienne que le texte nouvellement copié.

Pour récupérer le texte accumulé d'un autre tampon, utilisez `M-x insert-buffer` ; qui prend aussi un *nom de tampon* comme argument.

Elle insére une copie du texte du tampon *nom de tampon* dans le tampon sélectionné. Vous pouvez alternativement sélectionner l'autre tampon pour l'éditer, puis éventuellement y déplacer du texte en coupant. See Chapter 15 [Buffers], page 185, pour des informations supplémentaires sur les tampons.

Plutôt que d'accumuler du texte dans un tampon d'Emacs, vous pouvez ajouter du texte directement dans un fichier avec `M-x append-to-file`, qui prend *nom de fichier* comme argument. Elle ajoute le texte de la région à la fin du fichier spécifié. Ce fichier est immédiatement modifié sur le disque.

Vous devez utiliser `append-to-file` uniquement avec des fichiers qui ne sont *pas* visités par Emacs. L'utiliser avec un fichier que vous éditez avec Emacs pourrait modifier le fichier à l'insu d'Emacs, ce qui pourrait conduire à perdre certaines de vos éditions.

9.4 Rectangles

Les commandes de rectangle opèrent sur des régions rectangulaires de texte : tous les caractères compris entre une paire de colonnes donnée, dans un intervalle de lignes donné. Des commandes sont fournies pour couper des rectangles, coller des rectangles coupés, les effacer, les remplir avec des espaces ou du texte, ou les supprimer. Les commandes de rectangle sont utiles pour des textes au format multi-colonnes, et pour convertir des textes vers ou à partir de ce format.

Pour spécifier un rectangle sur lequel une telle commande doit travailler, vous placez la marque à un coin et le point au coin opposé. Le rectangle ainsi spécifié est appelé la *région rectangle* car vous contrôlez celui-ci de la même manière que vous contrôlez la région. Mais n'oubliez pas qu'une combinaison donnée d'un point et d'une marque peut être interprétée soit comme une région, soit comme un rectangle, selon la commande qui l'utilise.

Si le point et la marque sont sur la même colonne, le rectangle qu'ils délimitent est vide. S'ils sont sur la même ligne, le rectangle fait une ligne de haut. La différence entre lignes et colonnes vient du fait que le point (et la marque) sont entre deux colonnes, mais sur une ligne.

`C-x r k` Coupe le texte de la région rectangle, sauvegardant son contenu comme "dernier rectangle coupé" (`kill-rectangle`).

`C-x r d` Supprime le texte de la région rectangle (`delete-rectangle`).

`C-x r y` Colle le dernier rectangle coupé avec son coin en haut à gauche à l'endroit du point (`yank-rectangle`).

`C-x r o` Insère des espaces pour remplir l'espace occupé par la région rectangle (`open-rectangle`). Ce que contenait auparavant la région rectangle se trouve décalé vers la droite.

`M-x clear-rectangle`
: Efface la région rectangle en remplaçant son contenu par des espaces.

`M-x delete-whitespace-rectangle`
: Efface les espaces dans chacune des lignes du rectangle spécifié, en commençant par la colonne de gauche du rectangle.

`C-x r t` *chaîne* ⟨RET⟩
: Insère *chaîne* dans chaque ligne de la région rectangle. (`string-rectangle`).

`M-x replace-rectangle` ⟨RET⟩ *chaîne* ⟨RET⟩
: Remplace chaque ligne de la région rectangle par *chaîne* (`string-rectangle`).

Les opérations sur les rectangles se divisent en deux classes : les commandes supprimant et insérant des rectangles, et les commandes rendant des rectangles vides.

Il y a deux manières de se débarrasser du texte d'un rectangle : vous pouvez supprimer le texte ou le sauvegarder comme "dernier rectangle coupé". Les commandes correspondantes sont `C-x r d` (`delete-rectangle`) et `C-x r k` (`kill-rectangle`). Dans chacun des cas, la portion de chaque ligne contenue dans les frontières du rectangle est supprimée, ce qui décale le texte qui suit sur la ligne (s'il y en a) vers la gauche.

Notez que "couper" un rectangle n'est pas couper au sens habituel ; le rectangle n'est pas stocké dans le presse-papiers, mais dans un endroit spécial qui ne retient que le dernier rectangle coupé. Coller un rectangle est très différent de coller du texte linéaire, au point que l'on doit utiliser des commandes de colle différentes.

Pour coller le dernier rectangle coupé, tapez `C-x r y` (`yank-rectangle`). Coller un rectangle est l'inverse de le couper. Le point spécifie l'endroit où placer le coin en haut à gauche du rectangle. La première ligne du rectangle est insérée au point, la seconde ligne du rectangle est insérée une ligne plus bas, et ainsi de suite. Le nombre de lignes affectées est déterminé par la hauteur du rectangle sauvegardé.

Vous pouvez convertir une liste sur une colonne en une liste sur deux colonnes en coupant et collant des rectangles ; coupez la seconde moitié de la liste comme rectangle puis collez-la à côté de la première ligne de la liste. See Section 30.10 [Two-Column], page 444, pour un autre moyen d'éditer du texte multi-colonnes.

Vous pouvez aussi copier des rectangles dans et à partir de registres avec `C-x r r` *r* et `C-x r i` *r*. See Section 10.3 [Rectangle Registers], page 98.

Vous pouvez utiliser deux commandes pour remplir des rectangles avec des espaces : `M-x clear-rectangle` qui efface le texte existant, et `C-x r o` (`open-rectangle`) qui insère un rectangle d'espaces. Effacer un rectangle

équivaut à le supprimer puis à insérer un rectangle d'espaces de la même taille.

La commande `M-x delete-whitespace-rectangle` supprime les espaces à partir d'une colonne donnée. Cela s'applique à chacune des lignes du rectangle, et la colonne est spécifiée par le côté gauche du rectangle. Le côté droit du rectangle ne change rien au comportement de cette commande.

La commande `C-x r t` (`M-x string-rectangle`) insère une chaîne dans chaque ligne de la région rectangle, avant celui-ci, en décalant le texte vers la droite.

La commande `M-x replace-rectangle` est semblable à `C-x r t`, mais remplace le rectangle d'origine. La longueur de la chaîne ne doit pas forcément être la même que la largeur du rectangle. Si la longueur de la chaîne est inférieure, le texte à droite du rectangle est décalé vers la gauche ; si la chaîne est plus large que le rectangle, ce texte est décalé vers la droite.

10 Registres

Les *registres* d'Emacs sont des emplacements où vous pouvez sauvegarder du texte ou des positions pour un usage ultérieur. Une fois du texte ou un rectangle sauvegardé dans un registre, vous pouvez le copier dans le tampon une ou plusieurs fois ; vous pouvez déplacer le point à une position sauvegardée dans un registre une ou plusieurs fois.

Chaque registre a un nom qui est constitué d'un seul caractère. Un registre peut contenir un morceau de texte, un rectangle, une position, une configuration de fenêtres, ou un nom de fichier, mais seulement une chose à la fois. Tout ce que vous sauvegardez dans un registre y reste tant que vous n'y sauvegardez pas autre chose. Pour voir ce que contient un registre *r*, faites `M-x view-register`.

`M-x view-register` ⟨RET⟩ *r*
: Affiche une description du contenu du registre *r*.

10.1 Sauvegarder des Positions dans des Registres

La sauvegarde d'une position enregistre un emplacement dans un tampon pour que vous puissiez y retourner plus tard. Se déplacer à une position sauvegardée sélectionne le tampon et déplace le point à l'emplacement.

`C-x r` ⟨SPC⟩ *r*
: Sauvegarde la position du point dans le registre *r* (`point-to-register`).

`C-x r j` *r*
: Saute à la position sauvegardée dans le registre *r* (`jump-to-register`).

Pour sauvegarder la position courante du point dans un registre, choisissez un nom *r* et tapez `C-x r` ⟨SPC⟩ *r*. Le registre *r* retient la position ainsi sauvegardée jusqu'à ce que vous mettiez quelque chose d'autre dans ce registre.

La commande `C-x r j` *r* déplace le point à la position sauvegardée dans le registre *r*. Le registre n'est pas affecté ; il continue de retenir la même position. Vous pouvez sauter à la position sauvegardée autant de fois que vous le voulez.

Si vous utilisez `C-x r j` pour aller à une position sauvegardée, mais que le tampon correspondant a été fermé, `C-x r j` essaie de recréer le tampon en visitant le même fichier. Bien sûr, cela marche seulement pour des tampons qui visitaient des fichiers.

10.2 Sauvegarder du Texte dans des Registres

Lorsque vous voulez insérer une copie d'un même morceau de texte plusieurs fois, il peut être malaisé de le coller depuis le presse-papiers, car chaque nouvelle coupe déplace cette entrée un peu plus profondément dans la pile. Une alternative est de stocker le texte dans un registre et de le récupérer plus tard.

`C-x r s` *r* Copie la région dans le registre *r* (`copy-to-register`).

`C-x r i` *r* Insère le texte depuis le registre *r* (`insert-register`).

`C-x r s` *r* stocke une copie du texte de la région dans le registre appelé *r*. Avec un argument, `C-x r s` *r* supprime de plus le texte du tampon.

`C-x r i` *r* insère dans le tampon le texte contenu dans le registre *r*. Il laisse normalement le point avant le texte et place la marque après, mais avec un argument (`C-u`) il place le point après le texte et la marque avant.

10.3 Sauvegarder des Rectangles dans des Registres

Un registre peut contenir un rectangle au lieu d'un texte linéaire. Le rectangle est représenté comme une liste de chaînes. See Section 9.4 [Rectangles], page 93, pour des informations basiques sur la manière de spécifier un rectangle dans un tampon.

`C-x r r` *r* Copie la région rectangle dans le registre *r* (`copy-rectangle-to-register`). Avec un argument numérique, le supprime du tampon.

`C-x r i` *r* Insère le rectangle stocké dans le registre *r* (s'il contient un rectangle) (`insert-register`).

La commande `C-x r i` *r* insère du texte linéaire si c'est ce que contient le registre, et un rectangle si le registre en contient un.

Voyez aussi la commande `sort-columns`, que vous pouvez voir comme triant un rectangle. See Section 30.8 [Sorting], page 441.

10.4 Sauvegarder une Configuration de Fenêtres dans des Registres

Vous pouvez sauvegarder la configuration des fenêtres du cadre sélectionné dans un registre, ou encore de toutes les fenêtres dans tous les cadres, et retrouver cette configuration plus tard.

`C-x r w` *r* Sauvegarde l'état des fenêtres du cadre sélectionné dans le registre *r* (`window-configuration-to-register`).

`C-x r f` *r* Sauvegarde l'état de tous les cadres, incluant toutes leurs fenêtres, dans le registre *r* (`frame-configuration-to-register`).

Utilisez `C-x r j` *r* pour retrouver la configuration d'une fenêtre ou d'un cadre. C'est la même commande que pour retrouver une position du curseur. Lorsque vous récupérez la configuration d'un cadre, tout cadre existant non inclus dans la configuration devient invisible. Si vous préférez supprimer ces cadres, utilisez `C-u C-x r j` *r*.

10.5 Stocker des Nombres dans des Registres

Il existe des commandes pour stocker un nombre dans un registre, pour insérer le nombre dans un tampon en décimal, et pour l'incrémenter. Ces commandes peuvent être utiles pour des macros clavier. (see Section 31.3 [Keyboard Macros], page 470).

`C-u` *nombre* `C-x r n` *reg*
Stocke le *nombre* dans le registre *reg* (`number-to-register`).

`C-u` *nombre* `C-x r +` *reg*
Incrémente le nombre contenu dans le registre *reg* de *nombre* (`increment-register`).

`C-x r g` *reg*
Insère le nombre contenu dans le registre *reg* dans le tampon.

`C-x r g` est la même commande utilisée pour insérer n'importe quelle autre sorte de contenu de registre dans le tampon.

10.6 Sauvegarder des Noms de Fichiers dans des Registres

Si vous visitez certains fichiers fréquemment, vous pouvez placer leurs noms dans des registres pour les visiter plus facilement. Voici le code Lisp utilisé pour placer un nom de fichier dans un registre :

```
(set-register ?r '(file . nom))
```

Par exemple,

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

place le nom de fichier indiqué dans le registre ‘z’.

Pour visiter le fichier dont le nom est dans le registre *r*, tapez `C-x r j` *r*. (C'est la même commande utilisée pour sauter à une position ou retrouver une configuration d'un cadre.)

10.7 Marque-Pages

Les *Marque-pages* ressemblent aux registres en ce qu'ils retiennent des positions auxquelles vous pouvez sauter plus tard. À l'inverse des registres, ils ont des noms longs, et ils sont automatiquement conservés d'une session d'Emacs à la suivante. L'emploi standard des marque-pages est d'enregistrer “où vous étiez en train de lire” dans des fichiers divers.

`C-x r m` ⟨RET⟩
: Plaec le marque-page pour le fichier visité, au point.

`C-x r m` *marque-page* ⟨RET⟩
: Place le marque-page appelé *marque-page* au point (`bookmark-set`).

`C-x r b` *bookmark* ⟨RET⟩
: Saute au marque-page appelé *bookmark* (`bookmark-jump`).

`C-x r l`
: Liste tous les marque-pages (`list-bookmarks`).

`M-x bookmark-save`
: Sauvegarde toutes les valeurs courantes des marque-pages dans le fichier de marque-pages par défaut.

L'utilisation standard des marque-pages est d'enregistrer une position courante dans plusieurs fichiers à la fois. Ainsi la commande `C-x r m`, qui place un marque-page, utilise par défaut le nom du fichier visité comme nom pour le marque-page. Si vous nommez chaque marque-page comme le fichier sur lequel il pointe, vous pouvez aisément revisiter un de ces fichiers avec `C-x r b`, et vous rendre en même temps à la position du marque-page.

Pour afficher une liste de tous vos marque-pages dans un tampon séparé, tapez `C-x r l` (`list-bookmarks`). Si vous sélectionnez ce tampon, vous pouvez l'utiliser pour éditer les définitions de vos marque-pages ou les annoter. Tapez `C-h m` dans ce tampon pour avoir plus d'informations sur les commandes d'édition spéciales.

Lorsque vous quittez Emacs, ce dernier vous propose de sauvegarder vos marque-pages dans votre fichier de marque-pages par défaut, ‘`~/.emacs.bmk`’, si vous avez changé des marque-pages. Vous pouvez

aussi sauvegarder les marque-pages à tout moment avec la commande `M-x bookmark-save`. Les commandes de marque-pages chargent automatiquement votre fichier de marque-pages par défaut. Cette sauvegarde et ce chargement permettent à vos marque-pages de persister d'une session d'Emacs à la suivante.

Si vous mettez la variable `bookmark-save-flag` à 1, alors chacune des commandes qui place un marque-page sauvegardera aussi vos marque-pages ; de cette manière, vous ne perdez aucun marque-page, même si Emacs se plante. (Une valeur numérique indique combien de modifications de marque-pages doivent avoir lieu avant de sauvegarder.)

Les valeurs de position de marque-pages sont sauvegardées avec le contexte environnant, pour que `bookmark-jump` puisse retrouver la bonne position même si le fichier a été légèrement modifié. La variable `bookmark-search-size` indique le nombre de caractères de contexte à enregistrer, de chaque côté de la position du marque-page.

Voici quelques commandes supplémentaires pour travailler avec les marque-pages :

`M-x bookmark-load` ⟨RET⟩ *nom de fichier* ⟨RET⟩
: Charge un fichier appelé *nom de fichier* qui contient une liste de valeurs de marque-pages. Vous pouvez utiliser cette commande, comme `bookmark-write`, pour travailler avec d'autres fichiers de marque-pages que votre fichier de marque-pages par défaut.

`M-x bookmark-write` ⟨RET⟩ *nom de fichier* ⟨RET⟩
: Sauvegarde toutes les valeurs de marque-pages courantes dans le fichier *nom de fichier*.

`M-x bookmark-delete` ⟨RET⟩ *marque-page* ⟨RET⟩
: Supprime le marque-page appelé *marque-page*.

`M-x bookmark-insert-location` ⟨RET⟩ *marque-page* ⟨RET⟩
: Insère dans le tampon le nom du fichier sur lequel pointe le marque-page *marque-page*.

`M-x bookmark-insert` ⟨RET⟩ *marque-page* ⟨RET⟩
: Insère dans le tampon le *contenu* du fichier sur lequel pointe le marque-page *marque-page*.

11 Contrôler l'Affichage

Lorsqu'une partie seulement d'un tampon tient dans une fenêtre, Emacs essaie d'en montrer une partie qui est susceptible d'être intéressante. Les commandes de contrôle de l'affichage vous permettent de spécifier quelle partie du texte vous voulez voir, et comment l'afficher.

11.1 Utiliser plusieurs Faces de Caractères

Lors de l'utilisation d'Emacs dans un système de fenêtrage, vous pouvez utiliser plusieurs styles pour afficher les caractères. Certains aspects du style que vous contrôlez sont la police du carctère, la couleur du caractère, la couleur du fond, et si le le caractère est souligné ou non, et avec quelle couleur.

Les fonctionnalités qui s'appuient sur du texte en plusieurs faces (comme le mode Verrouillage de Police) fonctionneront aussi sur dse terminaux non fenêtrés qui peuvent afficher plusieurs faces, soit par couleurs, soit en soulignant et en mettant en gras. Ceux-ci incluent la console de GNU/Linux, un `xterm` avec support des couleurs, l'affichage MS-DOS (see Appendix E [MS-DOS], page 539), et la version MS-Windows invoquée avec l'option '`-nw`'. Emacs détermine automatiquement si le terminal a cette capacité.

La manière de contrôler le style de l'affichage est de définir des *faces* et de les nommer. Chaque face peut définir divers attributs, comme la hauteur de la police de caractères, son poids et son inclinaison, les couleurs du caractère et du fond, et le soulignement, mais n'a pas à les définir tous. En spécifiant la face ou les faces à utiliser pour une partie donnée du texte dans le tampon, vous contrôlez comment ce texte apparaît.

Le style d'affichage utilisé pour un caractère donné dans le texte est déterminé en combinant plusieurs faces. Tout aspect du style d'affichage qui n'est pas spécifié par des extensions ou des propriétés de texte provient d'une face par défaut qui hérite ces attributs du cadre lui-même.

Le mode Enrichi, mode pour éditer du texte formaté, inclut plusieurs commandes et menus pour spécifier des faces. See Section 21.11.4 [Format Faces], page 267, pour voir comment spécifier la police pour le texte du tampon. See Section 21.11.5 [Format Colors], page 268, pour voir comment spécifier les couleurs du caractère et du fond.

Pour altérer l'apparence d'une face, utilisez le tampon de personnalisation. See Section 31.2.2.3 [Face Customization], page 463. Vous pouvez aussi utiliser des ressources X pour spécifier les attributs d'une face particulière. (see Section B.13 [Resources X], page 521).

Alternativement, vous pouvez changer les couleurs du caractère et du fond d'une face donnée avec `M-x set-face-foreground` et `M-x`

`set-face-background`. Ces commandes demandent dans le mini-tampon un nom de face et un nom de couleur, avec complétion, puis fait utiliser cette couleur à cette face.

Pour voir quelles faces sont actuellement définies, et à quoi elles ressemblent, tapez `M-x list-faces-display`. Il est possible pour une face donnée d'apparaître différemment dans différents cadres ; cette commande montre l'apparence dans le cadre depuis lequel vous lancez la commande. Voici une liste des faces définies en standard :

`default` Cette face est utilisée pour le texte ordinaire n'utilisant aucune autre face.

`mode-line`
Cette face est utilisée pour les lignes de mode. Par défaut, elle est dessinée avec des ombres pour un effet "surélevé" sur les systèmes à fenêtrage, et dessinée comme l'inverse de la face par défaut sur les terminaux non fenêtrés. See Section 11.12 [Display Custom], page 115.

`header-line`
Similaire à `mode-line` pour une ligne d'entête de fenêtre. La plupart des modes n'utilisent pas de ligne d'entête, mais le mode Info l'utilise.

`highlight`
Cette face est utilisée pour mettre en surbrillance des portions de texte, dans divers modes. Par exemple, le texte sensible à la souris est mis en surbrillance en utilisant cette face.

`isearch` Cette face est utilisée pour mettre en surbrillance les correspondances d'une recherche incrémentale (Isearch).

`isearch-lazy-highlight-face`
Cette face est utilisée pour mettre en surbrillance légère les correspondances d'une recherche incrémentale autres que celle en cours.

`region` Cette face est utilisée pour afficher une région sélectionnée (lorsque le mode Marque Transitoire est actif—voir plus loin).

`secondary-selection`
Cette face est utilisée pour afficher une sélection X secondaire (see Section 17.2 [Secondary Selection], page 206).

`bold` Cette face utilise une variante grasse de la police par défaut, s'il en existe une.

`italic` Cette face utilise une variante italique de la police par défaut, s'il en existe une.

`bold-italic`
Cette face utilise une variante grasse italique de la police par défaut, s'il en existe une.

`underline`
: Cette face souligne le texte.

`fixed-pitch`
: La face à point fixe de base.

`fringe`
: La face pour les bordures gauche et droite des fenêtres sur des affichages graphiques. (Les bordures sont les portions étroites du cadre Emacs entre la zone de texte et les bords du cadre.

`scroll-bar`
: Cette face détermine l'apparence visuelle de la barre de défilement.

`border`
: Cette face détermine la couleur du bord du cadre.

`cursor`
: Cette face détermine la couleur du curseur.

`mouse`
: Cette face détermine la couleur du pointeur de la souris.

`tool-bar`
: C'est la face de base pour la barre d'outils. Aucun texte n'apparaît dans la barre d'outils, mais les couleurs de la face affectent l'apparence des icônes de la barre d'outils.

`tooltip`
: Cette face est utilisée pour les tooltips.

`menu`
: Cette face détermine les couleurs et la police des menus d'Emacs. Définir les polices des menus LessTif/Motif est pour l'instant non supporté ; les tentatives de définir la police sont ignorées dans ce cas.

`trailing-whitespace`
: La face pour mettre en surbrillance les espaces à la traîne lorsque `show-trailing-whitespace` est non nil.

`variable-pitch`
: La face à point variable de base.

Lorsque le mode de Marque Transitoire est actif, le texte de la région est mis en surbrillance lorsque la marque est active. Pour cela, la face `region` est utilisée ; vous pouvez contrôler le style de la surbrillance en changeant le style de cette face (see Section 31.2.2.3 [Face Customization], page 463). See Section 8.2 [Transient Mark], page 78, pour plus d'informations sur le mode de Marque Transitoire et l'activation et désactivation de la marque.

Un moyen facile d'utiliser les faces est d'activer le mode Verrouillage de Police. Ce mode mineur, qui est toujours local à un tampon particulier, s'arrange pour choisir les faces selon la syntaxe du texte que vous éditez. Il peut reconnaître les commentaires et les chaînes dans la plupart des langages ; dans plusieurs langages, il peut aussi reconnaître et mettre correctement en surbrillance plusieurs autres constructions importantes. See Section 11.2 [Font Lock], page 106, pour plus d'informations sur le mode Verrouillage de Police et la mise en surbrillance syntaxique.

Vous pouvez imprimer le tampon avec la surbrillance qui apparaît sur l'écran en utilisant la commande `ps-print-buffer-with-faces`. See Section 30.6 [PostScript], page 439.

11.2 Mode Verrouillage de Police

Le mode Verrouillage de Police est un mode mineur, toujours local à un tampon particulier, qui met en surbrillance (ou "fontifie") en utilisant diverses faces selon la syntaxe de texte que vous éditez. Il peut reconnaître les commentaires et les chaînes dans la plupart des langages, et peut aussi reconnaître et mettre correctement en surbrillance diverses autres constructions importantes—par exemple, les noms de fonctions étant définies ou les mots-clefs réservés.

La commande `M-x font-lock-mode` active ou désactive le mode Verrouillage de Police selon l'argument, et passe de l'un à l'autre lorsqu'aucun argument n'est fourni. La fonction `turn-on-font-lock` active inconditionnellement le mode Verrouillage de Police. Cette fonction est utile pour les foctions crochets de modes. Par exemple, pour activer le mode Verrouillage de Police lorsque vous éditez un fichier C, vous pouvez faire cela :

```
(add-hook 'c-mode-hook 'turn-on-font-lock)
```

Pour activer le mode Verrouillage de Police automatiquement dans tous les modes qui le permetent, personnalisez l'option utilisateur `global-font-lock-mode` ou utilisez la fonction `global-font-lock-mode` dans votre fichier '`.emacs`', comme ceci :

```
(global-font-lock-mode 1)
```

Le mode Verrouillage de Police utilise plusieurs faces spécifiques pour faire son travail, comme `font-lock-string-face`, `font-lock-comment-face`, et d'autres. Le meilleur moyen de les connaître toutes est d'utiliser la complétion du nom de face dans `set-face-foreground`.

Pour changer les couleurs ou les polices utilisées par le mode Verrouillage de Police pour fontifier différentes parties de texte, changez juste ces faces. Il existe deux moyens de faire cela :

- Invoquez `M-x set-face-foreground` ou `M-x set-face-background` pour changer les couleurs d'une face particulière utilisée par le Verrouillage de Police. See Section 11.1 [Faces], page 103. La commande `M-x list-faces-display` affiche toutes les faces actuellemen connues d'Emacs, dont celles utilisées par le Verrouillage de Police.
- Personnalisez les faces de manière interactive avec `M-x customize-face`, comme décrit dans Section 31.2.2.3 [Face Customization], page 463.

Pour obtenir le plein bénéfice du mode Verrouillage de Police, vous devez choisir une police ayant des variantes grasse, italique et grasse-italique ; autrement, vous devez avoir un écran couleur ou à niveau de gris.

La variable `font-lock-maximum-decoration` spécifie le niveau préféré de fontification, pour les modes fournissant plusieurs niveaux. Le niveau 1 est le niveau de fontification moindre ; certains modes fournissent des niveaux pouvant aller jusqu'à 3. La valeur par défaut est "aussi haut que possible." Vous pouvez spécifier un entier, qui s'applique à tous les modes, ou différents nombres pour des modes majeurs particuliers ; par exemple, pour utiliser le niveau 1 pour les modes C/C++, et le niveau par défaut autrement, utilisez :

```
(setq font-lock-maximum-decoration
      '((c-mode . 1) (c++-mode . 1)))
```

La fontification peut être trop longue pour de longs tampons, vous pouvez donc la supprimer. La variable `font-lock-maximum-size` spécifie une taille de tampon, à partir de laquelle la fontification du tampon est supprimée.

La fontifiction des commentaires et des chaînes (ou fontification "syntaxique") repose sur l'analyse de la structure syntaxique du texte du tampon. Pour des raisons de rapidité, certains modes comme le mode C et le mode Lisp reposent sur une convention spéciale : une parenthèse ouvrante dans la colonne la plus à gauche définit toujours le début d'une définition, et se trouve donc toujours en dehors de toute chaîne ou commentaire. (See Section 22.4 [Defuns], page 277.) Si vous ne suivez pas cette convention, le mode Verrouillage de Police peut mal fontifier le texte se trouvant après une parentèse ouvrante placée sur la première colonne et appartenant à une chaîne ou un commentaire.

La variable `font-lock-beginning-of-syntax-function` (toujours locale au tampon) spécifie comment le mode Verrouillage de Police peut trouver une position qui est garantie être en dehors de tout commentaire ou chaîne. Dans les modes utilisant la convention de la parenthèse ouvrante sur la colonne la plus à gauche, la valeur par défaut de la variable est `beginning-of-defun`—ceci indique au mode Verrouillage de Police d'utiliser cette convention. Si vous définissez cette variable à `nil`, le Verrouillage de Police ne repose plus sur la convention. Ceci annule les résultats incorrects, mais le prix est que, dans certains cas, la fontification pour une modification de texte doit rescanner le texte du tampon depuis le début du tampon. Ceci peut considérablement ralentir l'affichage lors de défilements, particulièrement si vous êtes près de la fin d'un tampon de grande taille.

Des motifs à mettre en surbrillance par le Verrouillage de Police existent pour un grand nombre de modes, mais vous pouvez vouloir fontifier des motifs supplémentaires. Vous pouvez utiliser la fonction `font-lock-add-keywords`, pour ajouter vos propres motifs à mettre en surbrillance pour un mode particulier. Par exemple, pour mettre en surbrillance les mots '`FIXME:`' dans les commentaires C, utilisez :

```
(font-lock-add-keywords
 'c-mode
 '(("\\<\\(FIXME\\):" 1 font-lock-warning-face t)))
```

11.3 Mode Surbrillance des Changements

Utilisez `M-x highlight-changes-mode` pour activer un mode mineur utilisant les faces (les couleurs, typiquement) pour indiquer quelles parties du tampon ont été changées récemment.

11.4 Surbrillance Interactive par Correspondance

Il est parfois utile de mettre en surbrillance les chaînes qui correspondent à une certaine expression rationnelle. Par exemple, vous pourriez vouloir voir toutes les références à une certaine variable dans le fichier source d'un programme, ou mettre en surbrillance certaines parties dans une sortie volumineuse d'un programme, ou faire ressortir certains *clichés* d'un article.

Utilisez la commande `M-x hi-lock-mode` pour activer un mode mineur vous permettant de spécifier des expressions rationnelles du texte à mettre en surbrillance. Le mode Hi-Lock fonctionne comme le mode Verrouillage de Police (see Section 11.2 [Font Lock], page 106), excepté qu'il vous laisse spécifier explicitement quelles parties de texte doivent être en surbrillance. Vous contrôlez le mode Hi-Lock avec ces commandes :

`C-x w h` *regexp* ⟨RET⟩ *face* ⟨RET⟩
: Met en surbrillance le texte correspondant à *regexp* en utilisant la face *face* (`highlight-regexp`). En utilisant cette commande plusieurs fois, vous pouvez mettre en surbrillance plusieurs parties de texte de différentes manières.

`C-x w r` *regexp* ⟨RET⟩
: Supprime la surbrillance *regexp* (`unhighlight-regexp`). Vous edvez entrer une des expressions rationnelles actuellement spécifiées pour la surbrillance. (Vous pouvez utiliser la complétion, ou un menu, pour saisir l'une d'elle plus facilement.)

`C-x w l` *regexp* ⟨RET⟩ *face* ⟨RET⟩
: Met en surbrillance les lignes contenant une correspondance pour *regexp*, en utilisant la face *face* (`highlight-lines-matching-regexp`).

`C-x w b`
: Insére toutes les paires regexp/face actuelles pour la surbrillance dans le tampon au point, avec des délimiteurs de commentaire pour empêcher la modification de votre programme. Ce raccourci lance la commande `hi-lock-write-interactive-patterns`.

Ces motifs seront lus le prochaine fois que vous lirez le fichier alors que le mode Hi-Lock sera actif, ou lorsque vous utiliserez la commande `M-x hi-lock-find-patterns`.

`C-x w i` Relit les paires regexp/face dans le tampon courant (`hi-lock-write-interactive-patterns`). La liste des paires est trouvée quel que soit l'endroit dans le tampon où elle se trouve.

Cette commande ne fait rien si le mode majeur est membre de la liste `hi-lock-exclude-modes`.

11.5 Espaces à la Traîne

Il est courant de laisser des espaces non nécessaires en fin de ligne sans y faire attention. Dans la plupart des cas, cet *espace à la traîne* n'a pas d'effet, mais dans certaines circonstances spéciales il peut en avoir.

Vous pouvez rendre les espaces à la traîne visibles à l'écran en définissant la variable `show-trailing-whitespace` à `t`. Emacs affiche alors les espaces à la traîne avec la face `trailing-whitespace`.

Les espaces à la traîne sont définis comme des espaces ou tabulations à la fin d'une ligne. Mais les espaces à la traîne ne sont pas affichés dans le cas spécial où le point se trouve à la fin de la ligne contenant cet espace à la traîne. (cela est déplaisant lorsque vous tapez du texte, et la location du point est suffisante dans ce cas pour montrer la présence des espaces.)

Emacs peut indiquer les lignes vides à la fin du tampon avec un dessin particulier dans la marge gauche de la fenêtre. Pour activer cette fonctionnalité, définissez la variable locale au tampon `indicate-empty-lines` à une valeur non `nil`. La valeur par défaut de cette variable est contrôlée par la variable `default-indicate-empty-lines` ; en définissant cette variable, vous pouvez activer ou désactiver cette fonctionnalité pour tous les nouveaux tampons.

11.6 Défilement

Si un tampon contient du texte trop long pour tenir entièrement dans une fenêtre affichant ce tampon, Emacs affiche une portion contigüe du texte. La portion affichée contient toujours le point.

Faire *Défiler* veut dire déplacer le texte vers le haut ou vers le bas dans la fenêtre pour que différentes parties du texte soient visibles. Faire défiler vers l'avant veut dire que le texte se déplace vers le haut, et du nouveau texte apparaît en bas. Faire défiler en arrière déplace le texte vers le bas et du nouveau texte apparaît en haut.

Le défilement arrive automatiquement si vous déplacez le point au delà du haut ou du bas de la fenêtre. Vous pouvez aussi demander explicitement un défilement avec les commandes décrites dans cette section.

`C-l`	Efface l'écran et réaffiche, faisant défiler la fenêtre sélectionnée pour y centrer le point verticalement (`recenter`).
`C-v`	Fait défiler vers l'avant (une fenêtre entière ou un nombre spécifié de lignes) (`scroll-up`).
⟨NEXT⟩	Même chose, fait défiler vers l'avant.
`M-v`	Fait défiler vers l'arrière (`scroll-down`).
⟨PRIOR⟩	Même chose, fait défiler vers l'arrière.
arg `C-l`	Fait défiler pour que le point soit en ligne *arg* (`recenter`).
`C-M-l`	Fait défiler heuristiquement pour porter des informations utiles à l'écran (`reposition-window`).

La commande de défilement la plus élémentaire est `C-l` (`recenter`) sans argument. Elle efface entièrement l'écran et réaffiche toutes les fenêtres. De plus, elle fait défiler la fenêtre sélectionnée pour que le point soit à mi-chemin entre le haut et le bas de la fenêtre.

Les commandes de défilement `C-v` et `M-v` vous permettent de déplacer tout le texte de la fenêtre de plusieurs lignes vers le haut ou le bas. `C-v` (`scroll-up`) avec un argument vous montre autant de ce nombre de lignes en bas de la fenêtre, déplaçcant le texte et le point ensemble vers le haut, comme `C-l` le ferait. `C-v` avec un argument négatif vous montre de nouvelles lignes en haut de la fenêtre. `M-v` (`scroll-down`) est similaire à `C-v`, mais déplace le texte dans la direction opposée. Les touches de fonction ⟨NEXT⟩ et ⟨PRIOR⟩ sont équivalentes à `C-v` et `M-v`.

Les noms de commandes de défilement sont basés sur la direction dans laquelle le texte se déplace dans la fenêtre. Ainsi, la commande pour faire défiler vers l'avant est appelée `scroll-up` car elle déplace le texte vers le haut.

Pour lire un tampon une fenêtre à la fois, utilisez `C-v` sans argument. Elle prend les deux dernières lignes en bas de la fenêtre et les place en haut, et les fait suivre d'une fenêtre entière de lignes non visibles précédemment. Si le point se trouvait dans le texte disparu en haut de la fenêtre, il est déplacé au nouveau début de fenêtre. `M-v` sans argument fait défiler en arrière, avec un chevauchement similaire. Le nombre de lignes de chevauchement lors d'un `C-v` ou `M-v` est contrôlé par la variable `next-screen-context-lines` ; elle est égale à 2 par défaut.

Certains utilisateurs préfèrent que les commandes de défilement par écran laissent le point sur la même ligne de l'écran. Pour utiliser cette fonctionnalité, mettez la variable `scroll-preserve-screen-position` à une valeur non `nil`. Ce mode est commode pour parcourir un fichier en le faisant défiler

écran par écran ; si vous revenez à l'écran d'où vous avez commencé, le point se retrouve sur la ligne initiale. Cependant, ce mode n'est pas commode lorsque vous vous déplacez vers l'écran suivant afin d'y placer le point.

Un autre moyen de faire défiler le texte est avec `C-l` avec un argument numérique. `C-l` n'efface pas l'écran lorsqu'on lui passe un argument ; elle fait seulement défiler la fenêtre sélectionnée. Avec un argument positif *n*, elle repositionne le texte pour placer le point sur la *n*ième ligne. Un argument égal à zéro place le point sur la première ligne. Le point ne se déplace pas en respect avec le texte ; plus exactement, le texte et le point se déplacent de façon rigide sur l'écran. `C-l` avec un argument négatif place le point au nombre spécifié de lignes du bas de l'écran. Par exemple, `C-u - 1 C-l` place le point sur la ligne du bas, et `C-u - 5 C-l` le place à 5 lignes du bas. `C-u` seul comme argument, comme dans `C-u C-l`, déplace le point au centre de la fenêtre sélectionnée.

La commande `C-M-l` (`reposition-window`) fait défiler la fenêtre courante heuristiquement de façon à obtenir des informations intéressantes à l'écran. Par exemple, dans un fichier Lisp, cette commande esasie de faire tenir le defun courant en entier dans lécran, si c'est possible.

Le défilement arrive automatiquement si le point est déplacé en dehors de la portion visible du texte lorsqu'il est temps d'afficher. Normalement, le défilement automatique centre le point verticalement dans l'écran. Cependant, si vous mettez la variable `scroll-conservatively` à une faible valeur *n*, alors si vous déplacez le point juste en dehors de l'écran (moins de *n* lignes), Emacs fait défiler le texte juste suffisamment pour faire revenir le point à l'écran. Par défaut, `scroll-conservatively` est à 0.

Lorsque la fenêtre défile d'une longue distance, vous pouvez contrôler l'agressivité du défilement, en définissant les variables `scroll-up-aggressively` et `scroll-down-aggressively`. La valeur de `scroll-up-aggressively` doit être soit `nil`, soit une fraction *f* entre 0 et 1. Une fraction spécifie où placer le point sur l'écran lors d'un défilement vers le haut. Plus précisément, lorsqu'une fenêtre défile vers le haut car le point est au dessus du début de la fenêtre, la nouvelle position de départ est choisie pour placer le point à une fraction *f* de la hauteur de la fenêtre à partir d'en haut. Plus grand est *f*, plus agressif est le défilement.

`nil`, qui est la valeur par défaut, place le point au centre. C'est donc équivalent à 0.5.

De même, `scroll-down-aggressively` est utilisé pour les défilements vers le bas. La valeur, *f*, spécifie à combien du bas de la fenêtre doit être placé le point ; ainsi, comme pour `scroll-up-aggressively`, une valeur plus grande est plus agressive.

La variable `scroll-margin` indique à quelle distance le point peut s'approcher du haut ou du bas d'une fenêtre. Sa valeur est un nombre de lignes d'écran ; si le point est déplacé dans ce nombre de lignes en haut ou en bas de la fenêtre, Emacs recentre la fenêtre. Par défaut, `scroll-margin` est à 0.

11.7 Défilement horizontal

Défilement horizontal veut dire déplacer toutes les lignes d'une fenêtre de côté—ainsi une partie du texte près de la marge gauche n'est pas affichée. Emacs fait cela automatiquement, dans toute fenêtre utilisant la troncation de ligne plutôt que la continuation : lorsque le point sort à droite ou à gauche de l'écran, Emacs fait défiler le tampon horizontalement pour rendre le point visible.

Lorsqu'une fenêtre a défilée verticalement, les lignes de texte sont tronquées plutôt que continuées (see Section 4.8 [Continuation Lines], page 49), avec un '`$`' apparaissant sur la première colonne lorsqu'il y a du texte tronqué à gauche, et sur la dernière colonne lorsque du texte est tronqué à droite.

Vous pouvez utiliser ces commandes pour faire du défilement horizontal explicite.

`C-x <` Fait défiler le texte de la fenêtre courante vers la gauche (`scroll-left`).

`C-x >` Fait défiler vers la droite (`scroll-right`).

La commande `C-x <` (`scroll-left`) fait défiler la fenêtre courante vers la gauche de *n* colonnes avec un argument *n*. Ceci déplace une partie du début de chaque ligne en dehors de la limite gauche de la fenêtre. Sans argument, elle fait défiler d'une largeur de fenêtre (moins deux colonnes, pour être précis).

`C-x >` (`scroll-right`) fait défiler de manière similaire vers la droite. La fenêtre ne peut pas être davantage défilée une fois qu'elle est affichée normalement (avec chacune des lignes commençant à la marge gauche de la fenêtre) ; tenter de le faire n'a pas d'effet. Cela veut dire que vous n'avez pas à calculer précisément l'argument pour `C-x >` ; un argument suffisamment grand fera revenir l'affichage normal.

Si vous faites défiler horizontalement une fenêtre à la main, une borne inférieure est placée pour le défilement horizontal automatique. Le défilement horizontal continuera de faire défiler la fenêtre, mais jamais au delà vers la droite de la quantité précédemment utilisée avec `scroll-left`.

Pour désactiver le défilement horizontal automatique, définissez la variable `automatic-hscrolling` à `nil`.

11.8 Mode Suivi

Le *Mode Suivi* est un mode mineur qui fait que deux fenêtres montrant le même tampon défilent comme une seule grande "fenêtre virtuelle." Pour utiliser le mode Suivi, allez dans un cadre avec une seule fenêtre, coupez-le

en deux fenêtres côte-à-côte en utilisant `C-x 3`, puis tapez `M-x follow-mode`. À partir de là, vous pouvez éditer le tampon dans l'une des deux fenêtres, ou faire défiler une d'elles ; l'autre fenêtre défile en même temps.

Dans le mode Suivi, si vous déplacez le point en dehors de la portion visible d'une fenêtre et à l'intérieur de la portion visible de l'autre fenêtre, l'autre fenêtre est sélectionnée—encore une fois, en traitant les deux fenêtres comme si elles faisaient partie d'une fenêtre plus grande.

Pour désactiver le mode Suivi, tapez `M-x follow-mode` une seconde fois.

11.9 Affichage Sélectif

Emacs a la possibilité de cacher des lignes indentées de plus qu'un certain nombre de colonnes (vous spécifiez ce nombre de colonnes). Vous pouvez utiliser ceci pour obtenir un aperçu d'une partie d'un programme.

Pour cacher des lignes, tapez `C-x $` (`set-selective-display`) avec un argument numérique *n*. Les lignes avec au moins *n* colonnes d'indentation disparaissent alors de l'écran. La seule indication de leur présence est que trois points ('`...`') apparaissent à la fin de chaque ligne visible suivie d'une ou plusieurs lignes cachées.

Les commandes `C-n` et `C-p` vous déplacent à travers les lignes cachées comme si elles n'étaient pas là.

Les lignes cachées sont toujours présentes dans le tampon, et la plupart des commandes d'édition les voit comme d'habitude, et vous pouvez trouver le point au milieu du texte caché. Lorsque ceci arrive, le curseur apparaît à la fin de la ligne précédente, après les trois points. Si le point est à la fin de la ligne visible, avant le caractère newline qui la finit, le curseur apparaît avant les trois points.

Pour rendre de nouveau toutes les lignes visibles, tapez `C-x $` sans argument.

Si vous mettez la variable `selective-display-ellipses` à `nil`, les trois points n'apparaissent pas à la fin d'une ligne qui précéde des lignes cachées. Il n'y a alors pas d'indication visible sur les lignes cachées. Cette variable devient automatiquement locale lorsqu'elle est modifiée.

11.10 Caractéristiques optionnelles de la ligne de mode.

++Le numéro de ligne courant du point apparaît en ligne de mode lorsque le mode Numéro de ligne est activé. Utilisez la commande `M-x line-number-mode` pour activer et désactiver ce mode ; il est normalement actif. Le numéro de ligne apparaît avant le pourcentage du tampon *pos*,

avec la lettre ‘L’ pour indiquer ce que c’est. See Section 31.1 [Minor Modes], page 455, pour plus d’informations sur les modes mineurs et sur la manière d’utiliser cette commande.

Si le tampon est très long (si son nombre de lignes est supérieur à la valeur de `line-number-display-limit`), alors le numéro de ligne n’apparaît pas. Emacs ne calcule pas le numéro de ligne lorsque le tampon est long, car cela pourrait être trop lent. Définissez cette variable à `nil` pour supprimer cette limite. Si vous avez restreint le tampon (see Section 30.9 [Narrowing], page 443), le numéro de ligne affiché est relatif àla portion accessible du tampon.

Vous pouvez aussi afficher le numéro de colonne courant en activant le mode Numéro de Colonne. Il affiche le numéro de colonne courant précédé par la lettre ‘C’. Tapez `M-x column-number-mode` pour activer et désactiver ce mode.

Emacs peut optionnellement afficher l’heure et la charge du système dans toutes les lignes de mode. Pour activer cette caractéristique, tapez `M-x display-time` ou personnalisez l’option `display-time-mode`. L’information ajoutée à la ligne de mode apparaît généralement après le nom du tampon, et avant les noms de modes entre parenthèses. Elle ressemble à :

hh`:`*mm*`pm` *l.ll*

Ici *hh* et *mm* sont les heures et les minutes, toujours suivies de ‘`am`’ ou ‘`pm`’. *l.ll* est le nombre moyen de processus exécutés sur le système récemment. (Certains champs peuvent manquer si votre système d’exploitation ne peut pas les supporter.) Si vous préférez afficher l’heure au format 24 heures, mettez la variable `display-time-24hr-format` à `t`.

Le mot ‘`Mail`’ apparaît après la charge du système s’il y a du courrier pour vous que vous n’avez pas encore lu. Dans un environnement graphique vous pouvez utiliser une icône plutôt que ‘`Mail`’ en personnalisant `display-time-use-mail-icon` ; ceci peut faire gagner un peu de place dans la ligne de mode. Vous pouvez personnaliser `display-time-mail-face` pour rendre l’indicateur de courrier proéminent.

Par défaut, la ligne de mode est dessinée sur les systèmes graphiques comme un bouton 3D relâché. Selon la police utilisée pour le texte de la ligne de mode, cette ligne peut utiliser plus d’espace qu’une ligne de texte dans une fenêtre, et la dernière ligne de la fenêtre peut être partiellement cachée. Cest-à-dire que la fenêtre affiche un nombre non entier de lignes de texte. Si vous n’aimez pas cet effet, vous pouvez désactiver l’apparence 3D de la ligne de mode en personnalisant les attributs de la face `mode-line` dans votre fichier init ‘`.emacs`’, de cette manière :

```
(set-face-attribute 'mode-line nil :box nil)
```

Alternativement, vous pouvez désactiver l’attribut 3D dans votre fichier ‘`.Xdefaults`’ :

```
Emacs.mode-line.AttributeBox: off
```

11.11 Comment le Texte est Affiché

Les caractères ASCII imprimables (codes octaux de 040 à 0176) dans les tampons d'Emacs sont affichés sous forme graphique. Ainsi des caractères imprimables non ASCII multi-octets (codes octaux supérieurs à 0400).

Certains caractères de contrôle ASCII sont affichés d'une manière particulière. Le caractère newline (code octal 012) est affiché en démarrant un nouvelle ligne. Le caractère tab (code octal 011) est affiché en vous déplaçant à la colonne d'arrêt de tabulation suivante (normalement tous les 8 colonnes).

Les autres caractères de contrôle ASCII sont normalement affichés comme un circonflexe (‘^’) suivi de la version non-contrôle du caractère ; ainsi, control-A est affiché ‘^A’.

Les caractères non ASCII de 0200 à 0237 (octal) sont affichés avec des séquences escape octales ; ainsi, le code caractère 0230 (octal) est affiché ‘\230’. L'affichage des codes caractère de 0240 à 0377 (octal) peut être soit une séquence escape soit graphique. Ils n'apparaissent normalement pas dans des tampons multi-octets mais s'ils apparaissent, ils sont affichés avec des graphiques Latin-1. Dans le mode uni-octet, si vous activez l'affichage européen, ils sont affichés en utilisant leurs propres graphiques (en supposant que votre terminal les supporte), autrement comme séquences escape. See ⟨undefined⟩ [Single-Byte Character Support], page ⟨undefined⟩.

11.12 Personnalisation de l'Affichage

Cette section contient uniquement des informations sur la personnalisation. Il est conseillé aux utilisateurs débutants de le sauter.

La variable `mode-line-inverse-video` est un moyen rendu obsolète de contrôler si la ligne de mode est affichée en inversion vidéo ; la méthode préférée pour faire ceci est de changer la face `mode-line`. See Section 1.3 [Mode Line], page 26. Si vous spécifiez la couleur d'avant-plan pour la face `mode-line`, et `mode-line-inverse-video` est non `nil`, alors la couleur de fond par défaut pour cette face est la couleur d'avant-plan habituelle. See Section 11.1 [Faces], page 103.

Si la variable `inverse-video` est non-`nil`, Emacs tente d'inverser toutes les lignes de l'affichage de ce qu'elles sont d'habitude.

Si la variable `visible-bell` est non-`nil`, Emacs tente de faire clignoter l'écran entier lorsqu'il devrait normalement faire un bruit de sonnerie. Cette n'a pas d'effet si votre terminal n'a pas le moyen de faire clignoter l'écran.

Lorsque vous revenez à Emacs après l'avoir suspendu, Emacs efface normalement l'écran puis le réaffiche entièrement. Sur certains terminaux avec plus d'une page de mémoire, il est possible d'arranger l'entrée termcap pour que les entrées ‘`ti`’ et ‘`te`’ (sortie sur le terminal lorsqu'Emacs est commencé

ou terminé, respectivement) fassent passer d'une page mémoire à une autre de manière à utiliser une page pour Emacs et une autre page pour les autres sorties. Vous pourrez alors mettre la variable `no-redraw-on-reenter` à non-`nil` ; qui dit à Emacs d'assumer, lorsqu'on y entre de nouveau, que la page d'écran a le même contenu que lorsqu'on l'a suspendu.

La variable `echo-keystrokes` contrôle l'écho des touches multi-caractères ; sa valeur est le nombre de secondes à attendre avant de démarrer l'écho, ou zéro pour aucun écho. See Section 1.2 [Echo Area], page 24.

Si la variable `ctl-arrow` est `nil`, les caractères de contrôle dans le tampon sont affichés sous forme de séquences escape octales, excepté pour les caractères newline et tab. Modifier la valeur de `ctl-arrow` rend cette variable locale au tampon courant ; jusque-là, la valeur par défaut est utilisée. La valeur par défaut est initialement `t`. See section "Display Tables" in *The Emacs Lisp Reference Manual*.

Normalement un caractère tab dans un tampon est représenté par des espaces qui vous emmènent à l'arrêt de tabulation suivant, et les arrêts de tabulation se trouvent à des intervalles de 8 espaces. Le nombre d'espaces par tabulation est contrôlé par la variable `tab-width`, qui devient locale en la changeant, comme `ctl-arrow`. Notez que la manière dont le catactère tab est affiché n'a rien à voir avec la définition de ⟨TAB⟩ comme une commande. La variable `tab-width` doit avoir une valeur comprise entre 1 et 1000, inclus.

Si la variable `truncate-lines` est non-`nil`, alors chaque ligne de texte prend juste une ligne d'écran pour s'afficher ; si la ligne de texte est trop longue, seulement une partie est affichée. Si `truncate-lines` est `nil`, alors les lignes de texte trop longues s'affichent sur plusieurs lignes d'écran, suffisamment pour montrer tout le texte de la ligne. See Section 4.8 [Continuation Lines], page 49. Modifier la valeur de `truncate-lines` rend cette variable locale au tampon courant ; jusque-là, la valeur par défaut est utilisée. La valeur par défaut est initialement `nil`.

Si la variable `truncate-partial-width-windows` est non-`nil`, elle force la troncation plutôt que la continuation dans toute fenêtre dont la largeur est inférieure à la largeur de l'écran ou du cadre, en regard de la valeur de `truncate-lines`. Pour des informations sur les fenêtres côte-à-côte, voir Section 16.2 [Split Window], page 196. Voir aussi section "Display" in *The Emacs Lisp Reference Manual*.

La variable `baud-rate` indique la vitesse de sortie du terminal. Modifier cette variable ne change pas la vitesse de la transmission de données, mais la valeur est utilisée pour certains calculs. Sur des terminaux, elle affecte le "padding," et les décisions pour savoir s'il faut faire défiler une partie de l'écran ou plutôt le redessiner.

Sur des systèmes graphiques, `baud-rate` est seulement utilisé pour déterminer la fréquence d'interrogation des entrées en attente lors du rafraîchissement de l'affichage. Une valeur plus grande de `baud-rate` indique d'interroger les entrées en attente moins fréquemment.

Vous pouvez personnaliser la manière dont un caractère particulier est affiché au moyen d'une table d'affichage. See section "Display Tables" in *The Emacs Lisp Reference Manual*.

Sur un système graphique, Emacs peut optionnellement afficher un pointeur de souris différent pour indiquer qu'Emacs est occupé. Pour activer ou désactiver cette fonctionnalité, personnalisez le groupe `cursor`. Vous pouvez aussi contrôler la quantité de temps pendant laquelle Emacs doit rester occupé avant que l'indicateur d'occupation soit affiché, en définissant la variable `hourglass-delay`.

11.13 Affichage du Curseur

Il y a plusieurs moyens de personnaliser l'affichage du curseur. `M-x hl-line-mode` active ou désactive un mode mineur global qui met en surbrillance la ligne contenant le point. Sur des systèmes graphiques, la commande `M-x blink-cursor-mode` active ou désactive le clignotement du curseur. (Sur des terminaux, le terminal lui-même fait clignoter le curseur, et eamcs n'a aucun contrôle dessus.)

Vous pouvez personnaliser la couleur du curseur, et s'il clignote ou non, en utilisant le groupe de personnalisation `cursor` (see Section 31.2.2 [Easy Customization], page 459).

Lors d'un affichage sur un système graphique, Emacs peut optionnellement dessiner le bloc curseur aussi large que le caractère sous le curseur—par exemple, si le curseur est sur une tabulation, il couvrira la largeur totale ocuupée par ce caractère de tabulation. Pour activer cette fonctionnalité, définissez la variable `x-stretch-cursor` à une valeur non `nil`.

Normalement, le curseur dans les fenêtres non sélectionnées est montrée comme une boîte vide. Pour désactiver l'affichage du curseur dans les fenêtres non sélectionnées, personnalisez l'option `show-cursor-in-non-selected-windows`, ou définissez la variable `cursor-in-non-selected-windows` à `nil`.

12 Recherche et Remplacement

Comme d'autres éditeurs, Emacs a des commandes pour rechercher des occurences de chaînes. La commande de recherche principale n'est pas habituelle dans le fait qu'elle soit *incrémentale* ; elle commence à chercher avant que vous ayez fini de taper la chaîne à rechercher. Il y a aussi des commandes de recherche non incrémentales ressemblant plus à celles d'autres éditeurs.

À côté de la commande habituelle `replace-string` qui trouve toutes les occurences d'une chaîne et les remplace par une autre, Emacs a une commande attachante appelée `query-replace` qui demane interactivement quelles occurences remplacer.

12.1 Recherche Incrémentale

Une recherche incrémentale commence à chercher dès que vous avez tapé le premier caractère de la chaîne à rechercher. Alors que vous tapez la chaîne recherchée, Emacs vous montre où la chaîne (comme vous l'avez tapé à ce moment) serait trouvée. Lorsque vous avez tapé assez de caractères pour identifier l'endroit recherché, vous pouvez arrêter. Selon ce que vous désirez faire ensuite, vous pouvez ou non terminer la recherche par ⟨RET⟩.

`C-s` Recherche incrémentale en avant (`isearch-forward`).

`C-r` Recherche incrémentale en arrière (`isearch-backward`).

`C-s` commence une recherche incrémentale. `C-s` lit des caractères depuis le clavier et positionne le curseur à la première occurence des caractères que vous avez tapé. Si vous tapez `C-s` puis `F`, le curseur se déplace juste après le premier '`F`'. Tapez un `O`, et voyez le curseur se déplacer juste après le premier '`FO`'. Après un autre `O`, le curseur est après le premier '`FOO`' suivant l'endroit où vous avez commencé la recherche. À chaque étape, le texte du tampon qui correspond à la chaîne recherchée est illuminée, si le terminal le permet ; à chaque étape, la chîne recherchée courante est mise à jour dans la zone de répercussion.

Si vous faîtes une erreur en tapant la chaîne à rechercher, vous pouvez annuler des caractères avec ⟨DEL⟩. chaque ⟨DEL⟩ annule le dernier caractère de la chaîne recherchée. Ceci n'arrive pas avant qu'Emacs soit prêt à lire un nouveau caractère ; il doit d'abord soit trouver, soit échouer dans la recherche du caractère que vous vuolez annuler. Si vous ne voulez pas attendre que cela arrive, utilisez `C-g` comme décrit plus bas.

Lorsque vous êtes satisfait de la place que vous avez atteint, vous pouvez taper ⟨RET⟩, qui stoppe la recherche, laissant le curseur où la recherche l'a mené. Se plus, toute commande n'ayant pas rapport avec la recherche stoppe

la recherche, puis est exécutée. Ainsi, taper `C-a` fait terminer la recherche puis vous déplace en début de ligne. ⟨RET⟩ est nécessaire seulement si la commande suivante que vous voulez taper est un caractère imprimable, ⟨DEL⟩, ⟨RET⟩, ou un autre caractère de contrôle qui est utilisé durant la recherche (`C-q`, `C-w`, `C-r`, `C-s`, `C-y`, `M-y`, `M-r`, or `M-s`).

Parfois vous recherchez '`FOO`' et le trouvez, mais pas celui que vous espériez trouver. Il y avait un second '`FOO`' que vous aviez oublié, précédant celui que vous recherchez. Dans ce cas, tapez de nouveau `C-s` pour aller à la prochaine occurence de la chaîne recherchée. Vous pouvez répéter cela un nombre quelconque de fois. Si vous en faîtes trop, vous pouvez annuler des caractères `C-s` avec ⟨DEL⟩.

Après avoir quitté une recherche, vous pouvez de nouveau chercher la même chaîne en tapant juste `C-s C-s` : le premier `C-s` est la touche qui invoque la recherche incrémentale, et le second `C-s` signifie "cherche de nouveau."

Pour réutiliser des recherches ultérieures, utilisez l'*anneau de recherche*. Les commandes `M-p` et `M-n` vous déplace dans l'anneau pour retrouver une chaîne de recherche à utiliser. Ces commandes introduisent l'élément de l'anneau de recherche sélectionné dans le mini-tampon, où vous pouvez l'éditer. Tapez `C-s` ou `C-r` pour terminer l'édition de la chaîne et la rechercher.

Si votre chaîne n'est pas trouvée du tout, la zone de répercussion indique '`Failing I-Search`'. Le curseur se trouve après la place où Emacs a trouvé le plus de caractères possibles de la chaîne recherchée. Ainsi, si vous recherchez '`FOOT`', et qu'il n'y a pas de '`FOOT`', vous pourrez voir le curseur après le '`FOO`' de '`FOOL`'. À partir de là, vous pouvez faire un certain nombre de choses. Si votre chaîne a été mal tapée, vous pouvez en supprimer une partie et la corriger. Si la place que vous avez trouvée vous convient, vous pouvez taper ⟨RET⟩ ou toute autre commande pour "accepter ce que la recherche a offerte." Ou vous pouvez taper `C-g`, qui retire de la chaîne recherchée les caractères non trouvés (le '`T`' de '`FOOT`'), laissant ceux trouvés (le '`FOO`' de '`FOOT`'). Un second `C-g` à ce point annule entièrement la recherche, et renvoie le point à l'endroit où il se trouvait en début de recherche.

Une lettre majuscule dans la chaîne recherchée rend la recherche sensible à la casse. Si vous supprimez le caractère majuscule de la chaîne de recherche, il cesse d'avoir cet effet. See Section 12.6 [Search Case], page 131.

Pour rechercher un caractère newline, tapez `C-j`. Pour rechercher un autre caractère de contrôle, comme Control-S ou carriage return, vous devez le citer en tapant d'abord `C-q`. Cette utilisation de `C-q` est analogue à son utilisation pour l'insertion (see Section 4.1 [Inserting Text], page 41) : elle permet au caractère suivant d'être traité comme tout autre caractère "ordinaire" est traité dans le même contexte. Vous pouvez aussi spécifier un caractère par son code octal : entrez `C-q` suivi d'une séquence de chiffres octaux.

Pour rechercher des caractères non ASCII, vous devez utiliser une méthode d'entrée (see Section 18.4 [Input Methods], page 221). Si une méthode d'entrée est activée dans le tampon courant lorsque vous commencez la recherche, vous pouvez aussi l'utiliser lorsque vous entrez la chaîne à rechercher. Emacs indique cela en incluant la mnémonique de la méthode d'entrée dans son prompt, comme ceci :

```
I-search [im]:
```

où *im* est la mnémonique de la méthode d'entrée active. Vous pouvez activer ou désactiver la méthode d'entrée lorsque vous tapez la chaîne à rechercher avec `C-\` (`isearch-toggle-input-method`). Vous pouvez activer une certaine méthode d'entrée (non par défaut) avec `C-^` (`isearch-toggle-specified-input-method`), qui vous demande le nom de la méthode d'entrée. Notez que la méthode d'entrée que vous activez durant la recherche incrémentale est aussi activée pour le tampon courant.

Si une recherche échoue et que vous demandez de la recommencer en tapant de nouveau `C-s`, il recommence à partir du début du tampon. Recommencer une recherche inverse échouée avec `C-r` recommence depuis la fin. Ceci est appelé le *bouclage*. '`Wrapped`' (bouclé) apparaît dans le prompt de recherche une fois que c'est arrivé. Si vous continuez et dépassez le point d'origine de votre recherche, il se change en '`Overwrapped`' (reboulcé), qui veut dire que vous revisitez des correspondances que vous avez déjà vues.

Le caractère "quitter" `C-g` opère de manière spéciale durant la recherche ; ce qu'il fait exactement dépend du status de la recherche. Si la recherche a trouvé ce que vous avez spécifié et attend une entrée, `C-g` annule la recherche entière. Le curseur revient au point de départ de la recherche. Si `C-g` est tapé alors que des caractères de la chaîne de recherche n'ont pas été trouvés—car Emacs recherche encore ces caractères, ou parce qu'il a échoué dans cette recherche—alors les caractères de la chaîne qui n'ont pas été trouvés sont retirés de la chaîne de recherche. Une fois ces caractères retirés, la recherche est alors satisfaite et attend une nouvelle entrée, ainsi un nouveau `C-g` annulera la recherche entière.

Vous pouvez changer pour rechercher en arrière avec `C-r`. Si une recherche échoue parce qu'elle a été initiée trop en avant dans le tampon, vous devriez faire ça. Des `C-r` répétés continuent de rechercher d'autres occurences précédentes. Un `C-s` commence à chercher vers l'avant de nouveau. `C-r` dans une recherche peut être annulé avec ⟨DEL⟩.

Si vous savez initialement que vous voulez chercher en arrière, vous pouvez utiliser `C-r` plutôt que `C-s` pour commencer la recherche, car `C-r` comme touche exécute une commande (`isearch-backward`) qui recherche en arrière. Une recherche arrière trouve des correspondances qui sont entièrement avant le point de départ, exactement comme la recherche avant trouve des correspondances qui commencent après ce point.

Les caractères `C-y` et `C-w` peuvent être utilisés durant la recherche incrémentale pour prendre du texte dans un tampon et le placer dans la

chaîne de recherche. Ceci rend facile de rechercher une autre occurence du texte qui se trouve au point. `C-w` copie le mot après le point dans la chaîne de recherche, avançant le point après ce mot. Un autre `C-s` pour répéter la recherche va alors rechercher une chaîne contenant ce mot. `C-y` est similaire à `C-w` mais copie toute la suite de la ligne dans la chaîne de recherche. `C-y` et `C-w` convertissent toutes deux le texte qu'elles copient en minuscule si la recherche n'est pas sensible à la casse ; c'est ainsi que la recherche reste insensible à la casse.

Le caractère `M-y` copie le texte du presse-papiers dans la chaîne de recherche. Elle utilise le même texte que `C-y` comme commande collerait. `Mouse-2` dans la zone de répercussion a le même effet. See Section 9.2 [Yanking], page 89.

Lorsque vous quittez la recherche incrémentale, elle place la marque à l'endroit où *était* le point avant la recherche. Ceci est utile pour retourner à cet endroit. En mode de Marque Transitoire, la recherche incrémentale place la marque sans l'activer, et le fait seulement si la marque n'est pas déjà active.

Lorsque vous faites une petite pause durant une recherche incrémentale, toutes les autres correspondances possibles sont mises en surbrillance. Ceci rend plus facile d'anticiper l'endroit où vous vous trouverez en tapant `C-s` ou `C-r` pour répéter la recherche. Le court délai avant la mise en surbrillance des autres correspondances aide à indiquer quelle correspondance est celle e, cours. Si vous n'aimez pas cette fonctionnalité, vous pouvez la désactiver en définissant la variable `isearch-lazy-highlight` à `nil`.

Vous pouvez contrôler l'aspect de la mise en surbrillance des correspondances en personnalisant les faces `isearch` (utilisée pour la correspondance en cours) et `isearch-lazy-highlight-face` (utilisée pour les autres correspondances).

Pour personnaliser les caractères spéciaux que la recherche incrémentale comprend, altérez ses raccourcis dans la table de touches `isearch-mode-map`. Pour une liste de raccourcis, consultez la documentation de `isearch-mode` avec `C-h f isearch-mode` ⟨RET⟩.

12.1.1 Recherche Incrémentale sur Terminaux Lents

La recherche incrémentale sur un terminal lent utilise un style modifié d'affichage conçu pour prendre moins de temps. Plutôt que de réafficher le tampon à chaque endroit où la recherche s'arrête, elle crée une nouvelle fenêtre d'une seule ligne et l'utilise pour afficher la ligne que la recherche a trouvée. Cette fenêtre d'une ligne entre en jeu aussitôt que le point sort du texte couramment affiché à l'écran.

Lorsque vous terminez la recherche, la fenêtre d'une ligne est enlevée. Emacs réaffiche alors la fenêtre dans laquelle la recherche a été faite, pour montrer la nouvelle position du point.

Le style d'affichage sur terminal lent est utilisé lorsque la vitesse de transmission du terminal est inférieur ou égal à la valeur de `search-slow-speed`, initialement 1200.

Le nombre de lignes à utiliser pour l'affichage de la recherche sur un terminal lent est contrôlé par la variable `search-slow-window-lines`. Sa valeur normale est 1.

12.2 Recherche Non Incrémentale

Emacs a aussi des commandes de recherche non incrémenatle conventionnelles, qui vous demandent de saisir la chaîne de recherche entière avant que la recherche commence.

`C-s` ⟨RET⟩ *chaîne* ⟨RET⟩
: Recherche *string*.

`C-r` ⟨RET⟩ *chaîne* ⟨RET⟩
: Recherche en arrière de *chaîne*.

Pour faire une recherche non incrémentale, tapez d'abord `C-s` ⟨RET⟩. Ceci vous envoie dans le mini-tampon pour saisir la chaîne de recherche ; terminez la chaîne avec ⟨RET⟩, et la recherche commence alors. Si la chaîne n'est pas trouvée, la commande de recherche échoue.

La manière dont `C-s` ⟨RET⟩ fonctionne est que `C-s` invoque la recherche incrémentale, qui est spécialement programmée pour invoquer une recherche non incrémentale si l'argument que vous lui passez est vide. (Un tel argument vide serait autrement sans utilité.) `C-r` ⟨RET⟩ fonctionne aussi de cette manière.

Cependant, des recherches non incrémentales effectuées avec `C-s` ⟨RET⟩ n'appellent pas `search-forward` tout de suite. La première chose faite est de voir si le prochain caractère est `C-w`, qui demande une recherche de mots.

Les recherches non incrémentales avant et arrière sont implémentées par les commandes `search-forward` et `search-backward`. Ces commandes peuvent être reliées à des touches de la manière usuelle. La possibilité de pouvoir les obtenir à travers les commandes de recherche incrémentale existe pour des raisons historiques, et pour éviter de trouver des séquences de touches leur convenant.

12.3 Recherche de Mots

La recherche de mot recherche une séquence de mots sans regarder comment les mots sont séparés. Plus précisément, vous tapez une suite de plusieurs mots, utilisant un espace pour les séparer, et la chaîne peut être trouvée même s'il y a plusieurs espaces, des caractères newline ou d'autres ponctuations entre les mots.

La recherche de mots est utile pour éditer un document imprimé créé avec un formateur de texte. Si vous éditez tout en regardant la version formatée et imprimée, vous ne pouvez pas dire où se trouvent les coupures de ligne dans le fichier source. Avec la recherche de mots, vous pouvez rechercher sans tenir compte de ceux-ci.

`C-s` ⟨RET⟩ `C-w` *mots* ⟨RET⟩
: Recherche *mots*, en ignorant les détails de ponctuation.

`C-r` ⟨RET⟩ `C-w` *mots* ⟨RET⟩
: Recherche arrière de *mots*, en ignorant les détails de ponctuation.

La recherche de mots est un cas particulier de recherche non incrémentale et est invoqué avec `C-s` ⟨RET⟩ `C-w`. Ceci est suivi de la chaîne de recherche, qui doit toujours être terminé par ⟨RET⟩. Étant non incrémentale, cette recherche ne commence pas avant que l'argument soit terminé. Elle marche en construisant une expression rationnelle et en la recherchant ; voir Section 12.4 [Regexp Search], page 124.

Utilisez `C-r` ⟨RET⟩ `C-w` pour effectuer une recherche de mots en arrière.

Les recherches de mots avant et arrière sont implémentées par les commandes `word-search-forward` et `word-search-backward`. Ces commandes peuvent être reliées à des touches de la manière usuelle. La possibilité de pouvoir les obtenir à travers les commandes de recherche incrémentale existe pour des raisons historiques, et pour éviter de trouver des séquences de touches leur convenant.

12.4 Recherche d'Expressions Rationnelles

Une *expression rationnelle* ("regular expression" ou *regexp*, pour raccourcir) est un motif qui dénote une classe de chaînes alternatives pouvant correspondre, dont le nombre peut être infini. Dans GNU Emacs, vous pouvez rechercher la prochaine correspondance d'une regexp, de façon incrémentale ou pas.

Une recherche incrémentale d'une regexp est faite en tapant `C-M-s` (`isearch-forward-regexp`). Cette commande lit une chaîne de recherche incrémentalement exactement comme `C-s`, mais elle traite la chaîne de recherche comme une regexp plutôt que d'en chercher une correspondance exacte dans le texte du tampon. Chaque fois que vous ajoutez du texte à la chaîne de recherche, vous rendez la regexp plus longue, et la nouvelle regexp est utilisée pour une nouvelle recherche. L'appel de `C-s` avec un argument préfixe (sa valeur n'a pas d'importance) est un autre moyen de faire une recherche incrémentale d'une regexp en avant. Pour rechercher en arrière dans le tampon, utilisez `C-M-r` (`isearch-backward-regexp`) ou `C-r` avec un argument préfixe.

Tous les caractères de contrôle qui ont un comportement spécial durant une recherche incrémentale ordinaire ont la même fonction durant une recherche incrémentale d'une regexp. Taper `C-s` ou `C-r` immédiatement après avoir commencé la recherche recommence la dernière recherche incrémentale de regexp ; toutefois, les recherches incrémentales de regexp et normales ont des valeurs par défaut indépendantes. Elles ont aussi des anneaux de recherche séparés que vous pouvez accéder avec `M-p` et `M-n`.

Si vous tapez ⟨SPC⟩ durant une recherche incrémentale de regexp, il correspond à toute séquence de caractères blancs, caractère newline inclus. Si vous voulez spécifier la correspondane à un unique espace, tapez `C-q` ⟨SPC⟩.

Notez que d'ajouter des caractères à la regexp durant la recherche incrémentale de regexp peut faire revenir le curseur en arrière et recommencer. Par exemple, si vous avez recherché '`foo`' et que vous ajoutez '`\|bar`', le curseur repart en arrière dans le cas où le premier '`bar`' précède le premier '`foo`'.

Une recherche non incrémentale d'une regexp est faite par les fonctions `re-search-forward` et `re-search-backward`. Vous pouvez les invoquer avec `M-x`, ou les relier à des touches, ou encore les invoquer par le biais des recherches incrémentales de regexp avec `C-M-s` ⟨RET⟩ et `C-M-r` ⟨RET⟩.

Si vous utilisez les commandes de recherche incrémentales de regexp avec un argument préfixe, elles effectuent une recherche de chaîne ordinaire, comme `isearch-forward` et `isearch-backward`. See Section 12.1 [Incremental Search], page 119.

12.5 Syntaxe d'Expressions Rationnelles

Les expressions rationnelles ont une syntaxe dans laquelle quelques caractères ont une signification particulière, et les autres sont *ordinaires*. Un caractère ordinaire est une expression rationnelle à laquelle correspond ce même caractère, et rien d'autre. Les caractères spéciaux sont '`$`', '`^`', '`.`', '`*`', '`+`', '`?`', '`[`', '`]`' et '`\`'. Tout autre caractère apparaissant dans l'expression rationnelle est ordinaire, à moins qu'un '`\`' le précède.

Par exemple, '`f`' n'est pas un caractère spécial, c'est donc un caractère ordinaire, et '`f`' est donc une expression rationnelle qui correspond à la chaîne '`f`' et à aucune autre chaîne. (Elle ne correspond *pas* à la chaîne '`ff`'.) De même, '`o`' est une expression rationnelle qui correspond seulement à '`o`'. (Lorsque la distinction de casse est ignorée, ces regexps correspondent aussi à '`F`' et '`O`', mais nous considérons ceci comme une généralisation de "la même chaîne", plutôt qu'une exception.)

Deux expressions rationnelles quelconques *a* et *b* peuvent être concaténées. Le résultat est une expression rationnelle qui correspond à une chaîne si *a* correspond à une partie du début de cette chaîne et *b* correspond au reste de cette chaîne.

Comme exemple simple, nous pouvons concaténer les expressions rationnelles '`f`' et '`o`' pour obtenir l'expression rationnelle '`fo`', qui correspond uniquement à la chaîne '`fo`'. Trivial. Pour faire quelque chose de non trivial, vous avez besoin d'utiliser un des caractères spéciaux. En voici une liste.

`.` (Point) est un caractère spécial auquel correspond tout caractère excepté newline. En utilisant la concaténation, vous pouvez créer des expressions rationnelles comme '`a.b`', qui correspond à toute chaîne de trois caractères commençant par '`a`' et finissant par '`b`'.

`*` n'est pas une construction par elle-même ; c'est un opérateur suffixe qui indique de faire correspondre l'expression rationnelle précédente répétitivement autant de fois que possible. Ainsi, '`o*`' correspond à un quelconque nombre de '`o`' (aucun '`o`' inclus).

'`*`' s'applique toujours à l'expression précéednte la *plus petite* possible. Ainsi, '`fo*`' a un '`o`' répétitif, et non un '`fo`' répétitif. Elle correspond à '`f`', '`fo`', '`foo`', etc.

La recherche de correspondance d'une construction '`*`' fait correspondre, immédiatement, autant de répétitions pouvant être trouvées. Elle continue alors avec la suite du motif. Si la suite échoue, un retour en arrière est utilisé, supprimant certaines des correspondances de la construction '`*`' modifiée, au cas où il serait possible de faire correspondre la suite du motif. Par exemple, en faisant correspondre '`ca*ar`' sur la chaîne '`caaar`', le '`a*`' essaie d'abord de correspondre aux trois '`a`' ; mais la suite du motif est '`ar`' et il ne reste plus que '`r`' à faire correspondre ; cet essai échoue alors. La prochaine alternative est que '`a*`' corresponde à deux '`a`' exactement. Avec ce choix, la suite de la regexp correspond parfaitement.

`+` est un opérateur suffixe, similaire à '`*`' excepté qu'il doit faire correspondre l'expression précédente au moins une fois. Ainsi, par exemple, '`ca+r`' correspond aux chaînes '`car`' et '`caaaar`' mais non à la chaîne '`cr`', alors que '`ca*r`' correspond à ces trois chaînes.

`?` est un opérateur suffixe, similaire à '`*`' excepté qu'il fait correspondre l'expression précédente soit une fois, soit pas du tout. Par exemple, '`ca?r`' correspond à '`car`' ou '`cr`' ; rien d'autre.

`*?, +?, ??`

sont les variantes non gourmandes des opérateurs précédents. Les opérateurs normaux '`*`', '`+`', '`?`' sont *gourmands* dans le fait qu'ils correspondent à des chaînes aussi longues que possible, aussi longtemps que la regexp entière peut continuer à correspondre. Avec un '`?`' à la suite, elles sont non gourmandes ; elles vont correspondre à des chaînes aussi courtes que possible.

Ainsi, ‘`ab*`’ et ‘`ab*?`’ peuvent toutes deux correspondre à la chaîne ‘`a`’ et à la chaîne ‘`abbbb`’ ; mais si vous essayez de faire correspondre les deux avec le texte ‘`abbb`’, ‘`ab*`’ va correspondre à la chaîne entière (la correspondance valide la plus longue possible), alors que ‘`ab*?`’ va correspondre à ‘`a`’ seulement (la correspondance la plus courte possible).

`\{`*n*`\}` est un opérateur suffixe qui spécifie une répétition *n* fois—ansi, l'expression rationnelle précédente dot correspondre exactement *n* fois à la suite. Par exemple, ‘`x\{4\}`’ correspond à la chaîne ‘`xxxx`’ et rien d'autre.

`\{`*n*`,`*m*`\}` est un opérateur suffixe qui spécifie une répétition entre *n* et *m* fois—ainsi, l'expression rationnelle précédente doit correspondre au moins *n* fois, mais pas plus de *m* fois. Si *m* est omis, il n'y a alors pas de limite supérieure, mais l'expression rationnelle précédente doit correspondre au moins *n* fois.
‘`\{0,1\}`’ est équivalent à ‘`?`’.
‘`\{0,\}`’ est équivalent à ‘`*`’.
‘`\{1,\}`’ est équivalent à ‘`+`’.

`[ ... ]` est un *jeu de caractères*, qui commence avec ‘`[`’ et termine par ‘`]`’. Dans sa forme la plus simple, les caractères entre les deux crochets sont ce à quoi ce jeu peut correspondre.

Ainsi, ‘`[ad]`’ correspond soit à un ‘`a`’ soit à un ‘`d`’, et ‘`[ad]*`’ correspond à une chaîne composée uniquement de ‘`a`’ et de ‘`d`’ (incluant la chaîne vide), de quoi il découle que ‘`c[ad]*r`’ correspond à ‘`cr`’, ‘`car`’, ‘`cdr`’, ‘`caddaar`’, etc.

Vous pouvez aussi inclure un rang de caractères dans un jeu de caractères, en écrivant les caractères de début et de fin séparés par un ‘`-`’. Ainsi, ‘`[a-z]`’ correspond à n'importe quelle lettre ASCII minuscule. Des rangs peuvent être mélangés librement avec des caractères individuels, comme dans ‘`[a-z$%.]`’, qui correspond à n'importe quelle lettre ASCII minuscule ou ‘`$`’, ‘`%`’ un point.

Notez que les caractères habituellement spéciaux pour une regexp ne le sont pas dans un jeu de caractères. Un jeu complètement différent de caractères spéciaux existe dans les jeux de caractères : ‘`]`’, ‘`-`’ et ‘`^`’.

Pour inclure un ‘`]`’ dans un jeu de caractères, il doit être le premier des caractères. Par exemple, ‘`[]a]`’ correspond à ‘`]`’ ou ‘`a`’. Pour inclure ‘`-`’, placez ‘`-`’ comme premier ou dernier caractère du jeu, ou placez-le après un raaang. Ainsi, ‘`[]-]`’ correspond à ‘`]`’ et ‘`-`’.

Pour inclure ‘`^`’ dans un jeu, placez-le n'importe où sauf au début du jeu.

Lorsque vous utilisez un rang dans une recherche non sensible à la casse, vous devez écrire les deux extrémités du rang en majuscule, ou les deux en minuscule, ou les deux doivent ne pas être des lettres. Le comportement d'un rang avec des casses mixées comme '`A-z`' est mal défini, et pourra être modifié dans des versions futures d'Emacs.

`[^ ... ]` '`[^`' commence un *jeu de caractères complémentaire*, qui correspond à tout caractère non spécifié dans le jeu. Ainsi, '`[^a-z0-9A-Z]`' correspond à tous les caractères *sauf* aux lettres et aux chiffres.

'`^`' n'est pas un caractère spécial dans un jeu de caractères à moins qu'il ne soit le premier caractère. Le caractère suivant le '`^`' est traité comme étant premier (en d'autres mots, '`-`' et '`]`' ne sont pas spéciaux s'ils suivent '`^`').

Un jeu de caractères complémentaire peut correspondre à un newline, sauf si newline est mentionné dans les caractères auxquels ne pas correspondre. C'est en contraste avec les regexps de programmes tels que `grep`.

`^` est un caractère spécial qui correspond à la chaîne vide, mais seulement en début d'une ligne dans le texte à faire correspondre. Autrement, il ne correspond à rien. Ainsi, '`^foo`' correspond à '`foo`' placé en début de ligne.

`$` est similaire à '`^`' mais correpond seulement en fin de ligne. Ainsi, '`x+$`' correspond à une chaîne d'un '`x`' ou plus en fin de ligne.

`\` a deux fonctions : il cite les caractères spéciaux ('\' inclus), et introduit des constructions spéciales supplémentaires.

Puisque '\' cite les caractères spéciaux, '`\$`' est une expression rationnelle qui correspond seulement à '`$`', et '`\[`' est une expression rationnelle qui correspond seulement à '`[`', et ainsi de suite.

Note : pour une compatibilité historique, les caractères spéciaux sont traités comme ordinaires s'ils sont dans un contexte où leur "caractère" spécial n'a pas de sens. Par exemple, '`*foo`' traite '`*`' comme ordinaire car il n'y a pas d'expression précédente sur laquelle '`*`' puisse agir. C'est une pratique pauvre que de dépendre de ce comportement ; il est préférable de toujours citer les caractères spéciaux, indépendamment de l'endroit où ils apparaissent.

Pour la plus grande part, '\' suivi d'un caractère correspond seulement à ce caractère. Il y a toutefois quelques exceptions : des séquences de deux caractères commençant par '\' qui une signification particulière. Le second caractère de la séquence est toujours un caractère ordinaire lorsqu'il est utilisé seul. Voici un tableau des constructions '\'.

`\|` spécifie une alternative. Deux expressions rationnelles *a* et *b* séparées par '`\|`' forment une expression qui correspond à un texte si *a* correspond à ce texte ou si *b* y correspond. il fonctionne en essayant de faire correspondre *a*, et en cas d'échec, essaie de faire correspondre *b*.

Ainsi, '`foo\|bar`' correspond soit à '`foo`' soit à '`bar`', et rien d'autre.

'`\|`' s'applique aux expressions qui l'entoure les plus longues possibles. Seul un regroupement '`\( ... \)`' peut limiter le pouvoir de regroupement de '`\|`'.

`\( ... \)` est une construction de regroupement qui a trois utilités :

1. pour entourer un jeu d'alternatives '`\|`' pour d'autres opérations. Ainsi, '`\(foo\|bar\)x`' correspond soit à '`foox`', soit à '`barx`'.
2. Pour entourer une expression complexe pour que les opérateurs suffixes '`*`', '`+`' et '`?`' agissent dessus. Ainsi, '`ba\(na\)*`' correspond à '`bananana`', etc., avec un certain nombre (zéro ou plus) de chaînes '`na`'.
3. Pour enregistrer une sous-chaîne correspondante pour une référence future.

Cette dernière application n'est pas une conséquence de l'idée de regroupement parenthésé ; c'est une caractéristique indépendante qui est assignée comme une seconde signification à la même construction '`\( ... \)`'. En pratique, il n'y a habituellement aucun conflit entre les deux significations ; lorsqu'il y a un conflit, vous pouvez utiliser un groupe "shy."

`\(?: ... \)`
spécifie un groupe "shy" qui n'enregistre pas la sous-chaîne correspondante ; vous ne pouvez pas vous y référer avec '`\`*d*'. Ceci est utile dans des expressions rationnelles combinées mécaniquement, pour que vous puissiez ajouter des groupes à usages syntaxiques sans interférer avec la numérotation des groupes écrits par l'utilisateur.

`\`*d* correspond au même texte correspondant avec la *d*ième occurence d'une construction '`\( ... \)`'.

Après la fin d'une construction '`\( ... \)`', la recherche de correspondance se souvient de la position de début et de fin du texte correspondant à cette construction. Alors, plus tard dans l'expression rationnelle, vous pouvez utiliser '`\`' suivi du chiffre *d* pour dire "correspond au même texte correspondant la *d*ième fois à la construction '`\( ... \)`'."

Les chaînes correspondant aux neuf premières constructions '`\( ... \)`' apparaissant dans une expression rationnelle se voient

assigner les chiffres 1 à 9 dans l'ordre où les parenthèses ouvrantes sont apparues dans l'expression rationnelle. Vous pouvez alors utiliser de '`\1`' à '`\9`' pour référencer le texte coorespondant à la construction '`\( ... \)`' associée.

Par exemple, '`\(.*\)\1`' correspond à toute chaîne sans newline composée de deux moitiés identiques. '`\(.*\)`' correspond à la première moitié, qui peut être n'importe quoi, mais le '`\1`' qui suit doit correspondre à exactement le même texte.

Si une construction '`\( ... \)`' particulière correspond plus d'une fois (ce qui peut facilement arriver si elle est suivie de '`*`'), seule la dernière correspondance est sauvegardée.

`\‘` correspond à la chaîne vide, mais seulement au début du tampon ou de la chaîne à faire correspondre.

`\’` correspond à la chaîne vide, mais seulement à la fin du tampon ou de la chaîne à faire correspondre.

`\=` correspond à la chaîne vide, mais seulement au point.

`\b` correspond à la chaîne vide, mais seulement au début ou à la fin d'un mot. Ainsi, '`\bfoo\b`' correspond à toute occurence de '`foo`' comme mot séparé. '`\bballs?\b`' correspond à '`ball`' ou '`balls`' comme mots.

'`\b`' correspond au début ou à la fin du tampon indépendamment du texte apparaissant près de lui.

`\B` correspond à la chaîne vide, mais *non* au début ou à la fin d'un mot.

`\<` correspond à la chaîne vide, mais seulement au début d'un mot. '`\<`' correspond au début du tampon seulement si un caractère constituant d'un mot le suit.

`\>` correspond à la chaîne vide, mais seulement à la fin d'un mot. '`\>`' correspond à la fin du tampon seulement si le contenu finit avec un caractère constituant d'un mot.

`\w` correspond à tout cractère constituant d'un mot. La table de syntaxe détermine quels caractères peuvent constituer un mot. See Section 31.6 [Syntax], page 485.

`\W` correspond à tout caractère non constituant d'un mot.

`\s`*c* correspond à tout caractère dont la syntaxe est *c*. Ici *c* est un caractère qui représente un code de syntaxe : '`w`' pour contituant de mot, '`-`' pour caractère blanc, '`(`' pour parenthèse ouvrante, etc. Représentez un caractère blanc (qui peut être un newline) par soit '`-`' soit un caractère espace.

`\S`*c* correspond à tout caractère dont la syntaxe n'est pas *c*.

`\c`*c* correspond à tout caractère appartenant à la catégorie *c*. Par exemple, '`\cc`' correspond aux caractères chinois, '`\cg`' correspond aux caractères grecs, etc. Pour une description des catégories connues, tapez `M-x describe-categories` ⟨RET⟩.

`\C`*c* correspond à tout caractère n'appartenant *pas* à la catégorie *c*.

Les constructions qui se rapportent à des mots et syntaxes sont contrôlées par la table de syntaxe (see Section 31.6 [Syntax], page 485).

Voici une expression rationnelle complexe, utilisée par Emacs pour reconnaître la fin d'une phrase, avec les espaces qui la suivent. Elle est donnée dans la syntaxe Lisp pour vous permettre de distinguer les espaces des caractères tab. Dans la syntaxe Lisp, la constante chaîne commence et finit avec des guillemets. '`\"`' spécifie un guillemet dans la regexp, '`\\`' une barre oblique inverse dans la regexp, '`t`' un caractère tab et '`\n`' un caractère newline.

```
"[.?!][]\"')]*\\($\\|\t\\|  \\)[ \t\n]*"
```

Elle contient quatre parties successives : un jeu de caractères correspondant à un point, '`?`' ou '`!`' ; un jeu de caractères correspondant à des crochets fermants, guillemets ou parenthèses, répétés un nombre quelconque de fois ; une alternative dans une construction de regroupement correspondant à une fin de ligne, un caractère tab ou deux espaces ; et un jeu de caractères correspondant à des caractères blancs, répétés un nombre quelconque de fois.

Pour entrer la même regexp interactivement, vous devrez taper ⟨TAB⟩ pour entrer un caractère tab, et `C-j` pour entrer un newline. Vous devrez aussi taper des barres obliques inverses simples, plutôt que de les doubler comme pour la syntaxe Lisp.

12.6 Recherche et Casse

Les recherches incrémentales d'Emacs ignorent normalement la casse du texte qu'elles recherchent, si vous spécifiez le texte en minuscule. Ainsi, lorsque vous spécifiez la recherche de '`foo`', alors '`Foo`' et '`foo`' sont tous deux considérés comme correspondants. Les regexps, et en particulier les jeux de caractères, sont inclus : '`[ab]`' peut correspondre à '`a`' ou '`A`', ou '`b`' ou '`B`'.

Une lettre majuscule n'importe où dans la chaîne de recherche incrémentale rend la recherche sensible à la casse. Ainsi, la recherche de '`Foo`' ne trouve ni '`foo`' ni '`FOO`'. Ceci s'applique aussi bien à la recherche d'expressions rationnelles qu'à la recherche de chaînes. L'effet cesse si vous supprimez la lettre majuscule de la chaîne de recherche.

Si vous mettez la variable `case-fold-search` à `nil`, alors toutes les lettres doivent correspondre exactement, casse comprise. C'est une variable locale au tampon ; altérer cette variable affecte seulement le tampon courant, mais elle a une valeur par défaut que vous pouvez aussi changer.

See Section 31.2.4 [Locals], page 466. Cette variable s'applique aussi aux recherches non incrémentales, incluant celles exécutées par les commandes de remplacement (see Section 12.7 [Replace], page 132) et les commandes de correspondance de l'historique du mini-tampon (see Section 5.4 [Minibuffer History], page 61).

12.7 Commandes de Remplacement

Les opérations de recherche et remplacement globales ne sont pas aussi utiles sous Emacs qu'elles le sont avec d'autres éditeurs[1], mais elles sont disponibles. En plus de la commande simple `M-x replace-string` identique à celle trouvée dans la plupart des éditeurs, il existe une commmande `M-x query-replace` qui vous demande, pour chaque occurence du motif recherché, s'il faut le remplacer.

Les commandes de remplacement opérent normalement sur le texte entre le point et la fin du tampon ; cependant, dans le mode de Marque Transitoire, lorsque la marque est active, elles opérent dans la région. Les commandes de remplacement remplacent toutes une chaîne (ou une regexp) avec une chaîne de remplacement. Il est possible d'exécuter plusieurs remplacements en parallèle en utilisant la commande `expand-region-abbrevs` (see Section 24.3 [Expanding Abbrevs], page 347).

12.7.1 Remplacement Inconditionnel

`M-x replace-string` ⟨RET⟩ *chaîne* ⟨RET⟩ *nouvelle chaîne* ⟨RET⟩
: Remplace toute occurence de *chaîne* avec *nouvelle chaîne*.

`M-x replace-regexp` ⟨RET⟩ *regexp* ⟨RET⟩ *nouvelle chaîne* ⟨RET⟩
: Remplace toute correspondance de *regexp* avec *nouvelle chaîne*.

Pour remplacer toute instance de '`foo`' après le point avec '`bar`', utilisez la commande `M-x replace-string` avec les deux arguments '`foo`' et '`bar`'. Le remplacement est effectué seulement sur le texte suivant le point, donc si vous voulez couvrir le tampon en entier, vous devrez d'abord placer le point au début. Toutes les occurences jusqu'à la fin du tampon sont remplacées ; pour limiter le remplacement à une partie du tampon, restreignez à cette partie du tampon avant de faire le remplacement (see Section 30.9 [Narrowing], page 443). Dans le mode de Marque Transitoire, lorsque la région est active, le remplacement est limité à la région (see Section 8.2 [Transient Mark], page 78).

[1] Dans certains éditeurs, les opérations de recherche et remplacement sont la seule manière de faire un changement unique dans le texte.

Lorsque `replace-string` se termine, elle laisse le point à la dernière accurence remplacée. Elle place la marque à la place précédente du point (où la commande `replace-string` a été lancée) ; utilisez `C-u C-`⟨SPC⟩ pour y retourner.

Un argument numérique restreint le remplacement aux correspondances entourées de frontières de mots. La valeur de l'argument n'importe pas.

12.7.2 Remplacement de Regexp

La commande `M-x replace-string` remplace des correspondances exactes d'une chaîne unique. La commande similaire `M-x replace-regexp` remplace toute correspondance d'un motif spécifié.

Dans `replace-regexp`, *nouvelle chaîne* n'est pas nécessairement constante : elle peut référer à tout ou partie du texte correspondant à la *regexp*. '`\&`' dans *nouvelle chaîne* indique le texte correspondant, en entier. '`\`*d*' dans *newstring*, où *d* est un chiffre, indique le texte correspondant au *d*ième groupe parenthésé dans *regexp*. Pour inclure un '`\`' dans le texte de remplaement, vous devez entrer '`\\`'. Par exemple,

`M-x replace-regexp` ⟨RET⟩ `c[ad]+r` ⟨RET⟩ `\&-safe` ⟨RET⟩

remplace (par exemple) '`cadr`' avec '`cadr-safe`' et '`cddr`' avec '`cddr-safe`'.

`M-x replace-regexp` ⟨RET⟩ `\(c[ad]+r\)-safe` ⟨RET⟩ `\1` ⟨RET⟩

exécute la transformation inverse.

12.7.3 Commandes de Remplacement et Casse

Si le premier argument d'une commande de remplacement est uniquement en minuscule, les commandes ignorent la casse lors de la recherche des occurences à remplacer—à condition que `case-fold-search` est non `nil`. Si `case-fold-search` est `nil`, la casse est toujours significative dans toutes les recherches.

De plus, lorsque l'argument *newstring* est entièrement ou en partie en minuscule, les commandes de remplacement essaient de respecter le motif de la casse de chaque occurence. Ainsi, la commande

`M-x replace-string` ⟨RET⟩ `foo` ⟨RET⟩ `bar` ⟨RET⟩

remplace '`foo`' en minuscule avec '`bar`' en minuscule, '`FOO`' tout en majuscule avec '`BAR`', et '`Foo`' majuscule avec '`Bar`'. (Ces trois alternatives—minuscule, tout en majuscule, et majuscule, sont les seules que `replace-string` peut distinguer.)

Si des lettres majuscules sont utilisées dans la chaîne de remplacement, elles restent en majuscule chaque fois que ce texte est inséré. Si des lettres majuscules sont utilisées dans le premier argument, le second argument

est toujours inséré exactement comme il est donné, sans conversion de la casse. De même, si soit `case-replace` soit `case-fold-search` est `nil`, le remplacement est effectué sans conversion de casse.

12.7.4 Remplacement Interrogatif

`M-%` *chaîne* ⟨RET⟩ *nouvelle chaîne* ⟨RET⟩
`M-x query-replace` ⟨RET⟩ *chaîne* ⟨RET⟩ *nouvelle chaîne* ⟨RET⟩
Remplace certaines occurences de *chaîne* avec *nouvelle chaîne*.

`C-M-%` *regexp* ⟨RET⟩ *newstring* ⟨RET⟩
`M-x query-replace-regexp` ⟨RET⟩ *regexp* ⟨RET⟩ *newstring* ⟨RET⟩
Remplace certaines correspondances de *regexp* avec *newstring*.

Si vous voulez changer seulement certaines occurences de '`foo`' en '`bar`', en non toutes, vous ne pouvez alors pas utiliser `replace-string`. À la place, utilisez `M-%` (`query-replace`). Cette commande trouve des occurences de '`foo`' une par une, affiche chaque occurence et vous demande s'il doit la remplacer. Un argument numérique à `query-replace` lui indique de considérer seulement les occurences délimitées par des caractères de frontières de mots. Elle préserve la casse, exactement comme `replace-string`, lorsque `case-replace` est non `nil`, ce qu'il est normalement.

À part interroger, `query-replace` fonctionne exactement comme `replace-string`, et `query-replace-regexp` exactement comme `replace-regexp`. Cette commande est lancée par `C-M-%`.

Lorsqu'il vous est montré une occurence de *chaîne* ou une correspondance de *regexp*, vous pouvez taper :

⟨SPC⟩ pour remplacer une occurence avec *nouvelle chaîne*.

⟨DEL⟩ pour sauter à l'occurence suivante sans remplacer celle-ci.

`,` (Comma)
pour remplacer cette occurence et afficher le résultat. Il vous est alors demandé d'entrer un autre caractère pour indiquer quoi faire ensuite. Le remplacement ayant déjà été fait, ⟨DEL⟩ et ⟨SPC⟩ sont équivalentes dans cette situation ; toutes deux vous déplace à l'occurence suivante.

Vous pouvez taper `C-r` à ce moment-là (voir plus loin) pour altérer le texte remplacé. Vous pouvez aussi taper `C-x u` pour annuler le remplacement ; cette commande termine `query-replace`, et si vous voulez faire d'autres remplacements, vous devez utiliser `C-x` ⟨ESC⟩ ⟨ESC⟩ ⟨RET⟩ pour redémarrer (see Section 5.5 [Repetition], page 63).

⟨RET⟩ pour terminer sans faire de remplacements supplémentaires.

`.` (Point) pour remplacer cette occurence puis terminer sans chercher d'occurences supplémentaires.

`!` pour remplacer toutes les occurences suivantes sans autre interrogation.

`^` pour retourner à la position de l'occurence précédente (ou ce qui était une occurence), au cas où vous l'ayez remplacée par erreur. Ceci marche en dépilant la pile des marques. Seulement un `^` à la suite est effectif, car une seule position précéednte est sauvegardée durant `query-replace`.

`C-r` pour entrer dans un niveau d'édition récursive, au cas où l'occurence doit être éditée pluôt que d'être simplement remplacée par *nouvelle chaîne*. Lorsque vous avez terminé, sortez du niveau d'édition récursive avec `C-M-c` pour rechercher l'occurence suivante. See Section 30.13 [Recursive Edit], page 447.

`C-w` pour supprimer l'occurence, puis entrer dans un niveau d'édition récursive comme avec `C-r`. Utilisez l'édition récursive pour insérer le texte remplaçant l'occurence supprimée de *chaîne*. Lorsque vous avez terminé, quittez le niveau d'édition récursive avec `C-M-c` pour rechercher l'occurence suivante.

`e` pour éditer la chaîne de remplacement dans le mini-tampon. Lorsque vous quittez le mini-tampon en tapant ⟨RET⟩, le contenu du mini-tampon remplace l'occurence en cours du motif. Il devient aussi la nouvelle chaîne de remplacement pour toute nouvelle occurence.

`C-l` pour réafficher l'écran. Vous devez alors taper un autre caractère pour spécifier quoi faire avec cette occurence.

`C-h` pour afficher un message résumant ces options. Vous devez alors taper un autre caractère pour spécifier quoi faire avec cette occurence.

Certains autres caractères sont des pseudonymes de caractères listés plus hauts : `y`, `n` et `q` sont équivalents à ⟨SPC⟩, ⟨DEL⟩ et ⟨RET⟩.

En dehors de ces caractères, tout autre caractère termine `query-replace`, puis est relu comme partie d'une séquence de touches. Ainsi, si vous tapez `C-k`, `query-replace` se termine et la fin de ligne est coupée.

Pour redémarrer `query-replace` une fois qu'il a été terminé, utilisez `C-x` ⟨ESC⟩ ⟨ESC⟩, qui répète `query-replace` car elle a utilisée le mini-tampon pour lire ses arguments. See Section 5.5 [Repetition], page 63.

Voir aussi Section 28.9 [Transforming File Names], page 395, pour des commandes pour renommer, copier ou lier des fichiers en remplaçant des correspondances de regexp dans leurs noms de fichiers.

12.8 Autres Commandes de Recherche en Boucle

Voici quelques autres commandes qui trouvent des correspondances pour une expression rationnelle. Elles ignorent toutes la casse dans la recherche de correspondances, si le motif ne contient pas de lettres majuscules et `case-fold-search` est non `nil`. À part `occur`, toutes opérent sur le texte entre le point et la fin du tampon, ou dans la région active dans le mode de Marque Transitoire.

`M-x occur` ⟨RET⟩ *regexp* ⟨RET⟩
: Affiche une liste montrant chaque ligne du tampon qui contient une correspondance pour *regexp*. Pour limiter la recherche à une partie du tampon, restreignez-le à cette partie (see Section 30.9 [Narrowing], page 443). Un argument numérique *n* spécifie que *n* lignes du contexte doivent être affichées avant et après les lignes correspondantes.

  Le tampon '`*Occur*`' contenant la sortie sert de menu pour trouver les occurences dans leur contexte original. Cliquez `Mouse-2` sur une occurence listée dans '`*Occur*`', ou placez-y le point et tapez ⟨RET⟩ ; ceci vous déplace dans le tampon où la recherche a eu lieu et place le point sur l'original de l'occurence choisie.

`M-x list-matching-lines`
: Synonyme de `M-x occur`.

`M-x how-many` ⟨RET⟩ *regexp* ⟨RET⟩
: Affiche le nombre de correspondances pour *regexp* existantes dans le tampon après le point. Dans le mode de Marque Transitoire, si la région est active, la commande opére plutôt sur la région.

`M-x flush-lines` ⟨RET⟩ *regexp* ⟨RET⟩
: Supprime toute ligne contenant une correspondance de *regexp*, opérant sur le texte après le point. Dans le mode de Marque Transitoire, si la région est active, la commande opére plutôt sur la région.

`M-x keep-lines` ⟨RET⟩ *regexp* ⟨RET⟩
: Supprime toute ligne ne contenant *pas* une correspondance pour *regexp*, opérant sur le texte après le point. Dans le mode de Marque Transitoire, si la région est active, la commande opére plutôt sur la région.

Vous pouvez aussi rechercher dans plusieurs fichiers sous le contrôle d'une table de marques (see Section 22.16.6 [Tags Search], page 309) ou à l'aide de la commande `A` de Dired (see Section 28.7 [Operating on Files], page 392), ou encore demander au programme `grep` de le faire (see Section 23.2 [Grep Searching], page 332).

13 Commandes pour la Correction d'Erreurs de Frappe

Nous décrivons dans ce chapitre les commandes spécialement utiles pour les fois où vous voyez une faute dans votre texte juste après l'avoir faite, ou lorsque vous changez d'avis en composant du texte à la volée.

La commande la plus fondamentale pour corriger une édition éronnée est la commande d'annulation, `C-x u` ou `C-_`. Cette commande annule une unique commande (habituellement), une partie de commande (dans le cas de `query-replace`), ou plusieurs insertions consécutives de caractères. Des répétitions consécutives de `C-_` ou `C-x u` annulent des changements de plus en plus anciens, dans la limite des informations d'annulation disponibles. See Section 4.4 [Undo], page 45, pour plus d'informations.

13.1 Couper vos Fautes

⟨DEL⟩ Supprime le dernier caractère (`delete-backward-char`).

`M-`⟨DEL⟩ Coupe le dernier mot (`backward-kill-word`).

`C-x` ⟨DEL⟩ Coupe le début de la phrase (`backward-kill-sentence`).

Le caractère ⟨DEL⟩ (`delete-backward-char`) est la commande de correction la plus importante. Elle supprime le caractère avant le point. Lorsque ⟨DEL⟩ suit une commande d'auto-insertion d'un caractère, vous pouvez voir ce caractère comme annulant cette commande. Toutefois, évitez de penser à ⟨DEL⟩ comme un moyen général d'annuler une commande !

Lorsque votre faute est étendue sur plus de quelques caractères, il peut être plus pratique d'utiliser `M-`⟨DEL⟩ ou `C-x` ⟨DEL⟩. `M-`⟨DEL⟩ coupe le texte depuis le début du dernier mot, et `C-x` ⟨DEL⟩ coupe le texte depuis le début de la dernière phrase. `C-x` ⟨DEL⟩ est particulièrement utile lorsque vous changez d'avis sur la tournure d'une phrase que vous êtes en train d'écrire. `M-`⟨DEL⟩ et `C-x` ⟨DEL⟩ sauvegardent le texte coupé pour que `C-y` et `M-y` puissent le recoller. See Section 9.2 [Yanking], page 89.

`M-`⟨DEL⟩ est souvent utile même si vous avez mal tapé juste quelques caractères, si vous savez que vous vous êtes trompé dans la frappe mais ne savez pas exactement comment. Dans ce cas, vous ne pouvez pas corriger avec ⟨DEL⟩ sans regarder l'écran pour voir ce que vous avez tapé. Il est plus vite fait de couper le dernier mot en entier et de recommencer.

13.2 Transposer du Texte

`C-t` Transpose deux caractères (`transpose-chars`).

`M-t` Transpose deux mots (`transpose-words`).

`C-M-t` Transpose deux expressions équilibrées (`transpose-sexps`).

`C-x C-t` Transpose deux lignes (`transpose-lines`).

L'erreur courante de transposer deux caractères peut être corrigée, lorsqu'ils sont adjacents, avec la commande `C-t` (`transpose-chars`). Normalement, `C-t` transpose les deux caractères qui se trouvent de chaque côté du point. Lorsqu'elle est utilisée en fin de ligne, plutôt que de transposer le dernier caractère avec le newline, ce qui ne serait pas vraiment utile, `C-t` transpose les deux derniers caractères de la ligne. Ainsi, si vous vous rendez compte de votre erreur de transposition assez vite, vous pouvz la corriger en tapant juste `C-t`. Si vous ne vous en rendez pas compte assez vite, vous devez revenir en arrière pour placer le point entre les deux caractères transposés. Si vous avez transposé un espace avec le dernier caractère du mot le précédant, les commandes de déplacement par mots est un bon moyen de s'y rendre. Autrement, une recherche en arrière `C-r` est souvent le moyen le plus pratique. See Chapter 12 [Search], page 119.

`M-t` (`transpose-words`) transpose le mot avant le point avec le mot après le point. Il avance le point d'un mot, amenant avec lui le mot précédant ou contenant le point. Les caractères de ponctuation entre les mots de bougent pas. Par exemple, '`FOO, BAR`' se transpose en '`BAR, FOO`' plutôt que '`BAR FOO,`'.

`C-M-t` (`transpose-sexps`) est une commande similaire pour transposer deux expressions (see Section 22.2 [Lists], page 274), et `C-x C-t` (`transpose-lines`) échange deux lignes. Elles fonctionnent comme `M-t` à l'exception de la manière dont elles divisent le texte en unités syntaxiques.

Un argument numérique à une commande de transposition sert de compte de répétition : il dit) la commande de transposition de déplacer le caractère (mot, sexp, ligne) avant ou contenant le point à travers plusieurs autres caractères (mots, sexps, lignes). Par exemple, `C-u 3 C-t` déplace le caractère avant le point à travers trois autres caractères. Cela changerait '`f*oobar`' en '`oobf*ar`'. C'est équivalent à répéter `C-t` trois fois. `C-u - 4 M-t` déplace le mot avant le point à travers quatre mots. `C-u - C-M-t` annulerait l'effet d'un `C-M-t`.

Un argument numérique de zéro a une signification particulière (car autrement une commande avec un compte de répétition de zéro ne ferait rien) : pour transposer le caractère (mot, sexp, ligne) finissant après le point avec celui finissant après la marque.

13.3 Conversion de Casse

`M-- M-l` Convertit le dernier mot en minuscule. Notez que `Meta--` est Meta-moins.

`M-- M-u` Convertit le dernier mot en majuscule.

`M-- M-c` Convertit le dernier mot en minuscule avec son initiale en capitale.

Une erreur très courante est de taper des mots dans la mauvaise casse. À cause de cela, les commandes de conversion de casse `M-l`, `M-u` et `M-c` ont un comportement particulier avec un argument négatif : elles ne déplacent pas le curseur. Aussitôt que vous avez vu que vous avez mal tapé le dernire mot, vous pouvez simplement convertir sa casse puis continuer à taper. See Section 21.6 [Case], page 254.

13.4 Vérifier et Corriger l'Orthographe

Cette section décrit les commandes pour vérifier l'orthographe d'un mot unique ou d'une portion du tampon. Ces commandes fonctionnent avec le programme de vérification d'orthographe Ispell, qui ne fait pas partie d'Emacs.

`M-x flyspell-mode`
Active le mode Flyspell, qui met en surbrillance les mots mal ortographiés.

`M-$` Vérifie et corrige l'orthographe du mot au point (`ispell-word`).

`M-`⟨TAB⟩ Complète le mot avant le point d'après le dictionnaire orthographique (`ispell-complete-word`).

`M-x ispell`
Vérifie l'ortographe de la région active ou du tampon courant.

`M-x ispell-buffer`
Vérifie et corrige l'orthographe de chaque mot du tampon.

`M-x ispell-region`
Vérifie et corrige l'orthographe de chaque mot de la région.

`M-x ispell-message`
Vérifie et corrige l'orthographe de chaque mot dans un courrier, excluant les passages cités.

`M-x ispell-change-dictionary` ⟨RET⟩ *dict* ⟨RET⟩
Redémarre le processus Ispell, en utilisant *dict* comme dictionnaire.

`M-x ispell-kill-ispell`
Termine le sous-processus Ispell.

Le mode Flyspell est un moyen complètement automatique de vérifier l'orthographe tout en éditant sous Emacs. Il opère en vérifiant les mots au moment où vous les modifiez ou les insérez. Lorsqu'il trouve un mot qu'il ne reconnaît pas, il met ce mot en surbrillance. Ceci n'interfère pas avec votre édition, mais lorsque vous voyez le mot en surbrillance, vous pouvez vous y déplacer et le corriger. Tapez `M-x flyspell-mode`pour activer ou désactiver ce mode dans le tampon courant.

Lorsque le mode Flyspell indique par surbrillance qu'un mot est mal orthographié, vous pouvez cliquer dessus avec `Mouse-2` pour afficher un menu des corrections possibles et des actions. Vous pouvez aussi corriger le mot en l'éditant manuellement de la manière que vous préférez.

Les autres fonctionnalités de correction orthographique d'Emacs vérifient les mots lorsque vous lancez une commande explicite pour cela. La vérification de tout ou partie du tampon est utile lorsque vous avez du texte qui a été écrit en dehors de cette session d'Emacs et qui peut contenir un certain nombre de fautes d'orthographe.

Pour vérifier l'orthographe du mot autour ou suivant le point, et optionnellement le corriger, utilisez la commande `M-$` (`ispell-word`). Si le mot n'est pas correct, la commande vous offre plusieurs alternatives.

Pour vérifier le tampon courant en entier, utilisez `M-x ispell-buffer`. Utilisez `M-x ispell-region` pour vérifier seulement la région courante. Pour vérifier l'orthographe d'un message électronique que vous êtes en train d'écrire, utilisez `M-x ispell-message` ; cette commande vérifie le tampon entier, sauf les passages indentés ou apparaissant comme citations d'autres messages.

La commande `M-x ispell` vérifie l'orthographe de la région active si le mode de Marque Transitoire est actif (see Section 8.2 [Transient Mark], page 78), autrement il vérifie l'orthographe dans le tampon courant.

Chaque fois que ces commandes rencontrent un mot incorrect, elles vous demandent quoi faire. Elles affichent une liste d'alternatives, incluant habituellement plusieurs mots se rapprochant de celui mal orthographié. Vous devez alors taper un caractère. Voici les réponses valides :

⟨SPC⟩
: Saute ce mot—continue à le considérer comme incorrect, mais ne le change pas ici.

`r` *nouveau* ⟨RET⟩
: Remplace ce mot (juste cette fois) avec *nouveau*.

`R` *nouveau* ⟨RET⟩
: Remplace ce mot avec *nouveau*, et fait un `query-replace` pour que vous puissiez le remplacer autre part dans le tampon si vous le désirez.

chiffre
: Remplace ce mot (juste cette fois) avec un des mots approchants affichés. Chaque mot approchant est listé avec un chiffre ; tapez ce chiffre pour le sélectionner.

`a` Accepte ce mot incorrect—le traite comme correct, mais seulement dans cette session d'édition.

`A` Accepte ce mot incorrect—le traite comme correct, mais seulement dans cette session d'édition et pour ce tampon.

`i` Insére ce mot dans votre fichier dictionnaire privé pour qu'Ispell le considère dorénavant comme correct, même dans de futures sessions.

`u` Insére la version en minuscule de ce mot dans votre fichier dictionnaire privé.

`m` Comme `i`, mais vous pouvez aussi spécifier des informations de complétion du dictionnaire.

`l` *mot* ⟨RET⟩
Cherche dans le dictionnaire des mots correspondants à *mot*. Ces mots deviennent la nouvelle liste de "mots approchants" ; vous pouvez sélectionner l'un d'eux pour remplacer celui incorrect en tapant un chiffre. Vous pouvez utiliser '`*`' dans *mot*.

`C-g` Quitte la vérification d'orthographe interactive. Vous pouvez le redémarrer plus tard avec `C-u M-$`.

`X` Identique à `C-g`.

`x` Quitte la vérification d'orthographe interactive et déplace le point à l'endroit où il se trouvait lorsque vous avez démarré la vérification.

`q` Quitte la vérification d'orthographe et termine le sous-processus Ispell.

`C-l` Rafraîchit l'écran.

`C-z` Cette touche garde son fonctionnement normal (suspend Emacs ou iconifie ce cadre).

La commande `ispell-complete-word`, qui est reliée à la touche `M-`⟨TAB⟩ dans les modes Texte et équivalents, montre une liste de complétions basée sur la correction d'orthographe. Insérez le début d'un mot, puis tapez `M-`⟨TAB⟩ ; la commande affiche dans une fenêtre une liste e complétions. Pour choisir une des complétions listées, cliquez dessus avec `Mouse-2`, ou déplacez dessus le curseur dans la fenêtre des complétions et tapez ⟨RET⟩. See Section 21.7 [Text Mode], page 255.

Une fois démarré, le sous-processus Ispell continue à s'exécuter (attendant quelque chose à faire), ainsi les commandes de vérification suivantes démarrent plus rapidement. Si vous voulez vous débarraser du processus Ispell, utilisez `M-x ispell-kill-ispell`. Ce n'est habituellement pas nécessaire, car le processus n'utilise de ressources que lorsque vous vérifiez l'orthographe.

Ispell utilise deux dictionnaires : le dictionnaire standard et votre dictionnaire privé. La variable `ispell-dictionary` spécifie le nom de fichier du dictionnaire standard à utiliser. Une valeur `nil` indique d'utiliser le dictionnaire par défaut. La commande `M-x ispell-change-dictionary` change le variable et redémarre le sous-processus Ispell, pour qu'il utilise un dictionnaire différent.

14 Gestion des Fichiers

Le système d'exploitation stocke de manière permanente les données dans des *fichiers*. Ainsi la plupart du texte que vous éditez provient d'un fichier et est à la fin enregistré dans un fichier.

Pour éditer un fichier, vous devez indiquer à Emacs de lire un fichier et de préparer un tampon contenant une copie du texte du fichier. Ceci est appelé *visiter* le fichier. Les commandes d'édition s'appliquent directement au texte du tampon ; c'est-à-dire à la copie dans Emacs. Les changements apparaissent dans le fichier lui-même seulement lorsque vous *sauvegardez* le tampon dans un fichier.

En plus de visiter et de sauvegarder des fichiers, Emacs peut supprimer, copier, renommer, et concaténer des fichiers, garder plusieurs versions de ceux-ci, et opérer sur des répertoires.

14.1 Noms de Fichiers

La plupart des commandes Emacs qui opérent sur un fichier vous demandent de spécifier un nom de fichier. (La sauvegarde et le retour sont des exceptions ; le tampon connaît le nom de fichier à utiliser pour celles-ci.) Vous entrez le nom de fichier en utilisant le mini-tampon (see Chapter 5 [Minibuffer], page 55). La *complétion* est disponible, pour rendre plus facile la saisie de longs noms de fichiers. See Section 5.3 [Completion], page 57.

Pour la plupart des opérations, un *nom de fichier par défaut* existe qui peut être utilisé si vous tapez seulement ⟨RET⟩ pour entrer un argument vide. Normalement, le nom de fichier par défaut est le nom du fichier visité dans le tampon courant ; ceci rend plus facile d'opérer avec ce fichier avec toute commande de fichiers d'Emacs.

Chaque tampon a un répertoire par défaut, normalement celui contenant le fichier visité dans ce tampon. Lorsque vous entrez un nom de fichier dans répertoire, le répertoire par défaut est utilisé. Si vous spécifiez un répertoire relatif, avec un nom ne commençant pas par une barre oblique, il est interprété en respect avec le répertoire par défaut. Le répertoire par défaut est gardé dans la variable `default-directory`, qui a une valeur différente dans chaque tampon.

Par exemple, si le nom de fichier par défaut est '`/u/rms/gnu/gnu.tasks`' alors le répertoire par défaut est '`/u/rms/gnu/`'. Si vous tapez seulement '`foo`', qui ne spécifie pas de répertoire, c'est un raccourci pour '`/u/rms/gnu/foo`'. '`../.login`' correspond à '`/u/rms/.login`'. '`new/foo`' correspond au nom de fichier '`/u/rms/gnu/new/foo`'.

La commande `M-x pwd` affiche le répertoire par défaut du tampon courant, et la commande `M-x cd` change celui-ci (à une valeur lue en utilisant le mini-tampon). Le réperoire par défaut d'un tampon change seulement lorsque la

commande `cd` est utilisée. Le répertoire par défaut d'un tampon visitant un fichier est initialisé avec le réperoire contenant le fichier visité par ce tampon. Si vous créez un tampon avec `C-x b`, son répertoire par défaut est copié sur celui du tampon qui était courant à ce moment.

Le répertoire par défaut apparaît dans le mini-tampon lorsque ce dernier devient actif pour lire un nom de fichier. Ceci permet deux choses : il vous *montre* quel est le répertoire par défaut, pour que vous puissiez taper un nom de fichier relatif tout en sachant exactement ce qu'il représente, et il vous permet d'*éditer* le répertoire par défaut pour en spécifier une valeur différente. Cette insertion du répertoire par défaut est annulée si vous mettez la variable `insert-default-directory` à `nil`.

Notez qu'il est légitime de taper un nom de fichier absolu après être entré dans le mini-tampon, ignorant la présence du nom du répertoire par défaut. Le mini-tampon final peut paraître invalide, mais ce n'est pas le cas. Par exemple, si le mini-tampon commence avec '`/usr/tmp/`' et que vous ajoutez '`/x1/rms/foo`', vous obtenez '`/usr/tmp//x1/rms/foo`' ; mais Emacs ignore tout depuis la première barre oblique jusqu'aux doubles barres obliques ; le résultat est '`/x1/rms/foo`'. See Section 5.1 [Minibuffer File], page 56.

'`$`' dans un nom de fichier est utilisé pour substituer des variables d'environnement. Par exemple, si vous avez utilisé la commande shell `export FOO=rms/hacks` pour définir une variable d'environnement `FOO`, vous pouvez alors utiliser '`/u/$FOO/test.c`' ou '`/u/${FOO}/test.c`' comme abréviation pour '`/u/rms/hacks/test.c`'. Le nom de la variable d'environnement est constitué de tous les caractères alphanumériques suivant '`$`' ; alternativement, il peut être entre accolades après '`$`'. Notez que les commandes shell définissant des variables d'environnement affectent Emacs seulement si elles sont utilisées avant le démarrage d'Emacs.

Vous pouvez utiliser '`~/`' dans un nom de fichier pour représenter votre répertoire personnel, ou '`~`*user-id*`/`' pour représenter le répertoire personnel de l'utilisateur dont le login est `user-id`. (Sur les systèmes DOS et Windows, où les utilisateurs n'ont pas de répertoire personnel, Emacs substitue '`~/`' avec la valeur de la variable d'environnement `HOME`; voir Section B.5.1 [General Variables], page 511.)

Pour accéder à un fichier avec '`$`' dans son nom, tapez '`$$`'. Cette paire est convertie en un unique '`$`' en même temps que la substitution de variable est faite pour de simples '`$`'. Alternativement, quotez le nom de fichier entier avec '`/:`' (see Section 14.13 [Quoted File Names], page 183). Les noms de fichiers commençant avec un '`~`' litéral doivent aussi être quotés avec '`/:`'.

La fonction Lisp qui effectue les substitutions est appelée `substitute-in-file-name`. La substitution est effectuée seulement sur les noms de fichiers lus à partir du mini-tampon.

Vous pouvez inclure des caractères non ASCII dans les noms de fichier si vous mettez la variable `file-name-coding-system` à une valeur non `nil`. See Section 18.8 [Specify Coding], page 227.

14.2 Visiter des Fichiers

`C-x C-f` Visite un fichier (`find-file`).

`C-x C-r` Visite un fichier pour le visualiser, sans permettre de changements à celui-ci (`find-file-read-only`).

`C-x C-v` Visite un fichier différent plutôt que celui visité en dernier (`find-alternate-file`).

`C-x 4 f` Visite un fichier, dans une autre fenêtre (`find-file-other-window`). N'altère pas ce qui est affiché dans la fenêtre sélectionnée.

`C-x 5 f` Visite un fichier, dans un nouveau cadre (`find-file-other-frame`). N'altère pas ce qui est affiché dans le cadre sélectionné.

`M-x find-file-literally`
Visite un fichier sans convertir son contenu.

Visiter un fichier veut dire copier son contenu dans un tampon d'Emacs pour que vous puissiez l'éditer. Emacs crée un nouveau tampon pour chaque fichier que vous visitez. Nous disons que ce tampon visite le fichier pour lequel il a été créé. Emacs construit le nom du tampon à partir du nom du fichier en enlevant le répertoire, et gardant juste le nom de base. Par exemple, un fichier appelé '`/usr/rms/emacs.tex`' aurait un tampon appelé '`emacs.tex`'. S'il existe déjà un tampon avec ce nom, un nom unique est construit en ajoutant '`<2>`', '`<3>`', etc, en utilisant le nombre le plus petit possible.

La ligne de mode de chaque fenêtre indique le nom du tampon affiché dans cette fenêtre, de façon que vous connaissiez toujours le tampon que vous éditez.

Les changements qaue vous effectuez avec des commandes d'édition sont faites sur le tampon d'Emacs. Elles ne prennent pas effet sur le fichier que vous avez visité, ou sur tout autre endroit permanent, avant d'*enregistrer* le tampon. Enregistrer le tampon veut dire qu'Emacs écrit le contenu courant du tampon dans le fichier qu'il visite. See Section 14.3 [Saving], page 148.

Si un tampon contient des changements qui n'ont pas été enregistrés, nous disons que le tampon est *modifié*. C'est important car cela implique que certaines modifications seront perdues si le tampon n'est pas enregistré. La ligne de mode affiche deux étoiles près de la marge gauche pour indiquer que le tampon est modifié.

Pour visiter un fichier, utilisez la commande `C-x C-f` (`find-file`). Faites suivre la commande par le nom du fichier que vous voulez visiter terminé par ⟨RET⟩.

Le nom du fichier est lu en utilisant le mini-tampon (see Chapter 5 [Minibuffer], page 55), la valeur par défaut et la complétion fonctionnant

de manière habituelle (see Section 14.1 [File Names], page 143). Depuis le mini-tampon, vous pouvez annuler `C-x C-f` en tapant `C-g`.

Lorsqu'Emacs est construit avec un toolkit adéquat, il ouvre le dialogue de Sélection de Fichier de ce toolkit plutôt que de demander le nom de fichier dans le mini-tampon. Sur les plateformes Unix et GNU/Linux, Emacs fait cela lorsqu'il est construit avec les toolkits LessTif et Motif ; sur MS-Windows, la version graphique le fait par défaut.

La confirmation que `C-x C-f` a fonctionné est l'apparition d'un nouveau texte sur l'écran et un nouveau nom de tampon dans la ligne de mode. Si le fichier spécifié n'existe pas et ne peut pas être créé, ou ne peut pas être lu, vous obtenez alors une erreur, avec un message d'erreur affiché dans la zone de répercussion.

Si vous visitez un fichier qui est déjà dans Emacs, `C-x C-f` n'en fait pas une nouvelle copie. Il sélectionne le tampon existant contenant ce fichier. Cependant, avant de faire cela, il vérifie que le fichier n'a pas changé depuis la dernière fois que vous l'avez visité ou enregistré. Si le fichier a changé, un message d'avertissement est affiché. See Section 14.3.2 [Simultaneous Editing], page 154.

Qu'en est-il si vous voulez créer un fichier ? Visitez-le, cest tout. Emacs affiche '`(New File)`' dans la zone de répercussion, mais autrement se comporte exactement comme si vous visitiez un fichier vide existant. Si vous faites des changements et les enregistrez, le fichier est créé.

Emacs reconnaît à partir du contenu d'un fichier quelle convention il utilise pour séparer les lignes—newline (utilisé sous GNU/Linux et sous Unix), carriage-return linefeed (utilisé sous les systèmes Microsoft), ou seulement carriage-return (utilisé sous Macintosh)—et convertit automatiquement le contenu dans la convention Emacs normale, qui est de séparer les lignes avec le caractère newline. C'est une partie de la fonctionnalité plus générale de conversion de système de codage (see Section 18.6 [Coding Systems], page 223), et rend possible l'édition de fichiers importés de différents systèmes d'exploitation avec facilité égale. Si vous changez le texte et enregistrez le fichier, Emacs exécute la conversion inverse, en changeant les caractères newline en carriage-return linefeed ou carriage-return seulement si approprié.

Si le fichier que vous spécifiez se trouve être un répertoire, `C-x C-f` appelle Dired, le navigateur de répetoires d'Emacs, pour que vous puissiez "éditer" le contenu du répertoire (see Chapter 28 [Dired], page 387). Dired permet de supprimer, regarder, ou opérer facilement sur les fichiers du répertoire. Cependant, si la variable `find-file-run-dired` est `nil`, une erreur survient lorsque vous essayez de visiter un répertoire.

Les fichiers qui sont des collections d'autres fichiers, ou *fichiers archive*, sont visités dans des modes spéciaux qui invoquent un environnement du style Dired pour permettre des opérations sur les membres de l'archive. See ⟨undefined⟩ [File Archives], page ⟨undefined⟩, pour plus d'informations sur ces fonctionnalités.

Si le nom de fichier spécifié contient des caractères générique de sytle shell, Emacs visite tous les fichiers correspondants. Les caractères génériques comprennent '`?`', '`*`' et les séquences '`[...]`'. See Section 14.13 [Quoted File Names], page 183, pour savoir comment visiter un fichier dont le nom contient des caractères génériques. Vous pouvez désactiver la fonctionnalité des caractères génériques en personnalisant `find-file-wildcards`.

Si vous visitez un fichier que le système d'exploitation ne vous laisse pas modifier, Emacs rend le tampon en lecture seule, pour que vous ne puissiez pas faire de changements que vous ne pourriez pas enregistrer plus tard. Vous pouvez rendre le tampon modifiable avec `C-x C-q` (`vc-toggle-read-only`). See Section 15.3 [Misc Buffer], page 187.

À l'occasion, vous pouvez vouloir visiter un fichier en lecture seule pour vous empêcher de le modifier accidentellement ; faites-le en visitant le fichier avec la commande `C-x C-r` (`find-file-read-only`).

Si vous visitez par erreur un fichier non existant (en tapant un mauvais nom de fichier), utilisez la commande `C-x C-v` (`find-alternate-file`) pour visiter le fichier que vous vouliez vraiment visiter. `C-x C-v` est similaire à `C-x C-f`, mais détruit le tampon courant (après vous avoir proposé de l'enregistrer si vous l'avez modifié). Lorsque cette commande lit le nom du fichier à visiter, elle insére entièrement le nom de fichier par défaut dans le tampon, le point se trouvant juste après la partie répertoire ; ceci est pratique si vous avez fait une erreur de frappe en tapant le nom.

Si vous trouvez un fichier qui existe mais qui ne peut être lu, `C-x C-f` signale une erreur.

`C-x 4 f` (`find-file-other-window`) est similaire à `C-x C-f` à l'exception que le tampon contenant le fichier spécifié est sélectionné dns une autre fenêtre. La fenêtre qui était sélectionnée avant `C-x 4 f` continue à montrer le même tampon qu'il montrait auparavant. Si cette commande est utilisée lorsqu'une seule fenêtre est affichée, cette fenêtre est coupée en deux, une fenêtre montrant le même tampon que précédemment, et l'autre fenêtre montrant le fichier demandé. See Chapter 16 [Windows], page 195.

`C-x 5 f` (`find-file-other-frame`) est similaire, mais ouvre un nouveau cadre, ou rend visible le cadre existant qui contient le fichier demandé. Cette caractéristique est disponible seulement si vous utilisez un système de fenêtrage. See Chapter 17 [Frames], page 203.

Si vous désirez éditer un fichier sous forme d'une séquence de caractères ASCII sans encodage spécial ou conversion, utilisez la commande `M-x find-file-literally`. Elle visite un fichier, comme `C-x C-f`, mais n'effectue pas de conversion de format (see Section 21.11 [Formatted Text], page 265), de conversion de code de caractère (see Section 18.6 [Coding Systems], page 223), ou de décompression automatique (see Section 14.11 [Compressed Files], page 182), et n'ajoute pas de caractère newline final dû à `require-final-newline`. Si vous avez déjà visité le même fichier de

la manière habituelle (non litérale), cette commande vous demande si vous désirez plutôt le visiter litéralement.

Deux variables crochet spéciales permettent aux extensions de modifier l'opération de visiter des fichiers. Visiter un fichier qui n'existe pas appelle les fonctions de la liste `find-file-not-found-hooks` ; cette variable contient une liste de fonctions, et les fonctions sont appelées une par une (sans arguments) jusqu'au moment où une d'elle retourne une valeur non `nil`. Ce n'est pas une fonction crochet habituelle, et le nom se termine par '`-hooks`' plutôt que '`-hook`' pour indiquer ce fait.

Chaque visite d'un fichier, existant ou non, attend que `find-file-hooks` contienne une liste de fonctions et les appelle toutes, une par une, sans arguments. Cette variable est une vraie fonction crochet, mais a un nom anormal pour une comptabilité historique. Dans le cas d'un fichier non existant, les crochets `find-file-not-found-hooks` sont exécutées en premier. See Section 31.2.3 [Hooks], page 465.

Il y a plusieurs moyens de spécifier automatiquement le mode majeur à utiliser pour éditer un fichier (see Section 19.1 [Choosing Modes], page 236), et pour spécifier les variables locales définies pour ce fichier (see Section 31.2.5 [File Variables], page 468).

14.3 Enregistrer des Fichiers

Enregistrer un fichier à partir d'Emacs veut dire écrire dans le fichier visité par un tampon le contenu de ce tampon.

`C-x C-s` Enregistre le tampon courant dans le fichier qu'il visite (`save-buffer`).

`C-x s` Enregistre certains ou tous les tampons dans les fichiers qu'ils visitent (`save-some-buffers`).

`M-~` Oublie que le tampon courant a été modifié (`not-modified`). Avec un argument préfixe (`C-u`), marque le tampon courant comme modifié.

`C-x C-w` Enregistre le tampon courant dans un fichier spécifique (`write-file`).

`M-x set-visited-file-name`
Change le nom de fichier sou lequel le tampon courant sera enregistré.

Lorsque vous désirez enregistrer un fichier et rendre vos changements permanents, tapez `C-x C-s` (`save-buffer`). Une fois que l'enregistrement est terminé, `C-x C-s` affiche un message comme :

```
Wrote /u/rms/gnu/gnu.tasks
```

Si le tampon sélectionné n'est pas modifié (aucun changement n'y a été fait depuis que le tampon a été créé ou depuis sa dernière sauvegarde), l'enregistrement n'est pas réellement effectué, car il n'aurait aucun effet. À la place, `C-x C-s` affiche un message de ce style dans la zone de répercussion :

```
(No changes need to be saved)
```

La commande `C-x s` (`save-some-buffers`) offre la possibilité d'enregistrer certains ou tous les tampons modifiés. Il vous demande que faire avec chacun des tampons. Les réponses possibles sont analogues à celles de `query-replace` :

`y` Enregistre ce tampon et demande pour les tampons suivants.

`n` N'enregistre pas ce tampon, mais demande pour les tampons suivants.

`!` Enregistre ce tampon et tous les suivants sans plus de questions.

⟨RET⟩ Termine `save-some-buffers` sans enregistrement supplémentaire.

`.` Enregistre ce tampon, puis termine `save-some-buffers` sans poser de questions pour les tampons suivants.

`C-r` Affiche le tampon pour lequel on vous questionne. Lorsque vous quittez le mode de Visualisation, vous retournez à `save-some-buffers`, qui vous repose la question.

`C-h` Affiche un message d'aide sur ces options.

`C-x C-c`, la séquence de touches pour sortir d'Emacs, invoque `save-some-buffers` donc pose les même questions.

Si vous avez modifié un tampon mais ne désirez pas en enregistrer les changements, vous devrez prendre quelques précautions pour vous en prévenir. Autrement, chaque fois que vous utilisez `C-x s` ou `C-x C-c`, vous risquez d'enregistrer ce tampon par erreur. Une chose que vous pouvez faire est de taper `M-~` (`not-modified`), qui supprime l'indication que le tampon est modifié. Si vous faites cela, aucune des commandes d'enregistrement ne pensera que ce tampon a besoin d'être enregistré. ('~' est aussi utilisé comme symbole mathématique pour 'non' ; ainsi, `M-~` est meta-'non'.) Vous pouvez aussi utiliser `set-visited-file-name` (vos plus bas) pour marquer le tampon come visitant un fichier différent, un qui n'est pas utilisé pour quelque chose d'important. Alternativement, vous pouvez annuler tous les changements faits depuis que le fichier a été visité ou enregistré, en lisant de nouveau le texte à partir du fichier. Ceci est appelé *faire revenir*. See Section 14.4 [Reverting], page 157. Vous pouvez aussi annuler tous les changements en répétant le commande d'annulation `C-x u` jusqu'à que vous ayez annulé tous les changements ; mais faire revenir est plus simple.

`M-x set-visited-file-name` altère le nom du fichier que le tampon courant est en train de visiter. Elle lit le nouveau nom de fichier en utilisant

le mini-tampon. Elle spécifie alors le nom du fichier visité et change le nom du tampon (dans le cas où le nouveau nom n'est pas déjà en usage). `set-visited-file-name` n'enregistre pas le tampon dans le fichier nouvellement visité ; mais Emacs le retient au cas où vous enregistriez plus tard. Cette commande marque aussi le tampon comme "modifié" pour que `C-x C-s` dans ce tampon l'enregistre.

Si vous désirez marquer le tampon comme visitant un fichier différent et l'enregistrer en même temps, utilisez `C-x C-w` (`write-file`). C'est précisément équivalent à `set-visited-file-name` suivi de `C-x C-s`. `C-x C-s` utilisée dans un tampon ne visitant pas un fichier a le même effet que `C-x C-w` ; elle lit un nom de fichier, marque le tampon comme visitant ce fichier, et l'enregistre dans celui-ci. Le nom de fichier par défaut dans un tampon ne visitant pas un fichier est créé en combinant le nom du tampon avec le répertoire par défaut du tampon.

Si le nouveau nom de fichier appelle un mode majeur, alors `C-x C-w` lance ce mode majeur, dans la plupart des cas. La commande `set-visited-file-name` agit de même. See Section 19.1 [Choosing Modes], page 236.

Si Emacs est sur le point d'enregistrer un fichier et remarque que la date de la dernière version sur disque ne correspond pas à la date de dernière lecture ou écriture d'Emacs, il vous notifie de ce fait, car cela indique certainement un problème causé par des éditions simultanées et demande votre attention immédiate. See Section 14.3.2 [Simultaneous Editing], page 154.

Si la valeur de la variable `require-final-newline` est `t`, Emacs place silencieusement un caractère newline à la fin des fichiers qui ne finissent pas avec un tel caractère, chaque fois qu'un fichier est enregistré ou écrit. Si la valeur est `nil`, Emacs laisse inchangée la fin du fichier ; si ce n'est ni `nil` ni `t`, Emacs demande s'il doit ajouter un caractère newline. La valeur par dééfaut est `nil`.

14.3.1 Fichiers Archives

Sur la plupart des systèmes d'exploitation, réécrire un fichier écrase automatiquement l'ancien contenu du fichier. Ainsi, enregistrer un fichier depuis Emacs écrase l'ancien contenu du fichier—ou presque, à l'exception qu'Emacs copie l'ancien contenu dans un autre fichier, appelé le fichier *archive*, avant d'enregistrer réellement.

Pour la plupart des fichiers, la variable `make-backup-files` détermine s'il faut créer des fichiers archives. Sur la plupart des systèmes d'exploitation, sa valeur par défaut est `t`, et Emacs crée donc des fichiers archive.

Pour les fichiers gérés par un système de contrôle de version (see Section 14.7 [Version Control], page 161), la variable `vc-make-backup-files` détermine s'il faut créer des fichiers archive. Par défaut, sa valeur est `nil`, les fichiers archives étant redondants lorsque vous enregistrez toutes les ver-

sions précédentes dans un système de contrôle de version. See ⟨undefined⟩ [General VC Options], page ⟨undefined⟩.

La valeur par défaut de la variable `backup-enable-predicate` empêche la création de fichiers archive pour les fichiers des répertoires utilisés pour les fichiers temporaires, spécifiés par `temporary-file-directory` ou `small-temporary-file-directory`.

Selon votre choix, Emacs peut garder soit un seul fichier archive soit une série de fichiers archives numérotés pour chaque fichier que vous éditez.

Emacs crée une archive pour un fichier seulement la première fois que le fichier est enregistré depuis un tampon. Peu importe le nombre de fois que vous enregistrez le fichier, son fichier archive reste toujours identique à ce qu'il était avant que le fichier soit visité. Cela veut dire que le fichier archive contient normalement le contenu du fichier avant la session d'édition ; cependant, si vous détruisez le tampon puis visitez le fichier de nouveau, un nouveau fichier archive sera créé à l'enregistrement suivant.

Vous pouvez aussi demander explicitement qu'un autre fichier archive soit créé à partir d'un tampon même s'il a déjà été enregistré au moins une fois. Si vous enregistrez le tampon avec `C-u C-x C-s`, la version ainsi enregistrée sera mise dans un fichier archive si vous enregistrez de nouveau le tampon. `C-u C-u C-x C-s` enregistre le tampon, mais copie d'abord l'ancien contenu du fichier dans un nouveau fichier archive. `C-u C-u C-u C-x C-s` fait ces deux choses : il crée une archive à partir de l'ancien contenu, et s'arrange pour en créer une autre à partir du contenu ainsi enregistré, si vous enregistrez de nouveau.

14.3.1.1 Archives Simples ou Numérotées

Si vous choisissez d'avoir un fichier archive unique (par défaut), le nom du fichier archive est normalement construit en ajoutant '`~`' au nom du fichier étant édité ; ainsi, le fichier archive pour '`eval.c`' sera '`eval.c~`'.

Vous pouvez changer cette fonctionnalité en définissant la variable `make-backup-file-name-function` comme une fonction adéquate. Alternativement, vous pouvez personnaliser la variable *backup-directory-alist* pour indiquer que les fichiers coorespondant à certains motifs doivent être archivés dans des répertoires spécifiques.

Une utilisation typique est d'ajouter un élément `("." . dir)` pour créer toutes les archives dans le répertoire dont le nom absolu est *dir* ; Emacs modifie les noms de fichiers archive pour empêcher les collisions entre les fichiers de même nom provenant de répertoires différents. Alternativement, ajouter, disons, `("." ".~")` créerait les archives dans le sous-répertoire invisible '`.~`' du répertoire du fichier original. Emacs crée le répertoire, si nécessaire, pour créer l'archive.

Si le contrôle d'accès empêche Emacs d'écrire un fichier archive sous son noms habituel, il écrit le fichier archive en '`%backup%~`' dans votre répertoire personnel. Un seul fichier de ce genre peut exister, et seul le plus récent est disponible.

Si vous choisissez d'avoir une série de fichiers archives numérotés, les noms de ces fichiers contiennent '`.~`', le nombre, et un autre '`~`' après le nom de fichier original. Ainsi, les fichiers archive de '`eval.c`' seraient appelés '`eval.c.~1~`', '`eval.c.~2~`', et ainsi de suite jusqu'à des noms comme '`eval.c.~259~`' et plus. La variable `backup-directory-alist` s'applique aux archives numérotées comme aux autres.

Le choix d'archives uniques ou numérotées est contrôlé par la variable `version-control`. Ses valeurs possibles sont :

`t` Crée des archives numérotées.

`nil` Crée des archives numérotées pour les fichiers qui ont déjà des archives numérotées. Autrement, crée des archives uniques.

`never` Ne crée jamais d'archives numérotées ; crée toujours des archives uniques.

Vous pouvez définir `version-control` localement dans un tampon individuel pour contrôler la création des archives pour le fichier de ce tampon. Par exemple, le mode Rmail définit localement `version-control` à `never` pour être sûr qu'il n'y ait qu'une archive pour un fichier Rmail. See Section 31.2.4 [Locals], page 466.

Si vous définissez la variable d'environnement `VERSION_CONTROL`, pour dire à divers utilitaires GNU que faire avec les fichiers archives, Emacs se base aussi sur la variable d'environnement en définissant la variable Lisp `version-control` correctement au démarrage. Si la valeur de la variable d'environnement est '`t`' ou '`numbered`', alors `version-control` devient `t` ; si la valeur est '`nil`' ou '`existing`', alors `version-control` devient `nil`; si c'est '`never`' ou '`simple`', alors `version-control` devient `never`.

14.3.1.2 Suppression Automatique des Archives

Pour prévenir une consommation illimitée de l'espace disque, Emacs peut supprimer des versions d'archives numérotées automatiquement. Habituellement, Emacs garde les archives les plus récentes et les archives les plus anciennes, supprimant celles entre. Ceci arrive chaque fois qu'une nouvelle archive est créée.

Les deux variables `kept-old-versions` et `kept-new-versions` contrôlent cette suppression. Leurs valeurs sont, respectivement, le nombre d'archives les plus anciennes (de nombre le plus petit) à garder et le nombre d'archives les plus récentes (de nombre le plus grand) à garder, chaque fois qu'une nouvelle archive est créée. Souvenez-vous que ces valeurs sont utilisées juste

après qu'une nouvelle version d'archive est créée ; cette archive nouvellement créée est inclue dans le compte dans `kept-new-versions`. Par défaut, ces deux variables sont 2.

Si `delete-old-versions` est non `nil`, les versions excessives du milieu sont supprimées asns un murmure. Si elle est à `nil`, valeur par défaut, Emacs vous demande alors si les versions excessives du milieu doivent réellement être supprimées.

La commande Dired . (Point) peut aussi être utilisée pour supprimer d'anciennes versions. See Section 28.3 [Dired Deletion], page 388.

14.3.1.3 Copier ou Renommer

Les fichiers archives peuvent être créés en copiant l'ancien fichier ou en le renommant. Ceci peut faire une différence lorsque l'ancien fichier a plusieurs noms. Si l'ancien fichier est renommé en fichier archive, alors les noms alternatifs deviennent les noms des fichiers archives. Si, plutôt, l'ancien fichier est copié, alors les noms alternatifs restent les noms pour les fichiers que vous éditez, et le contenu accédé par ces noms sera le nouveau contenu.

La méthode pour créer les fichiers archives peut aussi affecter le propriétaire et le groupe du fichier. Si la copie est utilisée, il n'y a pas de changement. Si le renommage est utilisé, vous devenez le propriétaire du fichier, et le groupe du fichier devient le groupe par défaut (différents systèmes d'exploitation ont différentes valeurs par défaut pour le groupe).

Changer le propriétaire est habituellement une bonne idée, car alors le propriétaire indique toujours qui a édité le fichier en dernier. De plus, les propriétaires des archives montrent qui a produit ces versions. Occasionnellement, il peut y avoir un fichier dont le propriétaire ne doit pas changer ; une bonne idée pour ces fichiers est qu'ils contiennent des listes de variables locales pour définir `backup-by-copying-when-mismatch` localement (see Section 31.2.5 [File Variables], page 468).

Le choix de renommer ou de copier est contrôlé par quatre variables. Renommer est le choix par défaut. Si la variable `backup-by-copying` ess non `nil`, la copie est utilisée. copying is used. Autrement, si la variable `backup-by-copying-when-linked` est non `nil`, alors la copie est utilisée pour des fichiers ayant plusieurs noms, mais le renommage est toujours utilisé lorsque le fichier édité n'a qu'un seul nom. Si la variable `backup-by-copying-when-mismatch` est non `nil`, alors la copie est utilisée si le renommage peut causer le changement de propriétaire ou de groupe. `backup-by-copying-when-mismatch` est `t` par défaut si vous démarrez Emacs en tant que super-utilisateur. La quatrième variable, `backup-by-copying-when-privileged-mismatch`, donne l'id utilisateur le plus haut pour lequel `backup-by-copying-when-mismatch` sera forcé. Ceci est utile lorsque les id-utilisateur les plus petits sont assignés à des utilisateurs système spéciaux,

comme `root`, `bin`, `daemon`, etc., qui doivent garder la propriété de leurs fichiers.

Lorsqu'un fichier est géré par un système de contrôle de version ((see Section 14.7 [Version Control], page 161), Emacs ne crée normalement pas d'archives pour ce fichier. Mais l'enregistrement et le retrait sont de quelques manières similaires à la création d'archives. Une similarité malheureuse est que ces opérations rompent généralement les liens durs, en déconnectant le nom du fichier visité des noms alternatifs du même fichier. Ceci n'a rien à voir avec Emacs—c'est le système de contrôle de version qui fait cela.

14.3.2 Protection contre des Éditions Simultanées

Des éditions simultanées arrivent lorsque deux utilisateurs visitent le même fichier, tous deux font des changements, puis tous deux l'enregistre. Si personne n'était informé du fait, l'utilisateur ayant enregistré en premier verrait plus tard que ses changements étaient perdus.

Sur certains systèmes, Emacs remarque immédiatement lorsque le deuxième utilisateur commence à modifier le fichier, et l'avertit de suite. Sur tous les systèmes, Emacs vérifie lorsque vous enregistrez le fichier, et vous avertit si vous êtes sur le point d'écraser le changements d'un autre utilisateur. Vous pouvez éviter la perte du travail de l'autre utilisateur en faisant l'action corrective appropriée plutôt que d'enregistrer le fichier.

Lorsque vous faites la première modification dans un tampon d'Emacs visitant un fichier, Emacs note que le fichier est *vérouillé* par vous. (Il fait ceci en créant un lien symbolique dans le même répertoire avec un nom différent.) Emacs supprime le verrou lorsque vous enregistrez les changements. L'idée est qu'un fichier est vérouillé lorsqu'un tampon d'Emacs le visitant contient des changements non sauvegardés.

Si vous commencez à modifier le tampon lorsque le fichier visité est vérouillé par quelqu'un d'autre, cela constitue une *collision*. Lorsqu'Emacs détecte une collision, il vous demande que faire, en appelant la fonction Lisp `ask-user-about-lock`. Vous pouvez redéfinir cette fonction à titre de personalisation. La définition standard de cette fonction vous pose une question et accepte trois réponses possibles :

`s` Dérobe le verrou; Quiconque étant en train de modifier le fichier perd le verrou, et vous obtenez le verrou.

`p` Passe. Continue et édite le fichier en dépit qu'il soit édité par quelqu'un d'autre.

`q` Quitte. Ceci entraîne une erreur (`file-locked`) et les modifications que vous essayiez de faire dans le tampon ne prennent pas effet.

Notez que le verrouillage se base sur les noms de fichiers ; si un fichier a plusieurs noms, Emacs ne réalise pas que les deux noms représentent le même

fichier, et ne peut pas empêcher deux utilisateurs d'éditer ce même fichier simultanément sous deux noms différents. Cependant, baser le verrouillage sur les noms permet à Emacs de verouiller l'édition de nouveaux fichiers qui n'existeront pas réellement avant d'être enregistrés.

Certains systèmes ne sont pas configurés pour permettre à Emacs de créer des verrous, et dans ces cas les fichiers verrous ne peuvent être écrits. Dans ces cas, Emacs ne peut pas détecter de problèmes à l'avance, mais il peut toujours détecter la collision lorsque vous essayez d'enregistrer un fichier et d'écraser les changements de quelqu'un d'autre.

Si Emacs ou le système d'exploitation crashe, ceci peut laisser des fichiers verrous qui n'ont plus lieu d'exister, vous pouvez alors à l'occasion obtenir des avertissements à propos de fausses collisions. Lorsque vous déterminez que la collision est irrélevante, tapez `p` pour indiquer à Emacs d'ignorer la collision.

Chaque fois qu'Emacs enregistre un tampon, il examine d'abord la date de dernière modification du fichier existant sur le disque pour vérifier qu'il n'a pas changé depuis la dernière visite ou sauvegarde de ce fichier. Si la date ne correspond pas, cela veut dire que des changements ont été faits sur le fichier d'une autre manière, et ces changements seront perdus si Emacs enregistre. Pour empêcher ceci, Emacs affiche un message d'avertissement et vous demande de confirmer avant d'enregistrer. Parfois vous saurez de quelle manière le fichier a été modifié et saurez que e n'est pas important ; vous pouvez alors répondre `yes` et poursuivre. Autrement, vous devriez annuler l'enregistrement avec `C-g` et examiner la situation.

La première chose que vous devriez faire lorsque vous êtes informé que des éditions simultanées sont en cours est de lister le répertoire avec `C-u C-x C-d` (see Section 14.8 [Directories], page 180). L'auteur courant du fichier est ainsi affiché. Vous devriez essayer de le contacter pour l'avertir de ne pas continuer à éditer. Souvent, la prochaine étape est de sauvegarder le contenu de votre tampon Emacs sous un nom différent, et utiliser `diff` pour comparer les deux fichiers.

14.3.3 Ombrer des fichiers

`M-x shadow-initialize`
: Initialise l'ombrage des fichiers.

`M-x shadow-define-literal-group`
: Déclare un fichier unique à être partagé entre plusieurs sites.

`M-x shadow-define-regexp-group`
: Rend tous les fichiers correspondant partagés entre plusieurs sites.

`M-x shadow-define-cluster` ⟨RET⟩ *nom* ⟨RET⟩
: Définit une grappe de fichier ombrés *nom*.

`M-x shadow-copy-files`
: Copie tous les fichiers ombrés en attente.

`M-x shadow-cancel`
: Annule l'instruction d'ombrer certains fichiers.

Vous pouvez vous arranger pour garder identique des copies *ombrées* de certains fichiers en plusieurs endroits—pourquoi pas sur des machines différentes. Pour faire cela, vous deez d'abord créer un *groupe de fichiers ombrés*, qui est un ensemble de fichiers de même nom partagés entre une liste de sites. Le groupe de fichiers est permanent et s'applique aux futures sessions d'Emacs aussi bien qu'à celle en cours. Une fois le groupe créé, chaque fois que vous quittez Emacs, il copiera le fichier que vous avez édité dans les autres fichiers de son groupe. Vous pouvez aussi lancer cette copie sans quitter Emacs, en tapant `M-x shadow-copy-files`.

Pour créer un groupe de fichiers, utilisez `M-x shadow-define-literal-group` ou `M-x shadow-define-regexp-group`. Voir leurs chaînes de documentation pour plus d'informations.

Avant de copier un fichier dans ses fichiers ombres, Emacs demande confirmation. Vous pouvez répondre "no" pour annuler la copie de ce fichier, cette fois. Si vous voulez annuler de manière permanente l'ombrage de certains fichiers, utilisez `M-x shadow-cancel` pour éliminer ou changer le groupe de fichiers ombrés.

Une *grappe ombrée* est un groupe de postes partageant des répertoires, pour lequel copier depuis ou vers l'un d'eux est suffisant pour mettre à jour le fichier sur chacun d'eux. Chaque grappe ombrée a un nom, et spécifie l'adresse réseau d'un poste primaire (celui sur lequel on copie le fichier), et une expression rationnelle qui correspond aux noms de tous les autres postes de la grappe. Vous pouvez définir une grappe ombrée avec `M-x shadow-define-cluster`.

14.3.4 Mise à jour automatique de timbres dateurs

Vous pouvez placer un timbre dateur dans un fichier, pour qu'il soit mis à jour automatiquement chaque fois que vous éditez et enregistrez le fichier. Le timbre dateur doit se trouver dans les huit premières lignes du fichier, et vous devez l'insérer de cette manière :

```
Time-stamp: <>
```

ou ainsi :

```
Time-stamp: ""
```

Ajoutez alors la fonction crochet `time-stamp` au crochet `write-file-hooks`; cette fonction crochet mettra automatiquement à jour le timbre dateur, en insérant la date et l'heure de la sauvegarde di fichier. Vous pouvez aussi utiliser la commande `M-x time-stamp` pour mettre à jour manuellement le timbre dateur. Pour d'autres personnalisations, voir le groupe de

personnalisation `time-stamp`. Notez que les champs non numériques du timbre dateur sont formatés selon votre localisation. (see Section B.5 [Environment], page 511).

14.4 Faire Revenir un Tampon

Si vous avez fait des changements importants à un fichier puis changez d'avis, vous pouvez vous en débarasser en lisant la version précédente du fichier. Pour faire ceci, utilisez `M-x revert-buffer`, qui opère sur le tampon courant. Faire revenir un tampon pouvant occasionner une perte importante de votre travail, vous devez confirmer cette commande avec `yes`.

`revert-buffer` laisse le point à la même distance (mesurée en caractères) du début du fichier. Si le fichier n'a été que légèrement édité, vous vous retrouverez dans la même partie de texte que précédemment. Si vous avez fait des changements radicaux, la même valeur du point dans l'ancien fichier peut adresser une partie de texte totalement différente.

Le retour marque le tampon comme "non modifié" avant que de nouveaux changements ne soient faits.

Certains types de tampons dont le contenu représente des bases de données autres que des fichiers, comme des tampons Dired, peuvent aussi être retournés. Pour ceux-là, faire revenir veut dire recalculer leur contenu à partir de la base de données appropriée. Les tampons créés explicitement avec `C-x b` ne peuvent être retournés ; `revert-buffer` reporte une erreur lorsqu'on lui demande de le faire.

Lorsque vous éditez un fichier qui change automatiquement et fréquemment—par exemple, un journal en sortie d'un processus qui continue de s'exécuter–il peut être intéressant pour Emacs de faire revenir le fichier sans vous le demander, lorsque vous visitez de nouveau le fichier avec `C-x C-f`.

Pour demander ce comportement, définissez la variable `revert-without-query` comme une liste d'expressions rationnelles. Lorsqu'un nom de fichier correspond à l'une de ces expressions rationnelles, `find-file` et `revert-buffer` le fera revenir automatiquement s'il a changé—en supposant que le tampon lui-même n'a pas été modifié. (Si vous avez édité le texte, il pourrait être mal venu d'annuler vos changements.)

14.5 Auto-Sauvegarde : Protection Contre des Désastres

Emacs enregistre tous les fichiers visités de temps en temps (basé sur le compte de vos pressions de touches) sans vous avertir. Ceci est appelé *auto-sauvegarde*. Ceci vous empêche de perdre une trop importante quantité de travail si le système crashe.

Lorsqu'Emacs détermine qu'il est temps d'auto-sauvegarder, chaque tampon est considéré et est auto-sauvegardé si l'auto-sauvegarde est effective pour ce tampon et qu'il a été modifié depuis la dernière auto-sauvegarde. Le message '`Auto-saving...`' est affiché dans la zone de répercussion lors de l'auto-sauvegarde, et tous les fichiers sont auto-sauvegardés. Les erreurs arrivant en cours d'auto-sauvegarde sont détournées pour qu'elles ne puissent pas interférer avec l'exécution des commandes que vous êtes en train de taper.

14.5.1 Fichier d'Auto-Sauvegarde

L'auto-sauvegarde n'enregistre pas dans les fichiers que vous visitez, car il peut être indésirable de sauvegarder un programme qui est dans un état inconsistant lorsque vous avez fait la moitié du changement souhaité. L'auto-sauvegarde est plutôt fait dans un fichier différent appelé *fichier d'auto-sauvegarde*, et le fichier visité est changé seulement lorsque vous demandez l'enregistrement (comme avec `C-x C-s`).

Normalement, le nom du fichier de sauvegarde est créé en ajoutant '`#`' au début et à la fin du nom du fichier visité. Ainsi, un tampon visitant le fichier '`foo.c`' est auto-sauvegardé dans un fichier '`#foo.c#`'. La plupart des fichiers ne visitant pas de fichiers sont auto-sauvegardés seulement si vous en faîtes la demande ; lorsqu'ils sont auto-sauvegardés, le nom du fichier d'auto-sauvegarde est créé en ajoutant '`#%`' au début et '`#`' à la fin du nom du tampon. Par exemple, le tampon '`*mail*`' dans lequel vous tapez des messages à envoyer est auto-sauvegardé dans un fichier appelé '`#%*mail*#`'. Les noms des fichiers d'auto-sauvegarde sont créés de cette manière à moins que vous reprogrammiez une partie d'Emacs pour faire quelque chose de différent (les fonctions `make-auto-save-file-name` et `auto-save-file-name-p`). Le nom de fichier à utiliser pour l'auto-sauvegarde d'un tampon est calculé au moment où l'auto-sauvegarde est permise dans ce tampon.

Lorsque vous supprimez une partie conséquente de texte dans un long tampon, l'auto-sauvegarde est arrêtée temporairement dans ce tampon. Dans les cas où vous supprimez le texte accidentellement, vous pouvez trouver plus utile que le fichier d'auto-sauvegarde contienne le texte supprimé. Pour relancer l'auto-sauvegarde, enregistrez le tampon avec `C-x C-s`, ou utilisez `C-u 1 M-x auto-save`.

Si vous désirez que l'auto-sauvegarde se fasse sur le fichier visité, mettez la variable `auto-save-visited-file-name` à une valeur non `nil`. Dans ce mode, il n'y a aucune différence entre l'auto-sauvegarde et la sauvegarde explicite.

Le fichier d'auti-sauvegarde d'un tampon est effacé lorsque vous enregistrez le tampon dans le fichier qu'il visite. Pour empêcher ceci, mettez la variable `delete-auto-save-files` à `nil`. Le fait de changer le nom du fichier visité avec `C-x C-w` ou `set-visited-file-name` renomme le fichier

d'auto-sauvegarde pour que son nom corresponde au nouveau nom du fichier visité.

14.5.2 Contrôler l'Auto-Sauvegarde

Chaque fois que vous visitez un fichier, l'auto-sauvegarde est lancée pour le fichier de ce tampon si la variable `auto-save-default` est non `nil` (mais pas en mode batch ; see Chapter 3 [Entering Emacs], page 37). La valeur par défaut pour cette variable est `t`, ainsi la sauvegarde automatique est la pratique habituelle pour les tampons visitant des fichiers. L'auto-sauvegarde peut être lancée ou arrêtée pour chacun des tampons existants avec la commande `M-x auto-save-mode`. Comme les autres commandes de modes mineurs, `M-x auto-save-mode` lance l'auto-sauvegarde avec un argument positif, l'arrête avec un argument nul ou négatif ; sans argument, le comportement est changé de lancé à arrêté et inversement.

Emacs auto-sauvegarde au bout d'une certaine période, basée sur le nombre de caractères que vous avez tapé depuis la dernière auto-sauvegarde. La variable `auto-save-interval` spécifie le nombre de caractères entre les auto-auvegardes. La valeur par défaut est 300.

L'auto-sauvegarde est aussi déclenchée lorsque vous arrêtez de taper pendant un moment. La variable `auto-save-timeout` indique combien de secondes Emacs doit attendre avant de faire une auto-sauvegarde (et peut-être aussi une collecte d'ordures). (La période de temps est plus longue si le tampon courant est long ; ceci est une heuristique qui vise à ne pas vous perturber lorsque vous éditez de longs tampons, pour lesquels l'auto-sauvegarde prend un temps important.) L'auto-sauvegarde pendant des périodes d'inactivité accomplit deux choses : tout d'abord, elle assure que tout votre travail est sauvegardé si vous vous éloignez de votre terminal pour un moment ; ensuite, elle peut annuler quelques auto-sauvegardes lorsque vous êtes en train de taper.

Emacs fait aussi une auto-sauvegarde lorsqu'il détecte une erreur fatale. Ceci inclut tuer le processus Emacs avec une commande shell comme '`kill %emacs`', ou déconnecter une ligne téléphonique ou une connexion réseau.

Vous pouvez demander explicitement une auto-sauvegarde avec la commande `M-x do-auto-save`.

14.5.3 Retrouver des Données d'après des Auto-sauvegardes

Vous pouvez utiliser le contenu d'un fichier d'auto-sauvegarde pour retrouver des données perdues avec la commande `M-x recover-file` ⟨RET⟩ *fichier* ⟨RET⟩. Cette commande visite *fichier* puis (après votre confirmation) en restaure sle contenu d'après son fichier d'auto-sauvegarde '`#`*fichier*`#`'.

Vous pouvez alors enregistrer avec `C-x C-s` pour placer le texte retrouvé dans *fichier* lui-même. Par exemple, pour retrouver le fichier '`foo.c`' depuis son fichier d'auto-sauvegarde '`#foo.c#`', faites :

```
M-x recover-file ⟨RET⟩ foo.c ⟨RET⟩
yes ⟨RET⟩
C-x C-s
```

Avant de demander la confirmation, `M-x recover-file` affiche une liste du répertoire décrivant le fichier spécifié et le fichier d'auto-sauvegarde, pour que vous puissiez comparer leurs tailles et dates. Si le fichier d'auto-sauvegarde est plus ancien, `M-x recover-file` ne propose pas de le lire.

Si Emacs ou l'ordinateur crashe, vous pouvez retrouver tous les fichiers que vous éditiez depuis leurs fichiers d'auto-sauvegarde avec la commande `M-x recover-session`. Cette commande vous affiche d'abord une liste de sessions interrompues enregistrées. Déplacez le point sur celui de votre choix, et tapez `C-c C-c`.

Puis `recover-session` vous demande, pour chaque fichier qui était édité durant la session, s'il faut retrouver ce fichier. Si vous répondez `y`, elle appelle `recover-file`, qui fonctionne comme d'habitude. Elle affiche les dates du fichier et de son fichier d'auto-sauvegarde, et demande de nouveau s'il faut retrouver ce fichier.

Lorsque `recover-session` est terminé, les fichiers que vous avez choisi de retrouver sont présents dans des tampons d'Emacs. Vous devez alors les enregistrer. Seulement ceci—les enregistrer—met à jour les fichiers eux-mêmes.

+ Emacs records interrupted sessions for later recovery in files named +'`~/.emacs.d/auto-save-list/.saves-`*pid*-*hostname*'. The +'`~/.emacs.d/auto-save-list/.saves-`' portion of these names comes +from the value of `auto-save-list-file-prefix`. You can record +sessions in a different place by customizing that variable. If you

Emacs enregistre les sessions interrompues pour de futures récupérations dans des fichiers appelés '`~/.emacs.d/auto-save-list/.saves-`*pid*-*hostname*'. La portion '`~/.emacs.d/auto-save-list/.saves-`' de ces noms vient de la valeur de `auto-save-list-file-prefix`. Vous pouvez choisir d'enregistrer les sessions à un endroit différent en redéfinissant cette variable. Si vous mettez `auto-save-list-file-prefix` à `nil` dans votre fichier '`.emacs`', les sessions ne sont pas enregistrées.

14.6 Alias de Noms de Fichiers

Les liens symboliques et les liens durs rendent tous deux possible pour plusieurs fichiers de se référer à un même fichier. Les liens durs sont des noms alternatifs qui se réfèrent directement au fichier ; tous les noms sont également valides, et aucun d'eux n'est préféré. Par contraste, un lien sym-

bolique est une sorte de définition d'alias : lorsque '`foo`' est un lien symbolique vers '`bar`', vous pouvez utiliser l'un ou l'autre des noms pour vous référer au fichier, mais '`bar`' est le nom réel, alors que '`foo`' est seulement un alias. Des cas plus complexes arrivent lorsque des liens symboliques pointent vers des répertoires.

Si vous visitez deux noms d'un même fichier, Emacs crée normalement deux tampons différents, mais il vous avertit de la situation.

Normalement, si vous visitez un fichier qu'Emacs est déjà en train de visiter sous un autre nom, Emacs affiche un message dans la zone de répercussion et utilise le tampon existant quivisite ce fichier. Ceci peut arriver sur des systèmes supportant les liens symboliques, ou si vous utilisez un nom de fichier long sur un système qui coupe les noms de fichiers longs. Vous pouvez interdire cette fonctionnalité en définissant la variable `find-file-existing-other-name` à `nil`. Ainsi, lorsque vous visitez un même fichier sous deux noms différents, vous obtenez un tampon différent pour chaque nom de fichier.

Si la variable `find-file-visit-truename` est non `nil`, alors le nom de fichier sauvegardé pour un tampon est le *nom réel* du fichier (créé enremplaçant tout lien symbolique par le nom de sa cible), plutôt que le nom que vous spécifiez. Définir `find-file-visit-truename` implique aussi l'effet de `find-file-existing-other-name`.

14.7 Contrôle de Version

Les *Systèmes de contrôle de version* sont des paquetages qui peuvent enregistrer de multiples versions d'un fichier source, habituellement en enregistrant les parties inchangées du fichier une seule fois. Les systèmes de contrôle de version enregistrent aussi des informations historiques comme la date de création de chaque version, leur créateur, et une description des changements dans cette version.

L'interface de contrôle de version d'Emacs est appelé VC. Ses commandes fonctionnent avec trois systèmes de contrôle de version—RCS, CVS et SCCS. Le Projet GNU recommande RCS et CVS, qui sont des logiciels libres et disponibles auprès de la Free Software Foundation. Nous avons aussi du logiciel libre pour remplacer SCCS, appelé CSSC ; si vous utilisez SCCS et ne voulez pas faire le changement incompatible vers RCS ou CVS, vous pouvez passer à CSSC.

14.7.1 Introduction au contrôle de version

VC vous permet d'utiliser un système de contrôle de version depuis Emacs, en intégrant les opérations de contrôle de version avec l'édition. VC fournit une interface uniforme pour le contrôle de version, aisni quel que soit

le système de contrôle de version utilisé, vous pouvez l'utiliser de la même manière.

Cette section fournit un aperçu général du contrôle de version, et décrit les systèmes de contrôle de version supportés par VC. Vous pouvez sauter cette section si vous êtes déjà familiers avec le système de contrôle de version que vous voulez utiliser.

14.7.1.1 Systèmes de Contrôle de Version Supportés

VC fonctionne à cette heure avec trois systèmes de contrôle de version ou "back ends" différents : RCS, CVS et SCCS.

RCS est un système de contrôle de version libre disponible auprès de la Free Software Foundation. Il est peut-être le plus mature des back ends supportés, et les commandes de VC sont conceptuellement plus proches de RCS. Tout ce que vous pouvez faire avec RCS peut être fait à travers VC.

CVS est construit au dessus de RCS, et étend les possibilités de RCS, permettant une gestion des versions sophistiquée, et le développement multi-utilisateurs concurrents. VC supporte les opérations d'édition élémentaires sous CVS, mais pour certaines tâches moins ordinaires, vous aurez toujours besoin d'invoquer CVS depuis la ligne de commande. Notez aussi qu'avant d'utiliser CVS, vous devrez mettre en place un dépositaire, ce qui est un sujet trop complexe à traiter ici.

SCCS est un système de contrôle de version propriétaire mais très utilisé. En termes d'aptitudes, il est le plus faible que VC supporte. VC compense certaines caractéristiques manquantes de SCCS (les instantanés, par exemple) en les implémentant lui-même, mais certaines autres caractéristiques de VC, comme les branches multiples, ne sont pas disponibles avec SCCS. Vous devriez utiliser SCCS seulement si pour quelque raison vous ne pouvez pas utiliser RCS.

14.7.1.2 Concepts du Contrôle de Version

Lorsqu'un fichier est sous contrôle de version, nous disons aussi qu'il est *déclaré* dans le système de contrôle de version. Chaque fichier déclaré a un *fichier maître* correspondant qui représente l'état actuel du fichier plus l'historique de ses changements, de sorte que vous pouvez reconstruire à partir de celui-ci soit la version courante, soit toute version précédente. Habituellement, le fichier maître enregistre aussi une *entrée journal* pour chaque version, décrivant ce qui a été changé dans cette version.

Le fichier qui est maintenu sous contrôle de version est parfois appelé le *fichier de travail* correspondant à son fichier maître.

Pour examiner un fichier, vous le *retirez*. Ceci extrait une version du fichier source (d'ordinaire la plus récente) du fichier maître. Si vous désirez éditer le fichier, vous devez le retirer *verrouillé*. Seul un utilisateur à la fois peut faire ceci pour un fichier source donné. (cette sorte de verrouillage est sans aucun rapport avec le verrouillage qu'utilise Emacs pour détecter des éditions simultanées d'un fichier.)

Lorsque vous avez fini votre édition, vous devez *faire enregistrer* la nouvelle version. Ceci enregistre la nouvelle version dans le fichier maître, et déverrouille le fichier source pour qu'un autre utilisateur puisse le verrouiller et ainsi le modifier.

Retirer et faire enregistrer sont les opérations élémentaires du contrôle de version. Vous pouvez les faire toutes deux avec une commande Emacs unique : `C-x C-q` (`vc-toggle-read-only`).

Il existe cependant des variantes de ce schéma élémentaire. CVS, par exemple, n'utilise pas le verrouillage, et vous pouvez alors éditer normalement les fichiers, sans avoir à les retirer d'abord. See Section 14.7.2.6 [CVS and VC], page 167. Avec RCS, vous pouvez optionnellement sélectionner un *verrouillage non strict* pour un fichier source particulier ; vous pouvez alors éditer le fichier sous Emacs sans le verrouiller explicitement.

Un *instantané* est une collection cohérente de versions des divers fichiers composant un programme. See Section 14.7.9 [Snapshots], page 175.

14.7.2 Éditer avec Contrôle de Version

Voici les commandes pour éditer un fichier maintenu par contrôle de version :

`C-x C-q`
`C-x v v` Retire ou fait enregistrer le fichier visité.

`C-x v u` Retourne le tampon et le fichier à la dernière version enregistrée.

`C-x v c` Retire le dernier changement entré dans le maître pour le fichier visité. Ceci annule votre dernier enregistrement.

`C-x v i` Déclare le fichier visité au contrôle de version.

(`C-x v` est la touche préfixe pour les commandes de contrôle de version ; toutes ces commandes à l'exception de `C-x C-q` commencent avec `C-x v`.)

14.7.2.1 Retrait

Lorsque vous désirez modifier un fichier maintenu par contrôle de version, tapez `C-x C-q` (`vc-toggle-read-only`). Ceci *retire* le fichier, et indique à RCS ou SCCS de verrouiller le fichier. Ceci rend le fichier accessible en écriture pour vous (et non pour les autres).

Si vous spécifiez un argument préfixe (`C-u C-x C-q`) pour retirer un fichier, Emacs vous demande un numéro de version, et retire cette version *déverouillée*. Ceci vous permet de vous rendre à d'anciennes versions, ou à des branches existantes de ce fichier (see Section 14.7.6 [Branches], page 172). Vous pouvez alors commencer à éditer la version sélectionnée en tapant `C-x C-q` de nouveau. (Si vous éditez une ancienne version d'un fichier de cette manière, le fait de le faire enregistrer de nouveau crée une nouvelle branche.)

Sous CVS, il n'est normalement pas nécessaire de retirer explicitement les fichiers. CVS n'a pas de verrouillage ; plusieurs utilisateurs peuvent éditer leurs copies d'un fichier lorsqu'ils le veulent. (Si deux utilisateurs font des changements conflictuels, ils doivent réconcilier leurs changements lorsqu'ils les enregistrent.) On dit alors qu'un retrait *implicite* a lieu au premier changement d'un fichier.

CVS a un mode alternatif dans lequel le retrait explicite est obligatoire. Et RCS a un moe alternatif appelé *verrouillage non strict* dans lequel le retrait explicite n'est pas obligatoire. La sélection de ces modes est faite en dehors de VC, mais une fois que vous les avez sélectionnés, VC y obéit. Avec RCS, vous pouvez sélectionner le verrouillage non strict pour un fichier particulier avec la commande '`rcs -U`'. See Section 14.7.2.6 [CVS and VC], page 167, pour une explication sur la manière de faire cela avec CVS.

14.7.2.2 Enregistrement

Lorsque vous avez fini d'éditer le fichier, tapez `C-x C-q` de nouveau. Utilisé sur un fichier retiré, cette commande enregistre le fichier. Mais l'enregistrement n'a pas lieu immédiatement ; vous devez d'abord saisir *l'entrée journal*—une description des changements dans la nouvelle version. `C-x C-q` ouvre un tampon pour que vous l'y entriez. Lorsque vous avez fini de saisir l'entrée journal, tapez `C-c C-c` pour terminer ; c'est à ce moment que l'enregistrement a réellement lieu. See Section 14.7.3 [Log Entries], page 168.

Avec RCS et SCCS, un fichier retiré est aussi *verrouillé*, ce qui veut dire qu'il est modifiable par vous, et par personne d'autre. Aussi longtemps que vous détenez le verrou sur le fichier, personne d'autre ne peut le modifier, et personne ne peut faire enregistrer de changements à cette version. Faire enregistrer vos changements déverrouille le fichier, et ainsi un autre utilisateur peut le verrouiller et le modifier.

CVS, au contraire, n'a pas de concept de verrouillage. Les fichiers de travail sont toujours modifiables, permettant des développements concurrents, avec des conflits possibles résolus au moment de l'enregistrement. See Section 14.7.2.6 [CVS and VC], page 167.

Pour spécifier le numéro de version de la nouvelle version, tapez `C-u C-x C-q` pour faire enregistrer un fichier. Emacs vous demande alors dans le mini-tampon le nouveau numéro de version. Ceci peut être utilisé pour créer

une nouvelle *branche* du fichier (see Section 14.7.6 [Branches], page 172), ou pour incrémenter le numéro majeur de version du fichier.

Il n'est pas impossible de verrouiller un fichier que quelqu'un d'autre a déjà verrouillé. Si vous essayez de retirer un fichier qui est verrouillé, `C-x C-q` vous demande si vous désirez "dérober le verrou". Si vous répondez oui, le fichier devient verrouillé par vous, mais un message est envoyé à la personne qui avait précédemment verrouillé le fichier, pour l'informer du fait. La ligne de mode indique qu'un fichier est verrouillé par quelqu'un d'autre en inscrivant le nom de login de cette personne, avant le numéro de version.

14.7.2.3 Déclarer un Fichier au Contrôle de Version

`C-x v i` déclare le fichier visité au contrôle de version.

Vous pouvez placer n'importe quel fichier sous contrôle de version en le visitant simplement, puis en tapant `C-x v i` (`vc-register`). Après `C-x v i`, le fichier est déverrouillé et en lecture seule. Tapez `C-x C-q` si vous désirez commencer à l'éditer.

Lorsque vous déclarez le fichier, Emacs doit choisir quel système de contôle de version utiliser pour ce fichier. Vous pouvez spécifier explicitement votre choix en mettant `vc-default-back-end` à `RCS`, `CVS` ou `SCCS`. Autrement, s'il existe un sous-répertoire appelé '`RCS`', '`SCCS`', ou '`CVS`', Emacs utilise le système de contrôle de version correspondant. En l'absence de toute spécification, le choix par défaut est RCS si RCS est installé, ou autrement SCCS.

Après avoir déclaré un fichier avec CVS, vous devez donner sa version initiale en tapant `C-x C-q`. See Section 14.7.2.6 [CVS and VC], page 167.

Le numéro de version initial pour un fichier nouvellement déclaré est 1.1, par défaut. Pour spécifier un numéro différent, donnez à `C-x v i` un argument numérique ; il lit alors le numéro de vesion initial en utilisant le mini-tampon.

Si `vc-initial-comment` est non `nil`, `C-x v i` lit un commentaire initial (un peu comme une entrée journal) décrivant l'usage du fichier source.

14.7.2.4 Annuler des Actions du Contrôle de Version

`C-x v u` Fait revenir le tampon et le fichier à la dernière version enregistrée.

`C-x v c` Retire le dernier changement entré dans le maître pour le fichier visité. Ceci annule votre dernier enregistrement.

Si vous voulez annuler vos changements courants et revenir à la dernière version enregistrée, utilisez `C-x v u` (`vc-revert-buffer`). Ceci annule votre

dernier retrait, laissant le fichier déverrouillé. Si vous désirez faire de nouveaux changements, vous devrez d'abord retirer le fichier de nouveau. `C-x v u` nécessite une confirmation de votre part, à moins que vous n'ayez pas fait de changements par rapport à la dernière version enregistrée.

`C-x v u` est aussi la commande à utiliser pour déverrouiller un fichier que vous avez verrouillé puis décidez de ne pas changer.

Vous pouvez annuler un changement après l'avoir enregistré, avec `C-x v c` (`vc-cancel-version`). Cette commande efface toute sauvegarde de la version la plus récemment enregistrée. `C-x v c` offre aussi la possibilité de faire revenir votre fichier de travail et tampon à la version précédente (celle précédant la version effacée). Si vous répondez `no`, alors VC garde vos changements dans le tampon et verrouille le fichier.

L'option de non retour est utile lorsque vous avez fait enregistrer un changement puis découvrez une erreur triviale dans ces changements ; vous pouvez annuler l'enregistrement erroné, corriger l'erreur, et faire enregistrer le fichier de nouveau.

Lorsque `C-x v c` ne fait pas revenir le tampon, les entêtes de contrôle de version du tampon ne sont plus développées (sont enveloppées) (see Section 14.7.10 [Version Headers], page 176). La cause est que le tampon ne correspond plus à aucune version existante. Si vous faites enregistrer de nouveau le tampon, le processus d'enregistrement développera correctement les entêtes pour le nouveau numéro de version.

Cependant, il est impossible d'envelopper l'entête RCS '`$Log$`' automatiquement. Si vous utilisez cette caractéristique des entêtes, vous devez l'envelopper à la main–en effaçant l'entrée pour la version que vous venez d'effacer.

Soyez attentif lorsque vous invoquez `C-x v c`, car il est facile de perdre de cette manière beaucoup de travail. Pour vous aider à être attentif, cette commande vous demande toujours de confirmer par `yes`. Notez aussi que cette commande n'est pas disponible sous CVS, car annuler des versions est très dangereux et déconseillé avec CVS.

14.7.2.5 La Ligne de Mode de VC

Lorsque vous visitez un fichier qui est sous contrôle de version, la ligne de mode indique le status courant du fichier : le nom du système de contrôle de version utilisé pour ce fichier, l'état de verrouillage, et la version.

L'état du verrouillage est affiché avec un caractère unique, qui peut être soit '`-`' soit '`:`'. '`-`' indique que le fichier n'est pas verrouillé ou non modifié par vous. Une fois que vous verrouillez le fichier, l'indicateur d'état se change en '`:`'. Si le fichier est verrouillé par une autre personne, le nom de l'utilisateur apparaît après le numéro de version.

Par exemple, '`RCS-1.3`' veut dire que vous examinez la version 1.3 par RCS, qui n'est pas verrouillée. '`RCS:1.3`' veut dire que vous avez verrouillé le fichier, et avez pu commencé à le modifier. '`RCS:jim:1.3`' indique que le fichier est verrouillé par jim.

14.7.2.6 Utiliser VC avec CVS

Avec CVS, les fichiers ne sont jamais verrouillés. Deux utilisateurs peuvent retirer le même fichier en même temps ; chaque utilisateur obtient une copie séparée et peut l'éditer. Les fichiers de travail sont toujours modifiables ; une fois retiré, vous n'avez pas à taper une commande de VC pour commencer à éditer le fichier. Vous pouvez l'éditer à n'importe quel moment.

En utilisant RCS et SCCS, vous utilisez normalement `C-x C-q` deux fois pour chaque changement ; une fois avant le changement, pour retirer le fichier, et une fois après, pour le faire enregistrer. Avec CVS, c'est différent : vous utilisez normalement `C-x C-q` une seule fois pour chaque changement, pour soumettre le changement une fois fait. Le fichier de travail reste modifiable, et vous pouvez de nouveau commencer à l'éditer sans utiliser de commande spéciale.

Une manière de comprendre ceci est que VC fait un retrait *implicite* lorsque vous enregistrez le fichier modifié pour la première fois. VC l'indique dans la ligne de mode : l'indicateur de status change de '`-`' en '`:`' aussitôt que vous enregistrez une version modifiée, vous indiquant que vous n'êtes plus synchronisé avec le dépositaire (see Section 14.7.2.5 [VC Mode Line], page 166). Le fichier reste "retiré" tant que vous ne le faites pas enregistrer de nouveau, même si vous détruisez le tampon et visitez de nouveau le fichier.

Si, plutôt, vous voudriez utiliser le retrait explicite avec CVS, mettez la variable d'environnement `CVSREAD` à une valeur quelconque. (La valeur que vous lui donnez n'importe pas.) CVS rend alors vos fichiers de travail en lecture seule par défaut, et VC attend que vous les retiriez explicitement avec `C-x C-q`. Lorsque vous définissez `CVSREAD` pour la première fois, faites attention de retirer tous vos modules de nouveau, pour que les protections des fichiers soient correctement définies.

VC ne fournit pas de moyen de retirer une copie de travail d'un fichier existant du dépositaire. Vous devez utiliser les commandes shell de CVS pour faire cela. Une fois que vous avez un fichier de travail, vous pouvez commencer à utiliser VC pour ce fichier.

La terminologie de CVS parle de *soumettre* un changement plutôt que de le faire enregistrer. Mais en termes pratiques ils marchent de la même manière : Emacs vous demande de taper une entrée journal, et vous la terminez avec `C-c C-c`.

Lorsque vous tentez de soumettre les changements d'un fichier, mais que quelqu'un d'autre a soumis d'autres changements pendant ce temps, un *conflit* est créé. VC détecte cette situation et propose de *mêler* vos changements

et ceux de l'autre utilisateur, créant une nouvelle version locale du fichier, que vous pouvez alors soumettre au dépositaire. Ceci marche bien si les changements sont faits sur différentes parties du fichier, bien qu'il soit prudent de vérifier l'uniformité sémantique du fichier résultant.

Cependant, si vous et l'autre utilisateur avez modifié les mêmes parties du fichier, le conflit ne peut être résolu automatiquement. Dans ce cas, CVS insére les deux variantes des régions en conflit dans le fichier de travail, et place des *marques de conflit* autour d'elles. Elles indiquent à quoi ressemble la région dans les versions respectives de chaque utilisateur. Vous devez résoudre le conflit manuellement, par exemple en choisissant une des deux variantes et en supprimant l'autre (et les marques de conflit). Vous pouvez alors soumettre le fichier résultant au dépositaire. L'exemple suivant montre l'apparence d'une région en conflit ; le fichier est appelé '`nom`' et la version courante du dépositaire avec les changements de l'utilisateur B est 1.11.

```
<<<<<<< nom
  User A's version
=======
  User B's version
>>>>>>> 1.11
```

Vous pouvez empêcher l'utilisation de VC pour les fichiers gérés par CVS en mettant la variable `vc-handle-cvs` à `nil`. Si vous faites cela, Emacs traite ces fichiers comme s'ils n'étaient pas gérés, et les commandes VC ne sont pas disponibles pour ceux-là. Vous devez faire toutes les opérations CVS manuellement.

14.7.3 Entrée journal

Lorsque vous éditez un commentaire initial ou une entrée journal à inclure dans un fichier maître, terminez l'entrée en tapant `C-c C-c`.

`C-c C-c` Termine l'édition de commentaire normalement (`vc-finish-logentry`). Ceci termine l'enregistrement.

Pour annuler l'enregistrement, ne tapez **pas** `C-c C-c` dans ce tampon. Vous pouvez aller dans d'autres tampons et faire d'autres éditions. Aussi longtemps que vous n'essayez pas de faire enregistrer un autre fichier, l'entrée que vous étiez en train d'éditer reste dans son tampon, et vous pouvez revenir à ce tampon à tout moment pour terminer l'enregistrement.

Si vous modifiez plusieurs fichiers sources pour la même raison, il est souvent pratique de spécifier la même entrée journal pour tous ces fichiers. Pour faire cela, utilisez l'historique des entrées journal précédentes. Les commandes `M-n`, `M-p`, `M-s` et `M-r` pour faire cela fonctionnent exactement comme les commandes d'historique du mini-tampon (excepté que ces versions sont utilisées en dehors du mini-tampon).

Chaque fois que vous faites enregistrer un fichier, le tampon des entrées journal est placé en mode Journal VC, ce qui nécessite d'exécuter deux commandes crochet : `text-mode-hook` et `vc-log-mode-hook`. See Section 31.2.3 [Hooks], page 465.

14.7.4 Journal des Changements et VC

Si vous utilisez RCS pour un programme et maintenez aussi un fichier Journal des changements pour lui (see Section 22.14 [Change Log], page 299), vous pouvez générer des entrées du journal des changements automatiquement à partir des entrées journal du contrôle de version :

`C-x v a` Visite le fichier journal des changements du répertoire courant et, pour les fichiers déclarés de ce répertoire, crée de nouvelles entrées pour les versions enregistrées depuis l'entrée la plus récente dans le fichier journal des changements. (`vc-update-change-log`).

Cette commande fonctionne uniquement avec RCS et CVS, et non avec SCCS.

`C-u C-x v a`
Comme précédemment, mais trouve seulement les entrées pour le fichier du tampon courant.

`M-1 C-x v a`
Comme précédemment, mais trouve les entrées pour tous les fichiers couramment visités étant maintenus par contrôle de version. Ceci fonctionne uniquement avec RCS, et place toutes les entrées dans le journal pour le répertoire par défaut, qui peut ne pas être approprié.

Par exemple, supposez que la première ligne de '`ChangeLog`' soit daté du 10 Avril 1992, et que le seul enregistrement depuis ait été fait par Nathaniel Bowditch à '`rcs2log`' le 8 Mai 1992 avec comme texte de journal '`Ignorez les messages du journal commençant par '#'.`'. `C-x v a` visite alors '`ChangeLog`' et insére du texte tel que :

```
Fri May  8 21:45:00 1992  Nathaniel Bowditch  <nat@apn.org>

        * rcs2log:  Ignorez les messages du journal commençant par '#'.
```

Vous pouvez alors éditer la nouvelle entrée du journal des changements si vous le désirez.

Normalement, l'entrée journal pour le fichier '`foo`' est affiché '`* foo:` *text de l'entrée journal*'. Le '`:`' après '`foo`' est omis si le texte de l'entrée journal commence avec '`(`*functionname*`):`'. Par exemple, si l'entrée journal pour '`vc.el`' est '`(vc-do-command): Check call-process status.`', alors le texte dans '`ChangeLog`' sera :

```
Wed May  6 10:53:00 1992  Nathaniel Bowditch  <nat@apn.org>

        * vc.el (vc-do-command): Check call-process status.
```

Lorsque `C-x v a` ajoute plusieurs entrées au journal des changements en même temps, il regroupe les entrées journal si toutes sont enregistrées par le même auteur à la même date. Si les entrées journal de plusieurs fichiers ont toutes le même texte, il les regroupe dans une seule entrée. Par exemple, supposez que le dernier enregistrement ait les entrées journal suivantes :

- Pour '`vc.texinfo`': '`Fixe typos expansion.`'
- Pour '`vc.el`': '`N'appelle pas expand-file-name.`'
- Pour '`vc-hooks.el`': '`N'appelle pas expand-file-name.`'

Elles apparaissent ainsi dans '`ChangeLog`':

```
Wed Apr  1 08:57:59 1992  Nathaniel Bowditch  <nat@apn.org>

        * vc.texinfo: Fixe typos expansion.

        * vc.el, vc-hooks.el: N'appelle pas expand-file-name.
```

Normalement, `C-x v a` sépare les entrées journal par une ligne vierge, mais vous pouvez marquer plusieurs entrées journal pour qu'elles soient réunies (sans ligne vierge entre elles) en commençant le texte de chaque entrée journal avec une étiquette de la forme '`{`*nom*`}` '. L'étiquette elle-même n'est pas copiée dans '`ChangeLog`'. Par exemple, supposez que les entrées journal soient :

- Pour '`vc.texinfo`': '`{expand} Fixe typos expansion.`'
- Pour '`vc.el`': '`{expand} N'appelle pas expand-file-name.`'
- Pour '`vc-hooks.el`': '`{expand} N'appelle pas expand-file-name.`'

Le texte dans '`ChangeLog`' apparaît alors ainsi :

```
Wed Apr  1 08:57:59 1992  Nathaniel Bowditch  <nat@apn.org>

        * vc.texinfo: Fixe typos expansion.
        * vc.el, vc-hooks.el: N'appelle pas expand-file-name.
```

Une entrée journal dont le texte commence par '`#`' n'est pas copié dans '`ChangeLog`'. Par exemple, si vous corrigez simplement quelques fautes d'orthographe dans les commentaires, vous pouvez journaliser le changement avec '`#`' pour éviter de le mettre dans '`ChangeLog`'.

14.7.5 Examiner et Comparer d'Anciennes Versions

`C-u C-x C-q` *version* ⟨RET⟩
: Sélectionne la version *version* comme version du fichier de travail courant.

`C-x v ~` *version* ⟨RET⟩
: Examine la version *version* du fichier visité, dans son propre tampon.

`C-x v =`
: Compare le contenu du tampon courant avec la dernière version enregistrée du fichier.

`C-u C-x v =` *fichier* ⟨RET⟩ *version1* ⟨RET⟩ *version2* ⟨RET⟩
: Compare les deux version spécifiées de *fichier*.

`C-x v g`
: Affiche le résultat de la commande d'annotation de CVS en utilisant des couleurs.

Il existe deux moyens de travailler avec une ancienne version d'un fichier. Vous pouvez prendre l'ancienne version comme fichier de travail, par exemple si vous désirez reproduire un stage précédent du développement, ou si vous désirez créer une branche pour l'ancienne version. (see Section 14.7.6 [Branches], page 172). Pour faire cela, visitez le fichier et tapez `C-u C-x C-q` *version* ⟨RET⟩. (Ceci marche seulement avec RCS.)

Si vous désirez seulement examiner une ancienne version, sans changer de fichier de travail, visitez le fichier puis tapez `C-x v ~` *version* ⟨RET⟩ (`vc-version-other-window`). Ceci place le texte de la version *version* dans un fichier appelé '*nom_fichier*`.~`*version*`~`', et le visite dans son propre tampon, dans une fenêtre séparée.

Pour comparer deux versions d'un fichier, utilisez la commande `C-x v =` (`vc-diff`). `C-x v =` seul compare le contenu du tampon courant (en l'enregistrant dans un fichier si nécessaire) avec la dernière version enregistrée du fichier. `C-u C-x v =`, avec un argument numérique, lit un nom de fichier et deux numéros de version, puis compare ces versions du fichier spécifié.

Si vous donnez un nom de répertoire plutôt q'un nom de fichier de travail, cette commande compare les deux version spécifiées de tous les fichiers déclarés dans ce répertoire et ses sous-répertoires. Vous pouvez aussi spécifier un nom d'instantané (see Section 14.7.9 [Snapshots], page 175) à la place de l'un ou des deux numéros de version.

Vous pouvez spécifier une version enregistrée par son nombre ; une entrée vierge spécifie le contenu courant du fichier de travail (qui peut être différent de toutes les versions enregistrées).

Cette commande fonctionne en exécutant l'utilitaire `diff`, obtenant les options de la variable `diff-switches`. Elle affiche la sortie dans un tampon spécial dans une autre fenêtre. À l'inverse de la commande `M-x diff`, `C-x v =` n'essaie pas de localiser les changements entre l'ancienne et la nouvelle version. Ceci car normalement, les deux versions n'existent pas en tant que fichier lorsque vous les comparez ; elles existent seulement dans les enregistrements du fichier maître. See Section 14.9 [Comparing Files], page 181, pour plus d'informations sur `M-x diff`.

Pour des fichiers contrôlés par CVS, vous pouvez afficher le résultat de la commande d'annotation, en utilisant des couleurs pour améliorer l'apparence visuelle. Utilisez la commande `M-x vc-annotate` pour faire cela. Rouge veut dire nouveau, bleu veut dire ancien, et des couleurs intermédiaires indique des âges intermédiaires. Un argument préfixe *n* spécifie un facteur d'envergure pour l'échelle de temps ; il fait couvrir chaque couleur une période *n* fois plus longue.

14.7.6 Branches Multiples d'un Fichier

Une utilisation du contrôle de version est de maintenir plusieurs versions "courantes" d'un fichier. Par exemple, vous pouvez avoir plusieurs versions d'un programme dans lesquelles vous ajoutez graduellement diverses nouvelles caractéristiques non terminées. Chacune de ces lignes de développement indépendantes sont appelées des *branches*. Notez, cependant, que les branches sont seulement supportées par RCS.

La ligne de développement principale d'un fichier est habituellement appelée le *tronc*. Les versions du tronc sont en général numérotées 1.1, 1.2, 1.3, etc. À chacune de ces versions, vous pouvez démarrer une branche indépendante. Une branche démarrant à la version 1.2 aura un numéro de version 1.2.1.1. Les versions consécutives de cette branche seront numérotées 1.2.1.2, 1.2.1.3, 1.2.1.4, et ainsi de suite. Si une seconde branche démarre à la version 1.2, elle sera constituée des versions 1.2.2.1, 1.2.2.2, 1.2.2.3, et ainsi de suite.

Si vous omettez le composant final d'un numéro de version, ceci est appelé un *numéro de branche*. Il se réfère à la version existante la plus haute dans cette branche. Les branches dans l'exemple précédent ont des numéros de branche 1.2.1 et 1.2.2.

Une version qui est la dernière dans sa branche est appelée une version *de tête*.

14.7.6.1 Passer d'une Branche à une Autre

Pour passer d'une branche à une aute, tapez `C-u C-x C-q` et spécifiez le numéro de version que vous désirez sélectionner. Cette version est alors retirée *déverrouillée* (protégée en écriture), pour que vous puissiez l'examiner avant de la retirer vraiment. Passer d'une branche à une autre de cette manière est autorisé seulement si le fichier n'est pas verrouillé.

Vous pouvez omettre le numéro mineur de la version, donnant ainsi seulement le numéro de branche ; ceci vous donne la version la plus haute de la branche indiquée. Si vous tapez seulement `RET`, Emacs va à la version la plus haute dans le tronc.

Après être passé à une autre branche (incluant la branche principale), vous y restez pour les commandes VC suivantes, jusqu'à ce que vous passiez explicitement à une autre branche.

14.7.6.2 Créer de Nouvelles Branches

Pour créer une nouvelle branche depuis une version de tête (la dernière dans sa branche), sélectionnez d'abord cette version si nécessaire, verrouillez-la avec `C-x C-q`, et faites les changements que vous désirez. Ensuite, lorsque vous faites enregistrer vos changements, utilisez `C-u C-x C-q`. Ceci vous laisse spécifier le numéro de version pour la nouvelle version. Vous devez spécifier un numéro de branche adapté pour une branche démarrant à la version en cours. Par exemple, si la version courante est 2.5, le numéro de branche doit être 2.5.1, 2.5.2, et ainsi de suite, selon le nombre de branches existant déjà à cet endroit.

Pour créer une nouvelle branche à partir d'une ancienne version (une qui n'est plus la tête de la branche), sélectionnez d'abord cette version, puis verrouillez-la avec `C-x C-q`. Il vous sera demandé de confirmer, lorsque vous verrouillez l'ancienne version, que vous désirez réellement créer une nouvelle branche–si vous répondez non, il vous sera offert la chance de verrouiller plutôt la dernière version.

Faites alors vos changements et tapez `C-x C-q` de nouveau pour faire enregistrer une nouvelle version. Ceci crée automatiquement une nouvelle branche démarrant à la verrsion sélectionnée. Vous n'avez pas besoin de demander à créer une nouvelle branche, car c'est le seul moyen d'ajouter une nouvelle version à un point qui n'est pas la tête d'une branche.

Une fois que la branche est créée, vous y "restez". Cela veut dire que les retraits et enregistrements suivants créent de nouvelles versions dans cette branche. Pour quitter la branche, vous devez explicitement sélectionner une nouvelle version avec `C-u C-x C-q` pour la retirer.

14.7.6.3 Branchage Multi-Utilisateur

Il est parfois utile pour plusieurs utilisateurs de travailler simultanément sur différentes branches d'un fichier. Ceci est possible si vous créez plusieurs répertoires de sources. Chaque répertoire de sources doit avoir un lien appelé '`RCS`' pointant vers un répertoire commun de fichiers maîtres RCS. Alors chacun des répertoires de sources peut avoir son propre choix de versions retirées, mais tous partagent les mêmes enregistrements RCS.

Cette technique fonctionne correctement et automatiquement, à condition que les fichiers sources contiennent des entêtes de version RCS (see Section 14.7.10 [Version Headers], page 176). Les entêtes permettent à Emacs

de savoir exactement, à tout moment, quel numéro de version est présent dans le fichier de travail.

Si les fichiers n'ont pas d'entête de version, vous devez à la place indiquer explicitement à Emacs et à chaque session dans quelle branche vous travaillez. Pour faire cela, trouvez d'abord le fichier, puis tapez `C-u C-x C-q` et spécifiez le numéro de branche correct. Ceci assure qu'Emacs connaît la branche qu'il utilise durant une session d'édition particulière.

14.7.7 Commandes de Status de VC

Pour voir le status détaillé du contrôle de vesion et l'historique d'un fichier, tapez `C-x v l` (`vc-print-log`). Cette commande affiche l'historique des changements du fichier courant, le texte des entrées journal inclus. La sortie apparraît dans une fenêtre séparée.

Lorsque vous travaillez sur un gros programme, il est souvent nécessaire de trouver tous les fichiers couramment verrouillés, ou tous les fichiers maintenus par contrôle de version. Vous pouvez utiliser `C-x v d` (`vc-directory`) pour voir tous les fichiers verrouillés dans ou sous un certain répertoire. Ceci inclut tous les fichiers verrouillés par n'importe quel utilisateur. `C-u C-x v d` liste tous les fichiers dans ou sous le répertoire spécifié qui sont maintenus par contrôle de version.

La liste des fichiers est affichée dans un tampon qui utilise un mode Dired étendu. Les noms des utilisateurs possédant un verrou sur divers fichiers sont affichés (entre parenthèses) à la place du propriétaire et du groupe. (Avec CVS, un status plus détaillé est affiché pour chaque fichier.) Toutes les commandes Dired ordinaires fonctionnent dans ce tampon. La plupart des commandes interactives de VC fonctionnent aussi, et s'appliquent au nom de fichier de la ligne courante.

La commande `C-x v v` (`vc-next-action`), lorsqu'elle est utilisée dans le tampon Dired étendu, opère sur tous les fichiers marqués (ou le fichier de la ligne courante). Si elle opère sur plus d'un fichier, elle gère chacun des fichiers selon son état courant ; ainsi, elle peut retirer un fichier et faire enregistrer un autre (car il est déjà retiré). Si elle doit faire enregistrer plusieurs fichiers, elle lit une seule entrée journal, et utilise ce texte pour tous les fichiers à faire enregistrer. Ceci peut être pratique pour déclarer ou faire enregistrer plusieurs fichiers en même temps, au cours d'un même changement.

14.7.8 Renommer les Fichiers de Travail et Fichiers Maîtres

de VC

Lorsque vous renommez un fichier déclaré, vous devez aussi renommer son fichier maître pour obtenir un résultat correct. Utilisez `vc-rename-file` pour renommer le fichier source comme vous le désirez, et renommer le fichier maître correctement. Elle met aussi à jour tous les instantanés (see Section 14.7.9 [Snapshots], page 175) mentionnant ce fichier, pour qu'ils utilisent le nouveau nom ; en dépit de ça, l'instantané ainsi modifié peut ne pas fonctionner totalement (see Section 14.7.9.2 [Snapshot Caveats], page 176).

Vous ne pouvez pas utiliser `vc-rename-file` avec un fichier étant verrouillé par quelqu'un d'autre.

14.7.9 Instantanés

Un *instantané* est un jeu nommé de versions de fichiers (une pour chaque fichier déclaré) que vous pouvez traiter comme une unité. Un type important d'instantané est une *release*, une version (théoriquement) stable du système qui est prête à être distribuée à des utilisateurs.

14.7.9.1 Créer et Utiliser des Instantanés

Il existe deux commandes élémentaires pour les instantanés ; une crée un instantané portant un nom donné, l'autre retire un instantané d'après son nom.

`C-x v s` *nom* ⟨RET⟩
: Définit les dernières versions enregistrées de tous les fichiers déclarés dans ou sous le répertoire courant comme instantané appelé *nom* (`vc-create-snapshot`).

`C-x v r` *nom* ⟨RET⟩
: Retire tous les fichiers déclarés dans ou sous le répertoire courant en utilisant les versions correspondant à l'instantané *nom* (`vc-retrieve-snapshot`).

  Cette commande retourne une erreur si des fichiers dans ou sous le répertoire courant sont verrouillés, sans rien changer ; ceci permet d'éviter de perdre du travail en cours.

Un instantané utilise une quantité très petite de ressources—juste assez pour sauvegarder une liste de fichiers et leurs versions correspondant à l'instantané. Ainsi, n'hésitez pas à créer des instantanés lorsqu'ils peuvent être utiles.

Vous pouvez donner un nom d'instantané comme argument à `C-x v =` ou `C-x v ~` (see Section 14.7.5 [Old Versions], page 171). Ainsi, vous pouvez utiliser ce moyen pour comparer un instantané avec les fichiers courants, ou deux instantanés entre eux, ou encore un instantané et une certaine version.

14.7.9.2 Avertissemennts sur les Instantanés

Les instantanés de VC sont modelés dans le support de configurations nommées de RCS. Ils utilisent les possibilités natives de RCS pour cela, ainsi sous VC les instantanés utilisant RCS sont visibles même si vous n'utlisez pas VC pour les créer.

Pour SCCS, VC implémente lui-même les instantanés. Les fichiers qu'il utilise contiennent des triplets nom/fichier/numéro de version. Ces instantanés sont visibles seulement sous VC.

Un instantané est un jeu de versions enregistrées. Prenez donc garde que tous les fichiers sont enregistrés et déverrouillés lorsque vous créez un instantané.

Le renommage et la suppression de fichiers peut créer quelques difficultés avec les instantanés. Ceci n'est pas un problème spécifique à VC, mais aux systèmes de contrôle de version en général qui n'a pas été trop bien résolu à cette heure.

Si vous renommez un fichier déclaré, vous devez renommer son fichier maître en même temps (la commande `vc-rename-file` le fait automatiquement). Si vous utilisez SCCS, vous devez aussi mettre à jour les enregistrements de l'instantané, pour mentionner le fichier parr son nouveau nom (`vc-rename-file` le fait, aussi). Un ancien instantané qui se réfère à un fichier maître qui n'existe plus sous le nom enregistré est invalide ; VC ne peut plus le retrouver. Il serait en dehors de la portée de manuel d'expliquer comment mettre à jour manuellement les instantanés pour RCS et SCCS.

L'utilisation de `vc-rename-file` permet de garder l'instantané valide pour les retraits, mais il ne résout pas tous les problèmes. Par exemple, certains fichiers d'un programme se réfèrent probablement à d'autres par leur nom. Le makefile mentionne probablement le fichier que vous avez renommé. Si vous retirez un ancien instantané, le fichier renommé est retiré sous son nouveau nom, qui n'est pas le nom que le makefile attend. Le programme ne fonctionnera donc pas comme il le faut.

14.7.10 Insérer des Entêtes de Contrôle de Version

Il est parfois utile de placer des chaînes d'identification de version directement dans les fichiers de travail. Certaines chaînes spéciales appelées *entêtes de version* sont remplacées dans chaque version successive par le numéro de version.

Si vous utilisez RCS, et que des entêtes de version sont présentes dans vos fichiers de travail, Emacs peut utiliser celles-ci pour déterminer la version courante et l'état des verrous des fichiers. Ceci est plus sûr que de se référer aux fichiers maîtres, ce qui est fait lorsqu'il n'y a pas d'entêtes de version. Notez que dans un environnement multi-branches, les entêtes de version

sont nécessaires pour que VC fonctionne correctement (see Section 14.7.6.3 [Multi-User Branching], page 173).

La recherche des entêtes de version est contrôlée par la variable `vc-consult-headers`. Si elle est non `nil`, Emacs recherche les entêtes pour déterminer le numéro de la version que vous éditez. Mettre cette variable à `nil` stoppe cette caractéristique.

Vous pouvez utiliser la commande `C-x v h` (`vc-insert-headers`) pour insérer une chaîne d'entête convenable.

`C-x v h` Insère des entêtes dans un fichier à utiliser avec votre système de contrôle de version.

La chaîne d'entête par défaut est '`$Id$`' pour RCS et '`%W%`' pour SCCS. Vous pouvez spécifier d'autres entêtes à insérer en définissant la variable `vc-header-alist`. Sa valeur est une liste d'éléments de la forme (*programme* `.` *chaîne*) où *programme* est `RCS` ou `SCCS` et *chaîne* est la chaîne à utiliser.

Plutôt qu'une seule chaîne, vous pouvez spécifier une liste de chaînes ; ainsi chaque chaîne de la liste est insérée comme entête à part entière sur sa propre ligne.

Il est parfois nécessaire d'utiliser des barres obliques inverses "superflues" pour écrire les chaînes que vous placez dans cette variable. Ceci pour empêcher la chaîne d'être interprétée comme une entête elle-même si le fichier Emacs Lisp la contenant est maintenue par contrôle de version.

Chaque entête est insérée entourée de tabs, et entre des délimiteurs de commentaire, sur une nouvelle ligne au début du tampon. Normalement, les chaînes de début et de fin de commentaire du mode courant sont utilisées, mais pour certains modes, il existe des délimiteurs de commentaires dédiés à cet usage ; la variable `vc-comment-alist` spécifie ces délimiteurs. Chaque élément de la liste a la forme (*mode début fin*).

La variable `vc-static-header-alist` spécifie des chaînes supplémentaires à rajouter basées sur le nom du tampon. Sa valeur doit être une liste d'éléments de la forme (*regexp* `.` *format*). Lorsque *regexp* correspond au nom du tampon, *format* est inséré dans l'entête. Une ligne d'entête est insérée pour chaque élément correspondant au nom du tampon, et pour chaque chaîne spécifiée par `vc-header-alist`. La ligne d'entête est créée en passant la chaîne de `vc-header-alist` dans le format pris dans l'élément. La valeur par défaut pour `vc-static-header-alist` est :

```
(("\\.c$" .
  "\n#ifndef lint\nstatic char vcid[] = \"\%s\";\n\
#endif /* lint */\n"))
```

Ce qui spécifie l'insertion d'un texte de la forme :

```
#ifndef lint
static char vcid[] = "chaîne";
#endif /* lint */
```

Notez que le texte précédent commence par une ligne vierge.

Si vous utilisez plus d'une entête de version dans un fichier, placez-les à côté dans le fichier. Le mécanisme de `revert-buffer` qui préserve les marques peut ne pas tenir compte des marqueurs placés entre deux entêtes de version.

14.7.11 Personnaliser VC

Il y a plusieurs moyens de personnaliser VC. Les variables qui contrôlent son comportement tombent dans trois catégories, décrites dans les sections suivantes.

14.7.11.1 Gestion des Fichiers de Travail de VC

Emacs ne crée normalement pas de fichier archive pour les fichiers source qui sont maintenus par contrôle de version. Si vous désirez créer des fichiers archives même pour ces fichiers, mettez la variable `vc-make-backup-files` à une valeur non`nil`.

Normalement, le fichier archive existe toujours, qu'il soit verrouilléé ou non. Si vous mettez `vc-keep-workfiles` à `nil`, alors faire enregistrer une nouvelle version avec `C-x C-q` efface le fichier de travail ; mais tout essai d'ouvrir le fichier avec Emacs crée de nouveau ce fichier. (Avec CVS, les fichiers de travail sont toujours conservés.)

L'édition d'un fichier sous contrôle de version à travers un lien symbolique peut être dangereux. Elle outrepasse le système de contrôle de version—il est possible d'éditer le fichier sans avoir à le retirer, et de ne pas pouvoir en faire enregistrer les changements. De plus, vos changements peuvent écraser ceux d'un autre utilisateur. Pour vous protéger de cela, VC vérifie tous les liens symboliques que vous visitez, pour voir s'il pointe vers un fichier sous contrôle de version.

La variable `vc-follow-symlinks` contrôle ce qu'il faut faire lorsqu'un lien symbolique pointe sur un fichier sous contrôle de version. Si elle est `nil`, VC affiche seulement un message d'avertissement. Si elle est `t`, VC suit automatiquement le lien, et visite plutôt le fichier réel, en vous avertissant dans la zone de répercussion. Si la valeur est `ask` (par défaut), VC vous demande à chaque fois s'il faut suivre le lien.

14.7.11.2 Recouvrement du Status de VC

Lorsqu'il déduit l'état verrouillé/déverrouillé d'un fichier, VC recherche d'abord une chaîne d'entête de version RCS dans le fichier (see Section 14.7.10 [Version Headers], page 176). S'il n'y a pas de chaîne entête, ou

si vous utilisez SCCS, VC examine normalement les permissions de fichier du fichier de travail ; ceci est rapide. Mais dans certaines situations les permissions de fichier ne peuvent être crédibles. Dans ce cas le fichier maître doit être consulté, ce qui est assez coûteux. De plus le fichier maître peut seulement vous dire *si* un verrou est posé sur le fichier, mais non si votre fichier de travail contient cette version verrouillée.

Vous pouvez indiquer à VC de ne pas utiliser les entêtes de version pour déterminer l'état de verrouillage en mettant `vc-consult-headers` à `nil`. VC utilise alors toujours les permissions de fichier (si elles sont crédibles), ou bien vérifie le fichier maître.

Vous pouvez spécifier le critère de crédibilité des permissions de fichier en définissant la variable `vc-mistrust-permissions`. Sa valeur peut être `t` (ne jamais croire les permissions de fichier et examiner le fichier maître), `nil` (toujours croire les permissions de fichier), ou une fonction avec un argument qui prend la décision. L'argument est le nom de répertoire du sous-répertoire '`RCS`', '`CVS`' ou '`SCCS`'. Une valeur non `nil` retournée par la fonction indique de ne pas croire les permissions de fichier. Si vous trouvez que les permissions de fichier des fichiers de travail sont modifiées de façon incorrecte, mettez `vc-mistrust-permissions` à `t`. VC vérifie toujours alors le fichier maître pour déterminer le status d'un fichier.

14.7.11.3 Exécution de Commandes VC

Si `vc-suppress-confirm` est non `nil`, alors `C-x C-q` et `C-x v i` peuvent sauvegarder le tampon courant sans demander, et `C-x v u` opère aussi sans demander confirmation. (Cette variable n'affecte pas `C-x v c`; cette opération est si radicale qu'elle doit toujours demander une confirmation.)

Le mode VC fait le plus gros de son travail en exécutant les commandes shell de RCS, CVS et SCCS. Si `vc-command-messages` est non `nil`, VC affiche des messages indiquant quelles commandes shell il exécute, et un message additionnel lorsque la commande finit.

Vous pouvez spécifier des répertoires supplémentaires pour la recherche des programmes de contrôle de version en définissant la variable `vc-path`. Ces épertoires sont parcourus avant le chemin de recherche usuel. Mais les bons fichiers sont d'habitude trouvés automatiquement.

14.8 Répertoires de Fichiers

Le système de fichiers groupe les fichiers dans des *répertoires*. La *liste d'un répertoire* est la liste de tous les fichiers contenus dans un répertoire. Emacs fournit des commandes pour créer et supprimer des répertoires, et pour affichier des listes de répertoires dans le format court (noms de fichiers seulement) ou au format long (tailles, dates, et auteurs inclus). Emacs a

aussi un navigateur de répertoires appelé Dired ; see Chapter 28 [Dired], page 387.

`C-x C-d` *rép-ou-motif* ⟨RET⟩
: Affiche la liste au format court d'un répertoire (`list-directory`).

`C-u C-x C-d` *rép-ou-motif* ⟨RET⟩
: Affiche la liste au format long d'un répertoire

`M-x make-directory` ⟨RET⟩ *nom-rép* ⟨RET⟩
: Crée un nouveau répertoire appelé *nom-rép*.

`M-x delete-directory` ⟨RET⟩ *nom-rép* ⟨RET⟩
: Supprime le répertoire appelé *nom-rép*. Il doit être vide, ou une erreur survient.

La commande pour afficher la liste d'un répertoire est `C-x C-d` (`list-directory`). Elle lit en utilisant le mini-tampon un nom de fichier qui est soit un répertoire à lister soit un motif contenant des caractères génériques décrivant les fichierrs à lister. Par exemple,

```
C-x C-d /u2/emacs/etc ⟨RET⟩
```

liste tous les fichiers du répertoire '`/u2/emacs/etc`'. Voici un exemple de spécification d'un motif de nom de fichier :

```
C-x C-d /u2/emacs/src/*.c ⟨RET⟩
```

Normalement, `C-x C-d` affiche une liste courte du répertoire contenant seulement des noms de fichiers. Un argument numérique (quelle que soit la valeur) indique d'afficher une liste longue incluant les tailles, dates et auteurs (comme '`ls -l`').

Le texte de la liste d'un répertoire est obtenu en exécutant `ls` dans un processus fils. Deux variables Emacs contrôlent les options passées à `ls` : `list-directory-brief-switches` est une chaîne donnant les options à utiliser pour une liste courte (`"-CF"` par défaut), et `list-directory-verbose-switches` est une chaîne donnant les options à utiliser pour une liste longue (`"-l"` par défaut).

14.9 Comparer des Fichiers

La commande `M-x diff` compare deux fichiers, affichant les différences dans un tampon Emacs appelé '`*Diff*`'. Elle fonctionne en exécutant le programme `diff`, utilisant les options obtenues de la variable `diff-switches`, dont la valeur doit être une chaîne.

Le tampon '`*Diff*`' a le mode Compilation comme mode majeur, et vous pouvez utiliser `C-x ‘` pour visiter les locations successives des changements dans les deux fichiers sources. Vous pouvez aussi vous déplacer sur un ensemble de changements particuliers et taper ⟨RET⟩ ou `C-c C-c`, ou cliquer

`Mouse-2` dessus, pour vous déplacer sur la source correspondante. Vous pouvez aussi utiliser les autres commandes spéciales du mode Compilation : ⟨SPC⟩ et ⟨DEL⟩ pour faire défiler, et `M-p` et `M-n` pour déplacer le curseur. See Section 23.1 [Compilation], page 331.

La commande `M-x diff-backup` compare un fichier spécifié avec son archive la plus récente. Si vous spécifiez le nom d'un fichier archive, `diff-backup` le compare avec le fichier source dont il est l'archive.

La commande `M-x compare-windows` compare le texte de la fenêtre courante avec le texte de la fenêtre suivante. La comparaison commence au point dans chaque fenêtre, et chaque position de départ est ajoutée à la pile des marques du tampon correspondant. Alors le point avance dans chaque fenêtre, un caractère à la fois, jusqu'à ce qu'une différence entre les deux fenêtres soit atteinte. La commande est alors terminée. Pour plus d'informations sur les fenêtres d'Emacs, Chapter 16 [Windows], page 195.

Avec un argument numérique, `compare-windows` ignore les changements d'espaces. Si la variable `compare-ignore-case` est non `nil`, elle ignore aussi les différences de casse.

Voir aussi Section 22.18 [Emerge], page 311, pour des moyens de fusionner deux fichiers similaires.

14.10 Opérations Diverses sur les Fichiers

Emacs a des commandes pour exécuter d'autres opérations sur les fichiers. Toutes opérent sur un fichier ; elles n'acceptent pas de caractères génériques dans les noms de fichiers.

`M-x view-file` vous permet de parcourir ou lire un fichier par écrans successifs. Elle lit un nom de fichier en argument eu utilisant le mini-tampon. Après avoir chargé le fichier dans un tampon d'Emacs, `view-file` en affiche le début. Vous pouvez alors taper ⟨SPC⟩ pour le faire défiler en avant d'une fenêtre entière, ou ⟨DEL⟩ pour le faire défiler en arrière. Diverses autres commandes sont fournies pour vous déplacer dans le fichier, mais aucune pour le modifier ; tapez `?` pour une liste de ces commandes. Elles sont pour la plupart les mêmes que les commandes de déplacement de curseur habituelles d'Emacs. Pour sortir de la visualisation, tapez `q`. Les commandes de visualisation sont définies par un mode majeur spécial appelé mode Visualisation.

Une commande similaire, `M-x view-buffer`, visualise un tampon déjà présent dans Emacs. See Section 15.3 [Misc Buffer], page 187.

`M-x insert-file` insére une copie du contenu du fichier spécifié au point dans le tampon courant, laissant le point à sa place avant le contenu inséré et plaçant la marque après le contenu.

`M-x write-region` est l'inverse de `M-x insert-file` ; elle copie le contenu de la région dans le fichier spécifié. `M-x append-to-file` ajoute le texte

de la région à la fin du fichier spécifié. See Section 9.3 [Accumulating Text], page 92.

`M-x delete-file` supprime le fichier spécifié, comme la commande `rm` dans un shell. Si vous supprimez plusieurs fichiers d'un même répertoire, il peut être plus approprié d'utiliser Dired (see Chapter 28 [Dired], page 387).

`M-x rename-file` lit deux noms de fichiers *ancien* et *nouveau* en utilisant le mini-tampon, puis renomme le fichier *ancien* en *nouveau*. Si un fichier appelé *nouveau* existe déjà, vous devez confirmer avec `yes` ou le fichier n'est pas renommé ; autrement le contenu du fichier appelé *nouveau* serait perdu. Si *ancien* et *nouveau* sont dans des systèmes de fichiers différents, le fichier *ancien* est copié puis supprimé.

La commande similaire `M-x add-name-to-file` est utilisée pour ajouter un nom à un fichier existant sans supprimer son ancien nom. Le nouveau nom doit rester sur le même système de fichiers que le nom du fichier.

`M-x copy-file` lit le fichier *ancien* et écrit un nouveau fichier appelé *nouveau* avec le même contenu. La confirmation est nécessaire si un fichier appelé *nouveau* existe déjà, car la copie a pour conséquence d'écraser l'ancien contenu du fichier *nouveau*.

`M-x make-symbolic-link` lit deux noms de fichiers *cible* et *lien*, puis crée un lien symbolique appelé *lien* et pointant vers *cible*. L'effet est que chaque future ouverture du fichier *lien* ouvrira le fichier appelé *cible* au moment où l'ouverture est faite, ou renverra une erreur si le nom *cible* n'est pas utilisé à ce moment-là. Cette commande ne développe pas l'argument *cible*, ce qui vous permet de spécifier un chemin relatif comme cible du lien.

Une confirmation est nécessaire lorsque vous créez le lien si *lien* est déjà utilisé. Notez que tous les systèmes ne supportent pas les liens symboliques.

14.11 Accéder à des Fichiers Compressés

Emacs est fourni avec une bibliothèque qui peut décompresser automatiquement des fichiers compressés lorsque vous les visitez, et les recompresser automatiquement si vous les modifiez et les enregistrez. Pour utiliser cette fonctionnalité, tapez la commande `M-x auto-compression-mode`.

Lors de la compression automatique (qui implique la décompression automatique), Emacs reconnaît un fichier compressé par son nom de fichier. Les noms de fichiers finissant par '`.gz`' indique un fihcier compressé avec `gzip`. D'autres terminaisons indiquent d'autres programmes de compression.

La décompression et la compression automatiques s'appliquent à toutes les opérations dans lesquelles Emacs utilise le contenu d'un fichier. Ceci inclut la visite, la sauvegarde, l'insertion dans un tampon, le charger, et le compiler.

14.12 Fichiers Distants

Vous pouvez vous référer à des fichiers sur d'autres machines en utilisant une syntaxe particulièrre de noms de fichiers :

/machine:fichier
/utilisateur@machine:fichier

Lorsque vous faîtes cela, Emacs utilise le programme FTP pour lire et écrire les fichiers sur la machine spécifiée. Il se connecte par FTP en utilisant votre nom d'utilisateur ou le nom *utilisateur*. Il peut vous demander un mot de passe de temps en temps ; il est utilisé pour la connexion à *machine*.

Vous pouvez arrêter complètement la fonctionnalité de noms de fichiers FTP en mettant la variable `file-name-handler-alist` à `nil`.

Le paquetage Emacs qui implémente l'aacès aux fichiers FTP est appelé `ange-ftp`. Cependant, vous n'avez pas besoin de vous rapeller ce nom pour utiliser cette fonctionnalité.

14.13 Noms de Fichier Cités

Vous pouvez *citer* un nom de fichier absolu pour empêcher à des caractères spéciaux d'avoir leurs effets particuliers. La manière de faire cela est d'ajouter '`/:`' au début du nom de fichier.

Par exemple, vous pouvez citer le nom d'un fichier local qui semble être distant, pour empêcher qu'il soit traité comme un nom de fichier distant. Ainsi, si vous avez un répertoire nommé '`/foo:`' et un fichier '`bar`' dans ce répertoire, vous pouvez vous référer à ce fichier sous Emacs avec '`/:/foo:/bar`'.

'`/:`' peut aussi empêcher '`~`' d'être traité comme le caractère spécial indiquant le répertoire personnel de l'utilisateur. Par exemple, '`/:/tmp/~hack`' se réfère à un fichier dont le nom est '`~hack`' dans le répertoire '`/tmp`'.

De même, citer avec '`/:`' est un moyen d'entrer dans le mini-tampon un nom de fichier contenant '`$`'. Cependant, '`/:`' doit être au début du tampon pour citer '`$`'.

15 Utiliser Plusieurs Tampons

Le texte que vous éditez sous Emacs réside dans un objet appelé un *tampon*. Chaque fois que vous visitez un fichier, un tampon est créé pour contenir le texte du fichier. Chaque fois que vous invoquez Dired, un tampon est créé pour contenir la liste du répertoire. Si vous envoyez un message avec `C-x m`, un tampon appelé '`*mail*`' est utilisé pour contenir le texte du message. Lorsque vous demandez la documentation d'une commande, elle apparaît dans un tampon appelé '`*Help*`'.

À tout moment, un et un seul tampon est *sélectionné*. Il esat aussi appelé le *tampon courant*. Nous disons fréquemment qu'une commande opère sur "le tampon", comme s'il n'y en avait qu'un seul ; mais cela veut réellement dire que la commande opère sur le tampon sélectionné (ce que font la plupart des commandes).

Lorsqu'Emacs a plusieurs fenêtres, chaque fenêtre affiche un certain tampon, mais à tout moment une seule fenêtre est sélectionnée et le tampon qu'elle contient est le tampon sélectionné. La ligne de mode de chaque fenêtre affiche le nom du tampon que la fenêtre affiche (see Chapter 16 [Windows], page 195).

Chaque tampon a un nom, qui peut être de longueur quelconque, et vous pouvez sélectionner un tampon en donnant son nom. La plupart des tampons sont créés en visitant des fichiers, et leurs noms sont dérivés des noms des fichiers visités. Vous pouvez aussi créer un tampon vide en lui donnant le nom que vous désirez. Au démarrage, Emacs contient un tampon appelé '`*scratch*`' qui peut être utilisé pour évaluer des expressions Lisp dans Emacs. La distinction des minuscules et des majuscules est faite dans les noms de tampons.

Chaque tampon enregistre individuellement quel fichier il visite, s'il est modifié, et quel mode majeur et modes mineurs y sont en effet (see Chapter 19 [Major Modes], page 235). Toute variable Emacs peut être rendue *locale* à un tampon particulier, c'est-à-dire que sa valeur dans ce tampon peut être différente de sa valeur dans d'autres tampons. See Section 31.2.4 [Locals], page 466.

15.1 Créer et Sélectionner des Tampons

`C-x b` *tampon* ⟨RET⟩
: Sélectionne ou crée un tampon appelé *tampon* (`switch-to-buffer`).

`C-x 4 b` *tampon* ⟨RET⟩
: Similaire, mais sélectionne *tampon* dans une autre fenêtre (`switch-to-buffer-other-window`).

`C-x 5 b` *tampon* ⟨RET⟩
: Similaire, mais sélectionne *tampon* dans un autre cadre (`switch-to-buffer-other-frame`).

Pour sélectionner un tampon appelé *nomtampon*, tapez `C-x b` *nomtampon* ⟨RET⟩. Ceci exécute la commande `switch-to-buffer` avec l'argument *nomtampon*. Vous pouvez utiliser la complétion ou une abbréviation pour le nom de tampon (see Section 5.3 [Completion], page 57). Un argument vide à `C-x b` spécifie le tampon le plus récemment sélectionné n'étant affiché dans aucune fenêtre.

La plupart des tampons sont créés en visitant des fichiers, ou par des commandes Emacs qui veulent afficher du texte, mais vous pouvez aussi créer explicitement un tampon en tapant `C-x b` *nomtampon* ⟨RET⟩. Ceci créé un nouveau tampon vide ne visitant aucun fichier, et le séletionne pour l'édition. De tels tampons sont utilisés comme bloc-notes. Si vous essayez d'en sauvegarder un, il vous est demandé un nom de fichier à utiliser pour le sauvegarder. Le mode majeur du nouveau tampon est déterminé par la valeur de `default-major-mode` (see Chapter 19 [Major Modes], page 235).

Notez que `C-x C-f`, comme toute autre commande pour visiter un fichier, peut aussi être utilisée pour passer à un tampon visitant un fichier existant. See Section 14.2 [Visiting], page 145.

Emacs utilise des noms de tampons commençant par un espace pour un usage interne. Il traite ces tampons spécialement—par exemple, ils n'enregistrent pas par défaut d'informations d'annulation. Il est préférable que vous n'utilisiez pas vous-même de tels tampons.

15.2 Liste des Tampons Existants

`C-x C-b` Liste les tampons existants (`list-buffers`).

Pour afficher une liste de tous les tampons existants, tapez `C-x C-b`. Chaque ligne de la liste montre un nom de tampon, son mode majeur et le fichier visité. Les tampons sont listés dans l'ordre où ils ont été sélectionnés ; les tampons les plus récemment sélectionnés sont affichés en premier.

'`*`' au début d'une ligne indique que le tampon est "modifié". Si plusieurs tampons sont modifiés, il peut être temps d'en enregistrer avec `C-x s` (see Section 14.3 [Saving], page 148). '`%`' indique un tampon en lecture seule. '`.`' marque le tampon sélectionné. Voici un exemple d'une liste de tampons :

```
 MR Buffer          Size  Mode           File
 -- ------          ----  ----           ----
.*  emacs.tex       383402 Texinfo       /u2/emacs/man/emacs.tex
    *Help*          1287  Fundamental
    files.el        23076 Emacs-Lisp     /u2/emacs/lisp/files.el
 %  RMAIL           64042 RMAIL          /u/rms/RMAIL
```

```
*% man            747   Dired          /u2/emacs/man/
   net.emacs      343885 Fundamental    /u/rms/net.emacs
   fileio.c       27691 C               /u2/emacs/src/fileio.c
   NEWS           67340 Text            /u2/emacs/etc/NEWS
   *scratch*    0  Lisp Interaction
```

Notez que le tampon '`*Help*`' a été créé par une requête d'aide ; il ne visite aucun fichier. Le tampon `man` a été créé par Dired dans le répertoire '`/u2/emacs/man/`'. Vous pouvez lister seulement les tampons qui visitent des fichiers en donnant à la commande un préfixe ; c'est-à-dire en tapant `C-u C-x C-b`.

15.3 Opérations Diverses sur les Tampons

`C-x C-q` Change le status de lecture seule du tampon (`vc-toggle-read-only`).

`M-x rename-buffer` ⟨RET⟩ *nom* ⟨RET⟩
Change le nom du tampon courant.

`M-x rename-uniquely`
Renomme le tampon courant en ajoutant '<*nombre*>' à la fin.

`M-x view-buffer` ⟨RET⟩ *tampon* ⟨RET⟩
Parcours le tampon *tampon*.

Un tampon peut être en *lecture seule*, ce qui veut dire que les commandes qui changent sont contenu ne sont pas autorisées. La ligne de mode indique un tampon en lecture seule avec '`%%`' ou '`%*`' près de la marge gauche. Les tampons en lecture seule sont habituellement créés par des sous-systèmes comme Dired et Rmail qui ont des commandes spéciales pour opérer sur le texte ; et aussi en visitant un fichier dont le contrôle d'accès indique que vous ne pouvez pas le modifier.

Si vous désirez faire des changements sur un tampon en lecture seule, utilisez la commande `C-x C-q` (`vc-toggle-read-only`). Elle rend modifiable un tampon en lecture seule, et inversement. Dans la plupart des cas, elle fonctionne en définissant la variable `buffer-read-only`, qui a une valeur locale dans chaque tampon et rend le tampon en lecture seule si sa valeur est non `nil`. Si le fichier est maintenu par contrôle de version, `C-x C-q` fonctionne à travers le système de contrôle de version pour changer le status de lecture seule du fichier autant que du tampon. See Section 14.7 [Version Control], page 161.

`M-x rename-buffer` change le nom du tampon courant. Spécifiez le nouveau nom dans le mini-tampon. Il n'y a pas de valeur par défaut. Si vous spécifiez un nom utilisé par un autre tampon, une erreur survient et le renommage n'est pas fait.

`M-x rename-uniquely` renomme le tampon courant avec un nom similaire, où un suffixe numérique est ajouté pour rendre le nom différent et unique. Cette commande ne nécessite pas d'argument. Elle est utile pour créer plusieurs tampons de shell : si vous renommez le tampon '`*Shell*`', puis faites `M-x shell` de nouveau, un nouveau tampon de shell est créé, appelé '`*Shell*`' ; pendant ce temps, l'ancien tampon de shell continue d'exister sous son nouveau nom. Cette méthode est aussi valable pour les tampons de mail, de compilation, et la plupart des fonctionnalités d'Emacs qui créent des tampons spéciaux avec des noms particuliers.

`M-x view-buffer` est similaire à `M-x view-file` (see Section 14.10 [Misc File Ops], page 181) excepté qu'il examine un tampon d'Emacs déjà existant. Le mode Visualisation fournit des commandes pour se déplacer dans le tampon mais pas pour le modifier. Lorsque vous quittez le mode Visualisation avec `e`, le tampon et la valeur du point qui résultent de votre lecture restent effectifs.

Les commandes `M-x append-to-buffer` et `M-x insert-buffer` peuvent être utilisées pour copier du texte d'un tampon vers un autre. See Section 9.3 [Accumulating Text], page 92.

15.4 Destruction de Tampons

Si vous prolongez une session Emacs sur une longue période de temps, vous pouvez accumuler un grand nombre de tampons. Vous pouvez alors trouver utile de *détruire* les tampons dont vous n'avez plus besoin. Sur la plupart des systèmes d'exploitation, la destruction d'un tampon rend l'espace qu'il utilisait au système pour que d'autres programmes puissent l'utiliser. Voici quelques commandes pour détruire des tampons :

`C-x k` *nomtampon* ⟨RET⟩
: Détruit le tampon *nomtampon* (`kill-buffer`).

`M-x kill-some-buffers`
: Propose de détruire tous les tampons, un par un.

`C-x k` (`kill-buffer`) détruit un tampon, dont vous spécifiez le nom dans le mini-tampon. L'action par défaut, utilisée si vous tapez simplement ⟨RET⟩ dans le mini-tampon, est de détruire le tampon courant. Si vous détruisez le tampon courant, un aute tampon est sélectionné ; celui qui a été sélectionné le plus récemment et qui n'apparaît à ce moment dans aucune fenêtre. Si vous demandez de détruire un tampon visitant un fichier qui est modifié (c'est-à-dire qui a des changements non sauvegardés), vous devez confirmer avec `yes` avant que le tampon soit détruit.

La commande `M-x kill-some-buffers` vous interroge pour chaque tampon, un après l'autre. Une réponse `y` détruit le tampon. Détruire le tampon courant ou un tampon contenant des modifications non sauvegardées

sélectionne un nouveau tampon ou vous edmande confirmation comme le fait `kill-buffer`.

La fonctionnalité du menu tampons (see Section 15.5 [Several Buffers], page 189) est aussi utile pour détruire plusieurs tampons.

Si vous désirez exécuter une tâche spéciale chaque fois qu'un tampon est détruit, vous pouvez ajouter des fonctions crochet au crochet `kill-buffer-hook` (see Section 31.2.3 [Hooks], page 465).

Si vous exécutez une session Emacs pour une période de plusieurs jours, comme beaucoup de gens font, il peut s'encombrer de tampons que vous avez utilisé plusieurs jours auparavant. La commande `M-x clean-buffer-list` est un moyen pratique de les purger ; elle détruit tous les tampons non modifiés que vous n'avez pas utilisé durant une longue période. Un tampon ordinaire est détruit s'il n'a pas été affiché pendant trois jours ; cependant, vous pouvez indiquer que certains tampons ne soient jamais détruits automatiquement, et que d'autres soient détruits s'ils ont été inutilisés pendant une heure seulement.

Cette purge des tampons peut aussi être faite pour vous, chaque jour à minuit, en activant le mode Minuit. Le mode Minuit opère chaque jour à minuit ; à ce moment, il exécute `clean-buffer-list`, ou les fonctions quelles qu'elles soient que vous aurez placée dans le crochet normal `midnight-hook` (see Section 31.2.3 [Hooks], page 465).

Pour activer le mode Minuit, utilisez le tampon de Personnalisation pour mettre la variable `midnight-mode` à `t`. See Section 31.2.2 [Easy Customization], page 459.

15.5 Opérations sur Plusieurs Tampons

La facilité *menu tampon* est comme un "Dired pour tampons" ; il vous permet de demander des opérations sur divers tampons d'Emacs en éditant un tampon d'Emacs contenant une liste de ceux-ci. Vous pouvez sauvegarder des tampons, les détruire (appelé ici les *supprimer*, pour être cohérent avec Dired), ou les afficher.

`M-x buffer-menu`
: Commence l'édition du tampon listant tous les tampons d'Emacs.

La commande `buffer-menu` écrit la liste de tous les tampons d'Emacs dans le tampon '`*Buffer List*`', et sélectionne ce tampon dans le mode Menu Tampon. Le tampon est en lecture seule, et peut seulement être modifié par les commandes spéciales décrites dans cette section. Les commandes de déplacement de curseur habituelles d'Emacs peuvent être utilisées dans le tampon '`*Buffer List*`'. Les commandes suivantes s'appliquent au tampon décrit sur la ligne courante.

`d` Demande de supprimer (détruire) le tampon, puis descend le curseur. La requête est représentée par un '`D`' sur la ligne, avant le nom du tampon. Les suppressions demandées prennent effet lorsque vous tapez la commande `x`.

`C-d` Identique à `d` mais déplace le curseur vers le haut plutôt que vers le bas.

`s` Demande de sauvegarder le tampon. La requête est représentée par un '`S`' sur la ligne. Les requêtes de sauvegarde prennent effet lorsque vous tapez la commande `x`. Vous pouvez demander la sauvegarde et la suppression pour un même tampon.

`x` Exécute les requêtes de suppression et de sauvegardes précédentes.

`u` Annule toute requête faite pour la ligne courante, et déplace le curseur vers le bas.

⟨DEL⟩ Déplace le curseur sur la ligne précédente et annule toute requête faite pour cette ligne.

Les commandes `d`, `C-d`, `s` et `u` pour ajouter ou annuler un drapeau déplace aussi le curseur d'une ligne vers le bas (ou vers le haut). Elles acceptent un argument numérique comme compteur de répétition.

Ces commandes opèrent immédiatement sur le tampon listé sur la ligne courante :

`~` Marque le tampon "non modifié". La commande `~` prend effet immédiatement.

`%` Modifie le drapeau de lecture seule du tampon. La commande `%` prend effet immédiatement.

`t` Visite le tampon comme table de tags. See Section 22.16.4 [Select Tags Table], page 307.

Il existe aussi des commandes pour sélectionner un autre tampon ou d'autres tampons :

`q` Quitte le menu tampon—affiche immédiatement à la place le tampon le plus récemment visible dans cette fenêtre.

⟨RET⟩
`f` Sélectionne immédiatement le tampon de la ligne courante à la place du tampon '`*Buffer List*`'.

`o` Sélectionne imédiatement le tampon de la ligne courante dans une autre fenêtre comme avec `C-x 4 b`, laissant le tampon '`*Buffer List*`' visible.

`C-o` Affiche immédiatement le tampon de la ligne courante dans une autre fenêtre, mais ne sélectionne pas la fenêtre.

`1` Sélectionne immédiatement le tampon de la ligne courante dans une fenêtre plein-écran.

`2` Met immédiatement en place deux fenêtres, une contenant le tampon de la ligne courante, et l'autre contenant le tampon précédemment sélectionné (avant '`*Buffer List*`').

`b` Place le tampon listé sur cette ligne en fin de liste.

`m` Marque le tampon de la ligne courante comme devant être affiché dans une autre fenêtre si vous quittez avec la commande `v`. La requête est représentée par un '`>`' au début de la ligne. (Un tampon ne peut avoir à lui seul une requête de suppression et une requête d'affichage.)

`v` Sélectionne immédiatement le tampon de la ligne courante, et affiche de plus dans d'autres fenêtres les tampons précédemment marqués avec la commande `m`. Si vous n'avez marqué aucun tampon, cette commande est équivalente à `1`.

Ce que `buffer-menu` fait directement est de créer un tampon et de le sélectionner, et de le mettre dans le mode Menu Tampon. Tout le reste décrit précédemment est implémenté par les commandes spéciales fournies avec le mode Menu Tampon. Une conséquence de cela est que vous pouvez passer du tampon '`*Buffer List*`' à un autre tampon d'Emacs, et l'éditer. Vous pouvez resélectionner le tampon '`*Buffer List*`' plus tard, pour exécuter les opérations déjà demandées, ou vous pouvez le détruire, ou encore ne plus y faire attention.

La seule différence entre `buffer-menu` et `list-buffers` est que `buffer-menu` affiche le tampon '`*Buffer List*`' dans la fenêtre sélectionnée ; `list-buffers` l'affiche dans une autre fenêtre. Si vous exécutez `list-buffers` (par exemple, en tapant `C-x C-b`) et sélectionnez le tampon manuellement, vous pouvez utiliser toutes les commandes décrites ici.

Le tampon '`*Buffer List*`' n'est pas mis à jour automatiquement lorsque des tampons sont créés ou détruits ; son contenu est seulement du texte. Si vous avez créé, supprimé ou renommé des tampons, la manière de mettre à jour le tampon '`*Buffer List*`' pour voir ce que vous avez fait est de taper `g` (`revert-buffer`) ou répéter la commande `buffer-menu`.

15.6 Tampons Indirects

Un *tampon indirect* partage le texte avec un autre tampon, qui est appelé le *tampon de base* du tampon indirect. Il est en quelque sorte l'analogue, pour les tampons, d'un lien symbolique entre fichiers.

`M-x make-indirect-buffer` ⟨RET⟩ *base-buffer* ⟨RET⟩ *nom-indirect* ⟨RET⟩
Crée un tampon indirect appelé *nom-indirect* dont le tampon de base est *base-buffer*.

`M-x clone-indirect-buffer` ⟨RET⟩
: Crée un tampon indirect qui est une copie parfaite du tampon courant. Crée un tampon indirect qui est une copie parfaite du tampon courant, et le sélectionne dans une autre fenêtre (`clone-indirect-buffer-other-window`).

Le texte du tampon indirect est toujours identique au texte de son tampon de base ; les changements effectués en éditant l'un d'eux sont immédiatement visibles dans l'autre. Mais dans tous les autres cas, le tampon indirect et son tampon de base sont totalement séparés. Ils ont des noms différents, différentes valeurs du point, différentes restrictions, différentes marques, différents modes majeurs, et différentes variables locales.

Un tampon indirect ne peut pas visiter un fichier, mais son tampon de base le peut. Si vous essayez de sauvegarder le tampon indirect, le tampon de base est sauvegardé. Détruire le tampon de base détruit aussi le tampon indirect, mais détruire le tampon indirect n'a pas d'effet sur son tampon de base.

Une manière d'utiliser les tampons indirects est d'afficher plusieurs vues d'un profil. See Section 21.8.4 [Outline Views], page 259.

Une manière rapide et pratique pour créer un tampon indirect est d'utiliser la commande `M-x clone-indirect-buffer`. Elle crée et sélectionne un tampon indirect dont le tampon de base est le tampon courant. Avec un argument numérique, elle demande le nom du tampon indirect ; autrement elle prend par défaut le nom du tampon courant, le modifiant en ajoutant un préfixe '<*n*>' si nécessaire. `C-x 4 c` (`clone-indirect-buffer-other-window`) fonctionne comme `M-x clone-indirect-buffer`, mais elle sélectionne le tampon cloné dans une autre fenêtre. Ces commandes sont commodes pour créer de nouveaux tampons '`*info*`' or '`*Help*`', par exemple.

La maniée plus générale est d'utiliser la commande `M-x make-indirect-buffer`. Elle crée un tampon indirect à partir du tampon *base-buffer*, sous le nom *nom-indirect*. Elle demande *base-buffer* et *nom-indirect* en utilisant le mini-tampon.

15.7 Fonctionnalités de Confort et Personnalisation de la manipulation des

Tampons

15.7.1 Rendre les Noms de Tampon Uniques

Lorsque plusieurs tampons visitent des fichiers portant le même nom, Emacs doit donner aux tampons des noms distincts. La méthode habituelle

pour rendre les noms de tampons uniques est d'ajouter '`<2>`', '`<3>`', etc. à la fin des noms de tampons (tous sauf un d'eux).

D'autres méthodes fonctionnent en ajoutant une partie du répertoire de chaque fichier au nom du tampon. Pour en sélectionner une, personnalisez la variable `uniquify-buffer-name-style` (see Section 31.2.2 [Easy Customization], page 459).

Par exemple, la méthode de nommage `forward` ajoute une partie du nom de répertoire au début du nom du tampon ; en utilisant cette méthode, les tampons visitant '`/u/mernst/tmp/Makefile`' et '`/usr/projects/zaphod/Makefile`' seraient nommés '`tmp/Makefile`' et '`zaphod/Makefile`', respectivement (plutôt que '`Makefile`' et '`Makefile<2>`').

Au contraire, la méthode de nommage `post-forward` nommeraient les tampons '`Makefile|tmp`' et '`Makefile|zaphod`', et la méthode de nommage `reverse` les nommeraient '`Makefile\tmp`' et '`Makefile\zaphod`'. La différence non triviale entre `post-forward` et `reverse` arrive lorsque un seul nom de répertoire n'est pas suffisant pour distinguer deux fichiers ; dans ce cas `reverse` place les noms de répertoires dans l'ordre inverse, et '`/top/middle/file`' devient '`file\middle\top`', alors que `post-forward` les place dans l'ordre après le nom de fichier, comme dans '`file|top/middle`'.

La règle à utiliser pour placer les noms de répertoires dans le nom du tampon n'est pas très importante si vous *regardez* les noms de tampons avant d'en taper un. Mais en tant qu'utilisateur expérimenté, si vous connaissez la règle, vous n'aurez pas à regarder. Et vous pouvez alors trouver qu'une règle est plus facile à retenir ou plus rapide à utiliser.

15.7.2 Naviguer Entre les Tampons en Utilisant des

Sous-Chaînes

Le mode mineur global Iswitchb fournit un moyen de naviguer entre les tampons en utilisant des sous-chaînes de leurs noms. Il remplace les définitions normales de `C-x b`, `C-x 4 b`, `C-x 5 b`, et `C-x 4 C-o` par des comamndes alternatives qui sont plus "astucieuses."

Lorsque une de ces commandes vous demande un nom de tampon, vous pouvez taper seulement une sous-chaîne du nom que vous voulez choisir. Pendant que vous entrez la sous-chaîne, le mode Iswitchb affiche continuellement une liste de tampons qui correspondent à la sous-chaîne que vous avez tapée.

À tout moment, vous pouvez taper ⟨RET⟩ pour sélectionner le premier tampon de la liste. Ainsi, le moyen de sélectionner un tampon particulier est de le rendre premier dans la liste. Il y a deux moyens de faire ça. Vous pouvez taper une plus grande partie du nom du tampon et ainsi restreindre la liste, en excluant les tampons non voulus se trouvant au dessus de lui.

Alternativement, vous pouvez utiliser `C-s` et `C-r` pour faire tourner la liste jusqu'à ce que le tampon désiré se trouve en premier.

⟨TAB⟩ lors de la saisie du nom de tampon performe la complétion de la chaîne que vous avez entrée, en se basant sur la liste des tampons affichés.

15.7.3 Personnalisation des Menus de Tampons

`M-x bs-show`
: Crée une liste de tampons similaire à `M-x list-buffers` mais pesonnalisable.

`M-x bs-show` ouvre une liste de tampons similaire à celle normalement affichée avec `C-x C-b` mais que vous pouvez personnaliser. Si vous préférez celle-ci à la liste habituelle des tampons, vous pouvez relier cette commande à `C-x C-b`. Pour personnaliser cette liste de tampons, utilisez le groupe de personnalisation `bs` (see Section 31.2.2 [Easy Customization], page 459).

Le mode mineur global MSB ("MSB" veut dire "mouse select buffer," ou "sélection du tampon à la souris") fournit un menu de tampon différent et personnalisable que vous pouvez préférer. Il remplace les liens de `mouse-buffer-menu`, normalement sur `C-Down-Mouse-1`, et le menu Tampon de la barre de menus. Vous pouvez personnaliser ce menu dans le goupe de personnalisation `msb`.

16 Fenêtres Multiples

Emacs peut découper un cadre en deux ou plusieurs fenêtres. Plusieurs fenêtres peuvent afficher des extraits de différents tampons, ou différents extraits d'un même tampon. Plusieurs cadres impliquent toujours plusieurs fenêtres, car chaque cadre a son propre ensemble de fenêtres. Chaque fenêtre appartient à un et un seul cadre.

16.1 Concepts des Fenêtres Emacs

Chaque fenêtre Emacs affiche à tout moment un tampon d'Emacs. Un tampon unique peut apparaître dans plus d'une fenêtre ; dans ce cas, tout changement dans son texte est affiché dans toutes les fenêtres où il apparaît. Mais les fenêtres montrant le même tampon peuvent afficher diffeérentes parties de ce tampon, car chaque fenêtre a sa propre valeur du point.

À tout moment, une des fenêtres est la *fenêtre sélectionnée* ; le tampon affiché dans cette fenêtre est le tampon courant. Le curseur du terminal montre la location du point dans cette fenêtre. Toute autre fenêtre a aussi une location du point, mais le terminal ayant un seul curseur, il n'y a pas de moyen de montrer ces locations. Lorsque plusieurs cadres sont visibles dans X, chaque cadre a un curseur qui apparaît dans la fenêtre sélectionnée du cadre. Le curseur dans le cadre sélectionné est plein ; il est vide dans les autres cadres.

Les commandes pour déplacer le point affectent la valeur du point dans la fenêtre sélectionnée d'Emacs seulement. Elles ne modifient pas la valeur du point dans les autres fenêtres d'Emacs, même celles affichant le même tampon. Ceci est aussi vrai pour les commandes comme `C-x b` qui changent le tampon sélectionné dans la fenêtre sélectionnée ; elles n'affectent pas du tout les autres fenêtres. Cependant, d'autres commandes comme `C-x 4 b` sélectionnent une autre fenêtre et changent de tampon dans celle-ci. De même, toutes les commandes affichant des informations dans une fenêtre, comme (par exemple) `C-h f` (`describe-function`) et `C-x C-b` (`list-buffers`), fonctionnent en changeant de tampon dans une fenêtre non sélectionnée sans affecter la fenêtre sélectionnée.

Lorsque plusieurs fenêtres montrent le même tampon, elles peuvent avoir différntes régions, car elles avoir différentes valeurs du point. Cependant, elles ont toutes la même valeur pour la marque, car chaque tampon a une seule position de marque.

Chaque fenêtre a sa propre ligne de mode, qui affiche le nom du tampon, le status de modification, le modes majeur et les modes mineurs du tampon qui est affiché dans la fenêtre. See Section 1.3 [Mode Line], page 26, pour tous les détails sur la ligne de mode.

16.2 Découper des Fenêtres

`C-x 2` Découpe la fenêtre sélectionnée en deux fenêtres, une en dessous de l'autre (`split-window-vertically`).

`C-x 3` Découpe la fenêtre sélectionnée en deux fenêtres positionnées côte à côte. (`split-window-horizontally`).

`C-Mouse-2`
Sur la ligne de mode ou la barre de défilement de la fenêtre, découpe cette fenêtre.

La commande `C-x 2` (`split-window-vertically`) découpe la fenêtre sélectionnée en deux fenêtres, une en dessous de l'autre. Les deux fenêtres affichent alors le même tampon, avec la même valeur du point. Par défaut chacune des deux fenêtres a une hauteur égale à la moitié de la hauteur qu'avait l'ancienne fenêtre ; un argument numérique spécifie le nombre de lignes que doit avoir la fenêtre du haut.

`C-x 3` (`split-window-horizontally`) découpe la fenêtre sélectionnée en deux fenêtres côte à côte. Un argument numérique spécifie le nombre de colonnes que doit avoir la fenêtre de gauche. Une ligne de barre verticales sépare les deux fenêtres. Les fenêtres n'ayant pas la largeur totale de l'écran ont leur ligne de mode, mais elles sont tronquées. Sur des terminaux ne supportant pas la surbrillance, les lignes de mode tronquées n'apparaissent parfois pas en inversion vidéo.

Vous pouvez découper une fenêtre horizontalement ou verticalement en cliquant `C-Mouse-2` sur la ligne de mode ou sur la barre de défilement. La ligne de coupe passe par l'endroit où vous avez cliqué : si vous cliquez sur la ligne de mode, la nouvelle barre de défilement est placée au dessus du point pù vous avez cliqué ; si vous cliquez sur la barre de défilement, la ligne de mode de la fenêtre découpée se trouve à côté du point où vous avez cliqué.

Lorsqu'une fenêtre a une largeur inférieure à la largeur totale, les lignes de texte trop longues pour entrer dans la fenêtre sont fréquentes. Continuer toutes ces lignes pourrait rendre l'affichage confus? La variable `truncate-partial-width-windows` peut être mise à une valeur non `nil` pour forcer la troncation dans toutes les fenêtres dont la largeur est inférieure à la largeur totale de l'écran, indépendemment du tampon affiché et de sa valeur de `truncate-lines`. See Section 4.8 [Continuation Lines], page 49.

Le défilement horizontal est fréquemment utilisé dans des fenêtres côte à côte. See Chapter 11 [Display], page 103.

Si `split-window-keep-point` est non `nil`, le défaut, les deux fenêtres résultant de `C-x 2` héritent la valeur du point de la fenêtre d'origine. Cela veut dire que le défilement est inévitable. Si cette variable est `nil`, alors `C-x 2` essaie de ne pas déplacer le texte sur l'écran en plaçant le point dans chaque fenêtre à une position déjà visible dans la fenêtre. Elle sélectionne

aussi la fenêtre qui contenait la ligne décran sur laquelle le curseur était précédemment. Certains utilisateurs préfèrent le dernier mode sur des terminaux lents.

16.3 Utiliser les Autres Fenêtres

`C-x o` Sélectionne une autre fenêtre (`other-window`). C'est `o`, et non zéro.

`C-M-v` Fait défiler la fenêtre suivante (`scroll-other-window`).

`M-x compare-windows`
Trouve l'endroit suivant où le texte de la fenêtre sélectionnée ne correspondant pas au texte de la fenêtre suivante.

`Mouse-1` `Mouse-1`, sur la ligne de mode d'une fenêtre, sélectionne cette fenêtre mais n'y déplace pas le point (`mouse-select-window`).

Pour sélectionner une fenêtre différente, cliquez avec `Mouse-1` sur sa ligne de mode. Avec le clavier, vous pouvez passer à une autre fenêtre en tapant `C-x o` (`other-window`). C'est un `o`, comme "other," et non un zéro. Lorsqu'il y a plus de deux fenêtres, cette commande vous fait passer dans toutes les fenêtres dans un ordre cyclique, généralement de haut en bas et de gauche à droite. Après la fenêtre la plus en bas et la plus à droite, vous repassez à la fenêtre du coin haut-gauche. Un argument numérique indique de faire plusieurs pas dans l'ordre cyclique des fenêtres. Un argument négatif vous déplace dans l'ordre cyclique inverse. Lorsque le mini-tampon est actif, le mini-tampon est la dernière fenêtre du cycle ; vous pouvez passer de la fenêtre du mini-tampon à une autre fenêtre, et plus tard revenir dans le mini-tampon pour finir de taper l'argument demandé. See Section 5.2 [Minibuffer Edit], page 56.

Les commandes de défilement habituelles (see Chapter 11 [Display], page 103) s'appliquent à la fenêtre sélectionnée uniquement, mais il existe une commande pour faire défiler la fenêtre suivante. `C-M-v` (`scroll-other-window`) fait défiler la fenêtre que `C-x o` sélectionnerait. Elle prend un argument, positif ou négatif, comme `C-v`. (Dans le mini-tampon, `C-M-v` fait défiler la fenêtre qui contient l'aide du mini-tampon, si elle existe, plutôt que la fenêtre suivante dans l'ordre cyclique ordinaire.)

La commande `M-x compare-windows` vous permet de comparer deux fichiers ou tampons visibles dans deux fenêtres, en vous déplaçant dans celles-ci jusqu'à la prochaine différence. See Section 14.9 [Comparing Files], page 181, pour des détails.

16.4 Afficher Dans une Autre Fenêtre

`C-x 4` est une touche préfixe pour les commandes qui sélectionnent une autre fenêtre (découpant la fenêtre s'il y en a une seule) et sélectionne un tampon dans cette fenêtre. Des commandes `C-x 4` différentes donnent différents moyens de trouver le tampon à sélectionner.

`C-x 4 b` *nom-tampon* ⟨RET⟩
: Sélectionne le tampon *nom-tampon* dans une autre fenêtre. Ceci exécute `switch-to-buffer-other-window`.

`C-x 4 C-o` *nom-tampon* ⟨RET⟩
: Affiche le tampon *bufname* dans une autre fenêtre, mais ne sélectionne pas ce tampon ou cette fenêtre. Ceci exécute `display-buffer`.

`C-x 4 f` *nom-fichier* ⟨RET⟩
: Visite le fichier *nom-fichier* et sélectionne son tampon dans une autre fenêtre. Ceci exécute `find-file-other-window`. See Section 14.2 [Visiting], page 145.

`C-x 4 d` *répertoire* ⟨RET⟩
: Sélectionne un tampon Dired pour le répertoire *répertoire* dans une autre fenêtre. Ceci exécute `dired-other-window`. See Chapter 28 [Dired], page 387.

`C-x 4 m`
: Commence à composer un nouveau message électronique dans une autre fenêtre. Ceci exécute `mail-other-window` ; son analogue "same-window" est `C-x m` (see Chapter 26 [Sending Mail], page 357).

`C-x 4 .`
: Trouve une marque dans la table courante des marques, dans une autre fenêtre. Ceci exécute `find-tag-other-window`, la variante "other-window" de `M-.` (see Section 22.16 [Tags], page 301).

`C-x 4 r` *nom-fichier* ⟨RET⟩
: Visite le fichier *nom-fichier* en lecture seule, et sélectionne son tampon dans une autre fenêtre. Ceci exécute `find-file-read-only-other-window`. See Section 14.2 [Visiting], page 145.

16.5 Forcer l'Affichage dans la Même Fenêtre

Certaines commandes Emacs vous place dans un tampon spécifique avec un contenu spécial. Par exemple, `M-x shell` vous place dans un tampon appelé '`*Shell*`'. Par convention, toutes ces commandes sont écrites pour ouvrir ce tampon dans une fenêtre séparée. Mais vuos pouvez spécifier que certains de ces tampons doivent apparaître dans la fenêtre sélectionnée.

Si vous ajoutez un nom de tampon à la liste `same-window-buffer-names`, l'effet est que de telles commandes affichent ce tampon particulier dans la

fenêtre sélectionnée. Par exemple, si vous ajoutez `"*grep*"` à la liste, la commande `grep` affichera son tampon de sortie dans la fenêtre sélectionnée.

La valeur par défaut de `same-window-buffer-names` n'est pas `nil` : elle spécifie les noms de tampons '`*info*`', '`*mail*`' et '`*shell*`' (ainsi que d'autres utilisés par des packages Emacs plus obscurs). C'est pourquoi `M-x shell` place normalement le tampon '`*shell*`' dans la fenêtre sélectionnée. Si vous enlevez cet élément de la valeur de `same-window-buffer-names`, le comportement de `M-x shell` changera—elle ouvrira alors le tampon dans une autre fenêtre. Vous pouvez spécifier ces tampons d'un manière plus générale avec la variable `same-window-regexps`. Définissez-la comme une liste d'expressions rationnelles ; ainsi tout tampon dont le nom correspond une de ces expressions rationnelles est affiché dans la fenêtre courante. (Une fois de plus, ceci s'applique uniquement aux tampons qui sont normalement affichés dans une autre fenêtre.) La valeur par défaut de cette variable spécifie les tampons Telnet et rlogin.

Une caractéristique analogue vous permet de spécifier les tampons qui doivent être affichés dans leur propre cadre individuel. See Section 17.11 [Special Buffer Frames], page 212.

16.6 Supprimer et Réarranger les Fenêtres

`C-x 0` Supprime la fenêtre sélectionnée (`delete-window`). Le dernier caractère de la séquence de touches est un zéro.

`C-x 1` Supprime toutes les fenêtres du cadre sélectionné excepté la fenêtre sélectionnée. (`delete-other-windows`).

`C-x 4 0` Supprime la fenêtre sélectionnée et détruit le tampon qu'elle contient (`kill-buffer-and-window`). Le dernier caractère de la séquence de touches est un zéro.

`C-x ^` Rend la fenêtre sélectionnée plus haute (`enlarge-window`).

`C-x }` Rend la fenêtre sélectionnée plus large (`enlarge-window-horizontally`).

`C-x {` Rend la fenêtre sélectionnée plus étroite (`shrink-window-horizontally`).

`C-x -` Réduit la fentêtre si son tampon ne nécessite pas autant de lignes (`shrink-window-if-larger-than-buffer`).

`C-x +` Rend toutes les fenêtres de la même hauteur (`balance-windows`).

`Drag-Mouse-1`
Traîner la ligne de mode d'une fenêtre vers le haut ou vers le bas avec `Mouse-1` change la hauteur de la fenêtre.

`Mouse-2` `Mouse-2` sur la ligne de mode d'une fenêtre supprime toutes les autres fenêtres d'un cadre (`mouse-delete-other-windows`).

`Mouse-3` `Mouse-3` sur la ligne de mode d'une fenêtre supprime cette fenêtre (`mouse-delete-window`), à moins que le cadre ait seulement une fenêtre, auquel cas le tampon courant est mis en queue et un autre tampon prend sa place.

Pour supprimer une fenêtre, tapez `C-x 0` (`delete-window`). (C'est un zéro.) L'espace occupé par la fenêtre supprimée est donné à une fenêtre adjacente (en dehors de la fenêtre du mini-tampon, même si elle est active à ce moment-là). Une fois la fenêtre supprimée, ses attributs sont perdus ; seule la restauration de la configuration d'une fenêtre peut les retrouver. Supprimer une fenêtre n'a pas d'effet sur le tampon qu'elle affichait ; le tampon continue d'exister, et vous pouvez le sélectionner dans n'importe quelle fenêtre avec `C-x b`.

`C-x 4 0` (`kill-buffer-and-window`) est une commande plus puissante que `C-x 0` ; elle détruit le tampon courant puis supprime la fnêtre sélectionnée.

`C-x 1` (`delete-other-windows`) est plus puissante d'une manière différente ; elle supprime toutes les fenêtres sauf celle sélectionnée (et le mini-tampon) ; la fenêtre sélectionnée s'étend alors pour occuper tout le cadre à l'exception de la zone de répercussion.

Vous pouvez aussi supprimer une fenêtre en cliquant sr sa ligne de mode avec `Mouse-2`, et supprimer toutes les fenêtres d'un cadre sauf une en cliquant sur la ligne de mode de cette fenêtre avec `Mouse-3`.

Le moyen le plus facile d'ajuster la hauteur d'une fenêtre est d'utiliser la souris. Si vous pressez `Mouse-1` sur la ligne de mode, vous pouvez traîner cette ligne de mode vers le haut ou vers le bas, et changer les hauteurs des fenêtres au dessus et en dessous de cette ligne de mode.

Pour réajuster la division d'espace entre des fenêtres verticalement adjacentes, utilisez `C-x ^` (`enlarge-window`). Elle rend la fenêtre sélectionnée plus haute d'une ligne, ou d'autant de lignes que spécifié par un argument numérique. Avec un argument négatif, elle diminue la hauteur de la fenêtre. `C-x }` (`enlarge-window-horizontally`) rend la fenêtre sélectionnée plus large du nombre de colonnes spécifié. `C-x {` (`shrink-window-horizontally`) rend la fenêtre sélectionnée plus étroite du nombre de colonnes spécifié.

Lorsque vous agrandissez une fenêtre, l'espace provient d'une fenêtre adjacente. Si cette opération rend une fenêtre trop petite, elle est supprimée et son espace est donné à une fenêtre adjacente. La taille minimale est spécifiée par les variables `window-min-height` et `window-min-width`.

La commande `C-x -` (`shrink-window-if-larger-than-buffer`) réduit la hauteur de la fenêtre sélectionnée, si elle est plus grande que nécessaire pour montrer tout le texte du tampon qu'elle contient. Elle donne les lignes supplémentaires aux autres fenêtres du cadre.

Vous pouvez aussi utiliser `C-x +` (`balance-windows`) pour égaliser les hauteurs de toutes les fenêtres du cadre sélectionné.

16.7 Fonctionnalités de Confort de Manipulation de Fenêtres et

Personnalisation

`M-x winner-mode` est un mode mineur global qui sauvegarde les changements de configuration des fenêtres (c'est-à-dire comment les cadres sont partitionnés en fenêtres), pour que vous puissiez les "annuler." Pour annuler, utilisez `C-x left` (`winner-undo`). Si vous changez d'avis alors que vous annulez, vous pouvez refaire les changements que vous venez d'annuler en utilisant `C-x right` (`M-x winner-redo`). Un autre moyen d'activer le mode Winner est de personnaliser la variable `winner-mode`.

Les commande de Windmode vous déplacent directionnellement entre les fenêtre voisines dans un cadre. `M-x windmove-right` sélectionne la fenêtre immédiatement à droite de celle actuellement sélectionnée, et de même pour les contreparties "left," "up," et "down." `M-x windmove-default-keybindings` relie ces commandes à `S-right`, etc. (Tous les terminaux ne supportent cependant pas les touches de directions shiftées.)

Le mode mineur Suivi (`M-x follow-mode`) synchronise plusieurs fenêtres sur le même tampon pour qu'ils affichent toujours des sections adjacentes de ce tampon. See Section 11.8 [Follow Mode], page 112.

`M-x scroll-all-mode` fournit des commandes pour faire défiler toutes les fenêtres visibles en même temps. Vous pouvez aussi activer ce mode en personnalisant la variable `scroll-all-mode`. Les commandes fournies sont `M-x scroll-all-scroll-down-all`, `M-x scroll-all-page-down-all` et les équivalents "up" correspondants. Pour rendre ce mode utile, vous devez relier ces commandes à des touches appropriées.

17 Cadres et Fenêtres X

Lorsque vous utilisez le système X Window, vous pouvez créer plusieurs fenêtres X dans une seule session Emacs. Chaque fenêtre X appartenant à Emacs affiche un *cadre* qui peut contenir une ou plusieurs fenêtres Emacs. Un cadre contient initialement une seule fenêtre Emacs à usage général que vous pouvez subdiviser verticalement et horizontalement en fenêtres plus petites. Un cadre contient normalement ses propres zone de répercussion et mini-tampon, mais vous pouvez créer des cadres qui n'en ont pas—ils utilisent la zone de répercussion et le mini-tampon d'un autre cadre.

L'édition que vous faites dans un cadre affecte aussi les autres cadres. Par exemple, si vous placez du texte dans le presse-papier à partir d'un cadre, vous pouvez le coller dans un autre cadre. Si vous quittez Emacs avec `C-x C-c` à partir d'un cadre, vous quittez aussi tous les autres cadres. Pour supprimer un seul cadre, utilisez `C-x 5 0`.

Pour empêcher toute confusion, nous réservons le mot "fenêtre" aux subdivisions qu'Emacs implémente, et ne l'utilisons jamais pour nous référer à un cadre.

Emacs compilé pour MS-DOS émule certains aspects du système de fenêtrage pour que vous puissiez utiliser un grand nombre des fonctionnalités décrites dans ce chapitre. See Section E.1 [MS-DOS Input], page 539, pour plus d'informations.

Emacs compilé pour MS Windows supporte pratiquement les mêmes fonctionnalités que sous X. Cependant, les images, les barres d'outils et les tooltips ne sont pas encore disponibles sous MS Windows pour la version d'Emacs 21.1.

17.1 Commandes Souris Pour l'Édition

Les commandes souris pour sélectionner et copier une région sont pour la plupart compatibles avec le programme `xterm`. Vous pouvez utiliser les mêmes commandes souris pour copier du texte entre Emacs et les autres programmes clients X.

+Si vous sélectionnez une région avec une de ces commandes souris, puis tapez immédiatement après la touche de fonction ⟨DELETE⟩, vous supprimez la région que vous avez sélectionné. La touche de fonction ⟨BACKSPACE⟩ et le caractère ASCII ⟨DEL⟩ ne font pas cela ; si vous tapez une autre touche entre la commande souris et ⟨DELETE⟩, la région n'est pas supprimée.

`Mouse-1` Déplace le point à l'endroit du clic (`mouse-set-point`). C'est normalement le bouton gauche.

`Drag-Mouse-1`
: Définit la région comme contenant le texte que vous avez sélectionné en traînant la souris, et la copie dans le presse-papiers (`mouse-set-region`). Vous pouvez spécifier les deux extrémités de la région avec cette unique commande.

  Si vous déplacez la souris au delà du haut ou du bas de la fenêtre tout en la traînant, la fenêtre défile à un rythme régulier jusqu'à ce que vous replaciez la souris dans la fenêtre. De cette manière, vous pouvez sélectionner des régions qui ne rentrent pas entièrement dans l'écran. Le nombre de lignes défilées à chaque pas dépend de la distance entre la souris et la limite de la fenêtre ; la variable `mouse-scroll-min-lines` spécifie une taille minimale du pas.

`Mouse-2`
: Colle le texte dernièrement coupé, à l'endroit du clic (`mouse-yank-at-click`). C'est normalement le bouton central.

`Mouse-3`
: Cette commande, `mouse-save-then-kill`, a plusieurs fonctions dépendant de l'endroit où vous cliquez et le status de la région.

  La cas le plus élémentaire est lorsque vous cliquez `Mouse-1` à un endroit puis `Mouse-3` à un autre endroit. Ceci sélectionne le texte entre ces deux positions comme région. Il copie aussi la nouvelle région dans le presse-papiers, pour que vous puissiez le copier à un autre endroit.

  Si vous cliquez `Mouse-1` dans le texte, faites défiler la fenêtre avec la barre de défilement puis cliquez `Mouse-3`, il se souvient de l'endroit où se trouvait le point avant le défilement (où vous l'avez placé avec `Mouse-1`), et utilise cette position comme autre extrémité de la région. Ainsi vous pouvez sélectionner une région qui n'entre pas entièrement dans l'écran.

  Plus généralement, si vous n'avez pas de région en surbrillance, `Mouse-3` sélectionne le texte entre le point et la position du clic comme région. Il fait ceci en plaçant la marque où se trouvait le point, et en déplaçant le point où vous cliquez.

  Si vous avez une région en surbrillance, ou si la région a été définie juste avant en traînant le bouton 1, `Mouse-3` ajuste l'extrémité la plus proche de la région en la déplaçant à l'endroit du clic. Le texte de la région ajustée remplace aussi le texte de l'ancienne région dans le presse-papiers.

  Si vous avez d'abord spécifié la région en utilisant un double ou triple `Mouse-1`, la région définie consistant alors en mots entiers ou lignes entières, l'ajustement de la région avec `Mouse-3` procède aussi par mots entiers ou lignes entières.

  Si vous utilisez `Mouse-3` une seconde fois consécutivement, au même endroit, vous coupez la région déjà sélectionnée.

`Double-Mouse-1`
: Cette touche place la région autour du mot que vous cliquez. Si vous cliquez sur un caractère avec une syntaxe “symbole” (comme un souligné, dans le mode C), elle place la région autour du symbole entourant ce caractère.

  Si vous cliquez sur un caractère avec une syntaxe ouverture ou fermeture de parenthèse, elle place la région autour du groupe parenthésé commençant ou finissant à ce caractère. Si vous cliquez sur un caractère avec une syntaxe délimitation de chaîne (comme un guillemet simple ou double en C), elle place la région autour de la constante chaîne (en utilisant des heuristiques pour déterminer si ce caractère est au début ou à la fin de la constante).

`Double-Drag-Mouse-1`
: Cette touche sélectionne une région faite de mots sur lesquels vous avez traîné la souris.

`Triple-Mouse-1`
: Cette touche place la région autour de la ligne cliquée.

`Triple-Drag-Mouse-1`
: Cette touche sélectionne une région faite de lignes sur lesquelles vous avez traîné la souris.

La moyen le plus simple de couper du texte avec la souris est de presser `Mouse-1` à une extrémité, puis presser `Mouse-3` deux fois à l’autre extrémité. See Section 9.1 [Killing], page 85. Pour copier le texte dans le presse-papiers sans le supprimer du tampon, pressez `Mouse-3` seulement une fois—une traînez juste la souris à travers le texte avec `Mouse-1`. Vous pouvez alors copier le texte autre part en le collant.

Pour coller le texte coupé ou copié autre part, déplacez la souris à cet endroit et pressez `Mouse-2`. See Section 9.2 [Yanking], page 89. Cependant, si `mouse-yank-at-point` est non `nil`, `Mouse-2` colle le texte au point. L’endroit où vous cliquez n’est alors pas important, ni même dans laquelle des fenêtres du cadre vous cliquez. La valeur par défaut est `nil`. Cette variable affecte aussi le collage de la seconde sélection.

Pour copier du texte dans une autre fenêtre X, coupez-le ou sauvegardez-le dans le presse-papiers. Sous X, cela met aussi la *sélection primaire*. Utilisez alors la commande “coller” du programme contrôlant l’autre fenêtre X pour insérer le texte de la sélection.

Pour copier du texte provenant d’une autre fenêtre X, utilisez la commande “couper” ou “copier” du programme contrôlant l’autre fenêtre, pour sélectionner le texte que vous désirez copier. Collez-le alors dans Emacs avec `C-y` ou `Mouse-2`.

Ces commandes de coupe et de colle fonctionnent aussi sous MS-Windows.

Lorsqu'Emacs place du texte dans le presse-papiers, ou déplace du texte en haut de la pile du presse-papiers, il remplit aussi la *sélection primaire* du serveur X. C'est ainsi que d'autres applications X peuvent accéder au texte. Emacs enregistre aussi le texte dans le tampon de coupe, mais seulement si le texte est assez court (`x-cut-buffer-max` spécifie le nombre maximum de caractères) ; placer de longs morceaux de texte dans le tampon de coupe peut être lent.

Les commandes pour coller la première entrée du presse-papiers vérifient d'abord l'existence d'une sélection primaire dans un autre programme ; après cela, elles vérifient l'existence de texte dans le tampon de coupe. Si aucune de ces sources ne fournit de texte à coller, le contenu du presse-papiers est utilisé.

17.2 Sélection Secondaire

La *sélection secondaire* est un autre moyen de sélectionner du texte en utilisant X. Il n'utilise ni le point ni la marque, vous pouvez donc l'utiliser pour couper du texte sans déplacer le point ou la marque.

`M-Drag-Mouse-1`
: Place la sélection secondaire, une extrémité à l'endroit où vous pressez le bouton, et l'autre extrémité où vous le relâchez (`mouse-set-secondary`). La surbrillance apparaît et change lorsque vous traînez la souris.

  Si vous déplacez la souris en dehors des limites de la fenêtre tout en traînant la souris, la fenêtre défile à un rythme régulier jusqu'à que vous rameniez la souris dans la fenêtre. De cette manière, vous pouvez marquer des régions qui n'entrent pas entièrement dans l'écran.

`M-Mouse-1`
: Place une extrémité de la *sélection secondaire* (`mouse-start-secondary`).

`M-Mouse-3`
: Définit une sélection secondaire, en utilisant la place spécifiée avecc `M-Mouse-1` comme autre extrémité (`mouse-secondary-save-then-kill`). Un second clic au même endroit efface la séélection secondaire.

`M-Mouse-2`
: Insère la sélection secondaire à l'endroit du clic (`mouse-yank-secondary`). Le point est alors placé à la fin du texte collé.

Des doubles et triples clics de `M-Mouse-1` opèrent sur des mots et lignes, exactement comme `Mouse-1`.

Si `mouse-yank-at-point` est non `nil`, `M-Mouse-2` colle le texte au point. L'endroit où vous cliquez n'est alors pas important ; seule importe la fenêtre dans laquelle vous cliquez. See Section 17.1 [Mouse Commands], page 203.

17.3 Utiliser le Presse-Papiers

En plus des types de sélection primaire et secondaire, X supporte un type de sélection *presse-papiers* utilisé par certaines applications, particulièrement sous OpenWindows et Gnome.

La commande `M-x menu-bar-enable-clipboard` fait que les entrées de menu `Cut`, `Paste` et `Copy` ainsi que les touches du même nom utilisent toutes le presse-papiers.

Vous pouvez personnaliser l'option `x-select-enable-clipboard` pour que les fonctions de colle d'Emacs consultent le presse-papiers avant la sélection primaire, et pour que les fonctions de coupe stockent dans le presse-papiers en plus de la sélection primaire. Autrement, elles n'accèdent pas du tout au prese-papiers. Le presse-papiers est utilisé par défaut sous MS Windows, au contraire de la plupart des systèmes.

17.4 Suivre des Références Avec la souris

Certains tampons Emacs affichent des listes de types variés. Celles-ci incluent des listes de fichiers, de tampons, de complétions possibles, de correspondances pour un motif, etc.

Copier du texte dans ces tampons n'étant pas vraiment utile, la plupart de ces tampons définissent `Mouse-2` différemment, comme une commande pour utiliser ou voir l'élément sur lequel vous cliquez.

Par exemple, si vous cliquez `Mouse-2` sur un nom de fichier dans un tampon Dired, vous visitez ce fichier. Si vous cliquez `Mouse-2` sur un message d'erreur dans un tampon '`*Compilation*`', vous allez dans le code source correspondant à ce message d'erreur. Si vous cliquez `Mouse-2` sur une complétion dans un tampon '`*Completions*`', vous choisissez cette complétion.

Vous pouvez habituellement savoir si `Mouse-2` a ce comportement particulier car le texte passe en surbrillance lorsque vous déplacez la souris sur celui-ci.

17.5 Clics de Souris Pour les Menus

Des clics de souris modifiés par les touches ⟨CTRL⟩ et ⟨SHIFT⟩ font apparaître des menus.

`C-Mouse-1`
: Ce menu permet la sélection d'un tampon.

  Le mode mineur global MSB ("mouse select buffer") rend ce menu plus élégant et personnalisable. See Section 15.7.3 [Buffer Menus], page 194.

`C-Mouse-2`
: Ce menu permet de spécifier la face ou d'autres propriétés de texte pour éditer du texte formaté. See Section 21.11 [Formatted Text], page 265.

`C-Mouse-3`
: Ce menu est spécifique au mode. Pour la plupart des modes si le mode Barre de Menus est actif, ce menu a les mêmes entrées que tous les menus spécifiques au mode de la barre de menu réunis. Certains modes peuvent spécifier un menu différent pour ce bouton.[1] si le mode Barre de Menus est inactif, ce menu contient toutes les entrées qui devraient être présentes dans la barre de menus—et pas seulement celles spécifiques au mode—pour que vous puissiez y accéder sans avoir à afficher la barre de menus.

`S-Mouse-1`
: Ce menu permet de spécifier la police principale du cadre.

17.6 Commandes Souris de la Ligne de Mode

Vous pouvez utiliser des clics de souris sur les lignes de mode des fenêtres pour sélectionner et manipuler des fenêtres.

`Mouse-1` `Mouse-1` sur une ligne de mode sélectionne la fenêtre du dessus. En traînant `Mouse-1` sur la ligne de mode, vous pouvez la déplacer, et ainsi changer les hauteurs des fenêtres du dessus et du dessous.

`Mouse-2` `Mouse-2` sur une ligne de mode étend cette fenêtre pour qu'elle remplisse son cadre.

`Mouse-3` `Mouse-3` sur une ligne de mode supprime la fenêtre du dessus. Si le cadre a seulement une fenêtre, le tampon courant est mis en queue et un autre tampon est affiché.

---

[1] Certains systèmes utilisent `Mouse-3` pour un menu spécifique au mode. Nous avons réalisé un sondage d'utilisateurs, et avons trouvé qu'ils préféraient garder `Mouse-3` pour sélectionner et couper des régions. D'où la décision d'utiliser `C-Mouse-3` pour ce menu.

`C-Mouse-2`
: `C-Mouse-2` sur une ligne de mode découpe la fenêtre du dessus horizontalement, au dessus de l'endroit où vous avez cliqué.

`C-Mouse-2` sur une barre de défilement découpe la fenêtre correspondante verticalement, à moins que vous utilisiez une implémentation X des barres de défilement. See Section 16.2 [Split Window], page 196.

Les commandes précédentes s'appliquent aux zones de la ligne de mode n'ayant pas elles-mêmes de raccourcis souris. Certaines zones, comme le nom de tampon et le nom du mode majeur, ont leurs propres raccourcis souris. Emacs affiche les informations sur le raccourci lorsque vous laissez la souris au dessus d'un tel endroit.

17.7 Créer des Cadres

La touche préfixe `C-x 5` est analogue à `C-x 4`, avec des sous-commandes parallèles. a différence est que les commandes `C-x 5` créent un nouveau cadre plutôt qu'une nouvelle fenêtre dans le cadre sélectionné. (see Section 16.4 [Pop Up Window], page 198.) Si un cadre visible ou iconifié affiche déjà ce que vous demandez, ces commandes utilisent le cadre existant, après l'avoir désiconifié ou mis en avant-plan si nécessaire.

Les diverses commandes `C-x 5` différent dans la manière où elles trouvent ou créent le tampon à sélectionner :

`C-x 5 2`
: Crée un nouveau cadre (`make-frame-command`).

`C-x 5 b` *nom-tampon* ⟨RET⟩
: Sélectionne le tampon *nom-tampon* dans un autre cadre. Ceci exécute `switch-to-buffer-other-frame`.

`C-x 5 f` *nom-fichier* ⟨RET⟩
: Visite le fichier *nom-fichier* et sélectionne son tampon dans un autre cadre. Ceci exécute `find-file-other-frame`. See Section 14.2 [Visiting], page 145.

`C-x 5 d` *répertoire* ⟨RET⟩
: Sélectionne un tampon Dired pour le répertoire *répertoire* dans un autre cadre. Ceci exécute `dired-other-frame`. See Chapter 28 [Dired], page 387.

`C-x 5 m`
: Commence à composer un message électronique dans un autre cadre. Ceci exécute `mail-other-frame`. C'est la variante "other-frame" de `C-x m`. See Chapter 26 [Sending Mail], page 357.

`C-x 5 .`
: Trouve une marque dans la table courante des marques, dans un autre cadre. Ceci exécute `find-tag-other-frame`, la variante "other-frame" de `M-.`. See Section 22.16 [Tags], page 301.

`C-x 5 r` *nom-fichier* ⟨RET⟩
: Visite le fichier *nom-fichier* en lecture seule, et sélectionne son tampon dans un autre cadre. Ceci exécute `find-file-read-only-other-frame`. See Section 14.2 [Visiting], page 145.

Vous pouvez contrôler l'apparence des nouveaux cadres que vous créez en définissant les paramètres du cadre dans `default-frame-alist`. Vous pouvez utiliser la variable `initial-frame-alist` pour spécifier les paramètres qui affectent seulement le cadre initial. See section "Initial Parameters" in *The Emacs Lisp Reference Manual*, pour plus d'informations.

Le meilleur moyen de spécifier la police principale pour tous vos cadres Emacs est avec une ressource X (see Section B.7 [Font X], page 516), mais vous pouvez aussi le faire en modifiant `default-frame-alist` pour spécifier le paramètre `police` comme montré ici :

```
(add-to-list 'default-frame-alist '(font . "10x20"))
```

17.8 Commandes pour les Cadres

Les commandes suivantes vous permettent de créer, supprimer et opérer sur les cadres :

`C-z`
: Iconifie le cadre Emacs sélectioné (`iconify-or-deiconify-frame`). Le comportement normal de `C-z`, de suspendre Emacs, n'est pas utile dans un système de fenêtres, et a donc un lien différent dans ce cas.

  Si vous tapez cette commande dans un icône d'un cadre Emacs, elle desiconifie le cadre.

`C-x 5 0`
: Supprime le cadre sélectionné (`delete-frame`). Ce n'est pas permis s'il n'y a qu'un seul cadre.

`C-x 5 o`
: Sélectionne un autre cadre, le met en avant-plan et y déplace la souris pour qu'il reste sélectionné. Si vous répétez cette commande, elle fait un cycle à travers tous les cadres de votre terminal.

`C-x 5 1`
: Supprime tous les cadres hormis celui sélectionné.

17.9 Créer et Utiliser un Cadre Speedbar

Un cadre Emacs peut avoir une *peedbar*, qui est une fenêtre verticale servant de menu défilable de fichiers que vous pouvez visiter et de marques dans ces fichiers. Pour créer une speedbar, tapez `M-x speedbar`; ceci crée une fenêtre speedbar pour le cadre sélectionné. À partir de là, vous pouvez

cliquer sur un nom de fichier dans la speedbar pour visiter ce fichier dans le cadre Emacs correspondant, ou cliquer sur le nom d'une marque pour sauter à cette marque dans le cadre Emacs.

Initialement, la speedbar liste le contenu immédiat du répertoire courant, un fichier par ligne. Chaque ligne a aussi un e boîte, '`[+]`' ou '`<+>`', sur laquelle vous pouvez cliquer avec `Mouse-2` pour "ouvrir" le contenu de cet élément. Si la ligne représente un répertoire, l'ouvrir ajoute le contenu de ce répertoire à la speedbar, juste en dessous de la ligne du répertoire. Si une ligne affiche un fichier ordinaire, l'ouvrir ajoute une liste des marques dans ce fichier à la speedbar. Lorsqu'un fichier est ouvert, le '`[+]`' se change en '`[-]`' ; vous pouvez cliquer sur cette boîte pour "fermer" ce fichier (cacher son contenu).

Certains modes majeurs, dont les modes Rmail, Info et GUD, ont une manière spécialisée de placer des éléments dans la speedbar que vous pouvez sélectionner. Par exemple, dans le mde Rmail, la speedbar affiche une liste de fichiers Rmail, et vous permet de déplacer le message courant dans un autre fichier Rmail en cliquant sur sa boîte '`<M>`'.

Une speedbar appartient à un cadre Emacs, et opère toujours sur ce cadre. Si vous utilisez plusieurs cadres, vous pouvez créer des speedbar pour certains ou tous les cadres ; tapez `M-x speedbar` dans un des cadres pour lui créer une speedbar.

17.10 Visuels Multiples

Un seul Emacs peut communiquer avec plus d'un visuel X. Initialement, Emacs utilise seulement un visuel—celui spécifié avec la variable d'environnement `DISPLAY` ou avec l'option '`--display`' (see Section B.2 [Initial Options], page 508). Pour vous connecter à un autre visuel, utilisez la commande `make-frame-on-display`:

`M-x make-frame-on-display` ⟨RET⟩ *visuel* ⟨RET⟩
: Crée un nouveau cadre sur le visuel *visuel*.

Un unique serveur X peut supporter plus d'un écran. Lorsque vous ouvrez des cadres sur deux écrans appartenant à un même serveur X, Emacs sait qu'ils partagent un même clavier, et traite toutes les commandes arrivant de ces écrans comme un unique flux d'entrée.

Lorsque vous ouvrez des cadres sur différents serveurs X, Emacs crée un flux d'entrée séparé pour chaque serveur. De cette manière, deux utilisateurs peuvent tamper simultanément sur les deux visuels, et Emacs ne mélangera pas leurs entrées. Chaque serveur a aussi son propre cadre sélectionné. Les commandes que vous entrez sur un serveur X spécifique s'appliquent au cadre sélectionné de ce serveur.

Malgré ces fonctionnalités, des personnes utilisant la même instance d'Emacs depuis différents visuels peuvent tout de même interférer entre eux

s'ils ne sont pas prudents. Par exemple, si l'un d'eux tape `C-x C-c`, la session d'Emacs se termine pour tout le monde !

17.11 Cadres Pour Tampons Spéciaux

Vous pouvez faire apparaître certains tampons, pour lesquels Emacs crée normalement une seconde fenêtre lorsque vous avez une seule fenêtre, dans leur propre cadre. Pour faire cela, définissez la variable `special-display-buffer-names` comme un liste de noms de tampons ; tout tampon dont le nom est dans cette liste est automatiquement placé dans un cadre spécial, lorsqu'Emacs tente de l'afficher dans "une autre fenêtre".

Par exemple, si vous définissez la variable ainsi,

```
(setq special-display-buffer-names
      '("*Completions*" "*grep*" "*tex-shell*"))
```

alors les tampons de listes de complétion, de sortie de `grep` et de mode shell TEX apparaissent dans leur propre cadre. Ces cadres, et les fenêtres qu'ils contiennent, ne sont jamais découpés automatiquement ou réutilisés pour un autre tampon. Ils continuent à afficher les tampons pour lesquels ils ont été créés, à moins que vous les modifiez à la main. La destruction d'un tampon spécial entraîne automatiquement la destruction de son cadre.

Plus généralement, vous pouvez définir `special-display-regexps` comme une liste d'expressions rationnelles ; ainsi un tampon obtient son propre cadre si son nom correspond à une des expressions rationnelles. (Une fois encore, ceci s'applique seulement aux tampons qui sont normalement affichés dans une fenêtre séparée.)

La variable `special-display-frame-alist` spécifie les paramètres pour ces cadres. Elle a une valeur par défaut, vous n'avez donc pas besoin de la définir.

Pour ceux qui connaissent Lisp, un élément de `special-display-buffer-names` ou `special-display-regexps` peut aussi être une liste. Alors le premier élément est le nom du tampon ou une expression rationnelle ; le reste de la liste spécifie comment créer le cadre. Cela peut être une liste d'associations spécifiant les valeurs des paramètres du cadre ; ces valeurs sont prioritaires par rapport aux valeurs spécifiées dans `special-display-frame-alist`. Alternativement, ça peut avoir cette forme :

(*fonction args...*)

où *fonction* est un symbole. Le cadre est alors construit en appelant *fonction*; son premier argument est le tampon, et ses arguments suivants sont *args*.

Une fonctionnalité analogue vous permet de spécifier les tampons qui doivent affichés dans la fenêtre sélectionnée. See Section 16.5 [Force Same Window], page 198. La fonctionnalité "même fenêtre" l'emporte sur la fonctionnalité "cadre spécial" ; de ce fait, si vous ajoutez un nom de tampon à

`special-display-buffer-names` et que ça n'a pas d'effet, vérifiez que cette fonctionnalité n'est pas aussi utilisée pour le même nom de tampon.

17.12 Définir les Paramètres des Cadres

Cette section décrit les commandes permettant d'altérer le style d'affichage et le comportement du gestionnaire de fenêtres pour le cadre sélectionné.

`M-x set-foreground-color` ⟨RET⟩ *couleur* ⟨RET⟩
: Spécifie la couleur *couleur* comme couleur d'avant-plan du cadre sélectionné. (Change aussi la couleur d'avant-plan de la face par défaut.)

`M-x set-background-color` ⟨RET⟩ *couleur* ⟨RET⟩
: Spécifie la couleur *couleur* comme couleur d'arrière-plan du cadre sélectionné. (Change aussi la couleur de fond de la face par défaut.)

`M-x set-cursor-color` ⟨RET⟩ *couleur* ⟨RET⟩
: Spécifie la couleur *couleur* pour le curseur du cadre sélectionné.

`M-x set-mouse-color` ⟨RET⟩ *couleur* ⟨RET⟩
: Spécifie la couleur *couleur* pour le curseur souris lorsqu'il est sur le cadre sélectionné.

`M-x set-border-color` ⟨RET⟩ *color* ⟨RET⟩
: Spécifie la couleur *couleur* pour la bordure du cadre sélectionné.

`M-x list-colors-display`
: Affiche les noms des couleurs définies et montre à quoi ressemble ces couleurs. Cette commande est parfois lente.

`M-x auto-raise-mode`
: Indique si le cadre sélectionné doit être ou non auto-élevé. Auto-élevé veut dire que chaque fois que vous déplacez la souris dans le cadre, le cadre est élevé.

  Notez que cette fonctionnalité d'auto-élévation est implémentée par Emacs lui-même. Certains gestionnaires de fenêtres implémentent aussi l'auto-élévation. Si vous permettez l'auto-élévation des cadres d'emacs dans votre gestionnaire de fenêtres, ça doit marcher, mais sous le contrôle d'Emacs et par conséquent `auto-raise-mode` n'a pas d'effet dessus.

`M-x auto-lower-mode`
: Indique si le cadre sélectionné doit être ou non auto-abaissé. Auto-abaissé veut dire que chaque fois que vous déplacez la souris en dehors du cadre, le cadre est positionné en bas de la pile de fenêtres X.

La commande `auto-lower-mode` n'a pas d'effet sur l'auto-abaissement implémenté par le gestionnaire de fenêtres. Pour contrôler cela, vous devez utiliser la fonctionnalité appropriée du gestionnaire de fenêtres.

`M-x set-frame-font` ⟨RET⟩ *police* ⟨RET⟩
Spécifie la police *police* comme police principale du cadre sélectionné. La police principale contrôle plusieurs attributs de face de la face `default` (see Section 11.1 [Faces], page 103). Par exemple, si la police principale a une hauteur de 12 pt, tout le texte sera dessiné avec des polices de 12 pt, à moins que vous utilisiez une autre face qui spécifie une hauteur différente. See Section B.7 [Font X], page 516, pour les moyens de connaître les polices disponibles sur votre système.

Vous pouvez aussi choisir la police principale d'un cadre par un menu. Pressez Press `S-Mouse-1` pour activer ce menu.

Pour les versions d'Emacs utilisant la boîte à outils X, les fonctions de choix de couleur et de police n'affectent pas les menus et la barre de menus, car ils sont affichés par leurs propres classes de widget. Pour changer l'apparence des menus et de la barre des menus, vous devez utiliser les ressources X (see Section B.13 [Resources X], page 521). See Section B.8 [Colors X], page 517, pour les couleurs. See Section B.7 [Font X], page 516, pour le choix d'une police.

Pour des informations sur les paramètres des cadres et le personalisation, voir section "Frame Parameters" in *The Emacs Lisp Reference Manual*.

17.13 Barres de Défilement

Lorsqu'il utilise X, Emacs crée normalement une *barre de défilement* à gauche de chaque fenêtre Emacs.[2] La barre de défilement couvre toute la hauteur de la fenêtre, et montre une rectangle déplaçable représentant la portion du tampon couramment affiché. La hauteur totale de la barre de défilement représente la longueur totale du tampon.

Vous pouvez utiliser `Mouse-2` (normalement, le bouton du milieu) dans la barre de défilement pour déplacer ou traîner le rectangle vers le haut ou vers le bas. Si vous le déplacez en haut de la barre de défilement, vous voyez le début du tampon. Si vous le déplacez en bas de la barre de défilement, vous voyez la fin du tampon.

Un clic sur les boutons gauche et droite dans la barre de défilement fait défiler par incréments contrôlés. `Mouse-1` (normalement, le bouton de gauche) déplace la ligne au niveau où vous cliquez en haut de la fenêtre.

[2] La placer à gauche est habituellement plus pratique avec des cadres se chevauchant et le texte commençant à la marge gauche.

`Mouse-3` (normalement, le bouton droit) déplace la ligne se trouvant en haut de la fenêtre vers le niveau où vous cliquez. En cliquant de façon répétée au même endroit, vous pouvez faire défiler le tampon d'une même distance de façon répétée.

Si vous utilisez l'implémentation Emacs des barres de défilement, à l'opposé des barres de défilement du toolkit X, vous pouvez aussi cliquer sur `C-Mouse-2` dans la barre de défilement pour découper une fenêtre verticalement. La fenêtre est découpée sur la ligne où vous cliquez.

Vous pouvez permettre ou interdire le mode Barre de Défilement avec la commande `M-x scroll-bar-mode`. Sans argument, il modifie l'usage de la barre de défilement. Avec un argument, il permet l'usage des barres de défilement si et seulement si l'argument est positif. Cette commande s'applique à tous les cadres, y compris les cadres non encore créés. Personnalisez l'option `scroll-bar-mode` pour contrôler l'utilisation des barres de défilement au démarrage. Vous pouvez l'utiliser pour spécifier qu'elles sont placées à droite de la fenêtre si vous préférez. Vous pouvez utiliser la ressource X '`verticalScrollBars`' pour contrôler l'état initial du mode Barre de Défilement. See Section B.13 [Resources X], page 521.

Pour permettre ou interdire les barres de défilement pour le cadre sélectionné uniquement, utilisez la commande `M-x toggle-scroll-bar`.

17.14 Scrolling With "Wheeled" Mice

Certaines souris ont une "molette" à la place du troisième bouton. Vous pouvez habituellement cliquer sur la molette pour agir comme `Mouse-3`. Vous pouvez aussi utiliser la molette pour faire défiler des fenêtres plutôt que d'utiliser la barre de défilement ou les commandes au clavier. Utilisez `M-x mouse-wheel-install` pour utiliser la molette pour faire défiler ou placez '`(require 'mouse-wheel)`' dans votre fichier '`.emacs`'. (Le support de la molette dépend du système générant les évènements appropriés pour Emacs.)

Les variables `mouse-wheel-follow-mouse` et `mouse-wheel-scroll-amount` déterminent où et de combien les tampons sont défilés.

17.15 Barres de Menu

Vous pouvez afficher ou non les barres de menu avec `M-x menu-bar-mode` ou en personnalisant l'option `menu-bar-mode`. Sans argument, cette commande lance ou arrête le mode Barre de Menu, un mode mineur. Avec un argument, cette commande lance le mode Barre de Menu si l'argument est positif, ou l'arrête si l'argument n'est pas positif. Vous pouvez utiliser la ressource X '`menuBarLines`' pour contrôler l'état initial du mode Barre de Menu. See Section B.13 [Resources X], page 521.

Les utilisateurs experts désactivent fréquemment la barre de menus, spécialement sur les terminaux texte seulement, où elle prend une ligne supplémentaire de texte. Si la barre de menu est désactivée, vous pouvez toujours faire apparaître un menu avec son contenu avec `C-Mouse-3`, sur un terminal supportant les menus flottants. See Section 17.5 [Menu Mouse Clicks], page 208.

See Section 1.4 [Menu Bar], page 28, pour savoir comment appeler des commandes avec la barre de menus.

17.16 Barres d'Outils

La *barre d'outils* est unr ligne (ou plusieurs lignes) d'icônes en haut de la fenêtre Emacs. Vous pouvez cliquer sur ces icônes avec la souris pour exécuter diverses tâches.

La barre d'outils générale contient les commandes générales. Certains modes majeurs définissent leur propre barre d'outils pour la remplacer. Un petit nombre de modes "spéciaux" qui ne sont pas prévus pour une édition ordinaire suppriment certains éléments de la barre d'outils générale.

Les barres d'outils apparaissent seulement dans les systèmes graphiques. La barre d'outils utilise des icônes XPM colorés si Emacs a été construit avec le support XPM. Autrement, la barre d'outils utilise des icônes monochromes (au format PBM ou XBM).

Vous pouvez activer ou désactiver l'affichage des barres d'outils avec `M-x tool-bar-mode`.

17.17 Utiliser des Boîtes de Dialogue

Une boîte de dialogue est un type spécial de menu vous posant une question oui-ou-non ou une autre question spéciale. Beaucoup de commandes Emacs utilisent une boîte de dialogue pour poser une question oui-ou-non, si vous avez utilisé la souris pour appeler la commande.

Vous pouvez personnaliser l'option `use-dialog-box` pour supprimer l'utilisation de boîtes de dialogue. Elle contrôle aussi l'utilisation de fenêtres de sélection de fichiers (mais celles-ci ne sont pas supportées sur toutes les plateformes).

17.18 Tooltips (ou "Aide Ballon")

Les tooltips sont de petites fenêtres X affichant une chaîne d'aide à la position courante de la souris, typiquement au dessus de texte—dont la ligne

de mode—qui peut être activé avec la souris ou d'autres touches. (Cette facilité est parfois connue sous le nom de *aide ballon*.) Le texte d'aide peut aussi être diponible pour des entrées de menu.

Pour utiliser les tooltips, activez le mode Tooltip avec la commande `M-x tooltip-mode`. Le groupe de personnalisation `tooltip` contrôle les divers aspects du fonctionnement des tooltips. Lorsque le mode tooltip est désactivé, le texte d'aide est affiché dans la zone de répercussion.

Pour la version 21.1 d'Emacs, les tooltips ne sont pas supportés sous MS-Windows.

17.19 Évasion de la Souris

Le mode Évasion de Souris garde le pointeur de souris du système de fenêtres à bonne distance du point, pour empêcher d'obscurcir le texte. Lorsqu'il déplace la souris, il surélève aussi le cadre. Pour utiliser le mode Évasion de Souris, personnalisez l'option `mouse-avoidance-mode`. Vous pouvez la définir avec plusieurs valeurs pour déplacer la souris de différentes manières :

`banish` Déplace la souris dans le coin haut-droit à toute frappe de touche ;

`exile` Déplace la souris dans le coin si le curseur s'approche trop, et lui permet de revenir lorsque le curseur est sorti du chemin ;

`jump` Si le curseur s'approche trop de la souris, déplace la souris d'une distance et dans une direction au hasard ;

`animate` Comme `jump`, mais affiche les étapes du trajet pour une illusion de déplacement ;

`cat-and-mouse`
Comme `animate` ;

`proteus` Comme `animate`, mais change aussi la forme du pointeur de la souris.

Vous pouvez aussi utiliser la commande `M-x mouse-avoidance-mode` pour activer le mode.

17.20 Terminaux Sans Système de Fenêtres

Si votre terminal n'a pas de système de fenêtrage qu'Emacs supporte, il ne peut alors afficher qu'un cadre à la fois. Vous pouvez cependant toujours créer plusieurs cadres Emacs, et passer de l'un à l'autre. Passer d'un cadre

à l'autre sur ces terminaux est un peu comme passer d'une configuration de fenêtre à une autre.

Utilisez `C-x 5 2` pour créer un nouveau cadre et y passer ; utilisez `C-x 5 o` pour faire un cycle parmi les cadres existants ; utilisez `C-x 5 0` pour supprimer le cadre courant.

Chaque cadre a un numéro pour le distinguer. Si votre terminal peut afficher un seul cadre à la fois, le numéro du cadre sélectionné *n* apparaît près du début de la ligne de mode, sous la forme '`F`*n*'.

'`F`*n*' est actuellement le nom du cadre. Vous pouvez aussi, si vous préférez, spécifier un nom différent, et vous pouvez sélectionner un cadre par son nom. Utilisez la commande `M-x set-frame-name` ⟨RET⟩ *nom* ⟨RET⟩ pour spécifier un nouveau nom pour le cadre sélectionné, et utilisez `M-x select-frame-by-name` ⟨RET⟩ *nom* ⟨RET⟩ pour sélectionner un cadre selon son nom. Le nom que vous spécifiez apparaît dans la ligne de mode lorsque le cadre est sélectionné.

17.21 Utiliser le Souris dans des Émulateurs de Terminaux

Certains émulateurs de terminaux sous X supportent les clics souris dans la fenêtre du terminal. Dans un émulateur de terminal compatible avec `xterm`, vous pouvez utiliser `M-x xterm-mouse-mode` pour activer l'usage simple de la souris—seuls les simples clics sont autorisés. La fonctionnalité normale de la souris sous `xterm` est toujours disponible en gardant la touche `SHIFT` enfoncée lorsque vous pressez le bouton de la souris.

18 Support de Jeux de Caractères Internationaux

Emacs supporte une grande variété de jeux de caractères internationaux, dont les variantes Européenes de l'alphabet Latin, ou encore les scripts Chinois, Devanagari (Hindi et Marathi), Éthiopien, Grec, IPA, Japonais, Koréen, Lao, Russe, Thai, Tibétain, et Vietnamien. Ces fonctionnalités ont été fusionnées à partir de la version modifiée d'Emacs connue sous le nom de MULE (pour "MULti-lingual Enhancement to GNU Emacs")

18.1 Introduction aux Jeux de Caractères Internationaux

Les utilisateurs de ces scripts ont établi plusieurs systèmes de codage plus ou moins standards pour sauvegarder des fichiers. Emacs utilise en interne un unique encodage de caractères multi-octets, et peut ainsi mélanger des caractères provenant de tous ces scripts dans un seul tampon ou chaîne. Cet encodage représente chaque caractère non ASCII par une séquence d'octets dans l'intervalle 0200 à 0377. Emacs convertit entre l'encodage de caractères multi-octets et divers autres systèmes d'encodage lorsqu'il lit ou écrit un fichier, échange des données avec des sous-processus et (dans certains cas) dans la commande `C-q` (voir plus loin).

La commande `C-h h` (`view-hello-file`) affiche le fichier '`etc/HELLO`', qui montre comment dire "bonjour" dans plusieurs langues. Ceci illustre plusieurs scripts.

Les claviers, même dans les pays où ces jeux de caractères sont utilisés, n'ont généralement pas de touches pour tous les caractères. Pour cela, Emacs supporte plusieurs *méthodes d'entrée*, typiquement une pour chaque script ou langue, pour rendre leur frappe plus facile.

La touche préfixe `C-x` ⟨RET⟩ est utilisée pour les commandes relatives aux caractères multi-octets, systèmes d'encodage, et méthodes d'entrée.

18.2 Activer les Caractères Multi-Octets

Vous pouvez activer ou désactiver le support des caractères multi-octets, soit pour Emacs lui-même, soit pour un seul tampon. Lorsque les caractères multi-octets sont désactivés dans un tampon, chaque octet dans ce tampon représente alors un caractère, même les caractères entre 0200 et 0377. Les anciennes caractéristiques pour supporter les jeux de caractères européens, ISO Latin-1 et ISO Latin-2, fonctionnent comme avec Emacs 19.

Cependant, il n'est pas nécessaire de désactiver le support des caractères multi-octets pour utiliser ISO Latin-1 ou ISO Latin-2 ; le jeu de caractère multi-octet d'Emacs inclut tous les caractères de ces jeux de caractères, et Emacs peut convertir automatiquement vers et à partir de ces codes ISO.

La ligne de mode indique si le support des caractères multi-octets est activé pour le tampon courant. S'il est activé, deux-points apparaissent au début de la ligne de mode, précédant la place où les étoiles apparaissent lorsque le tampon est modifié. Lorsque les caractères multi-octets ne sont pas activés, les deux-points n'apparaissent pas.

La commande `C-x` ⟨RET⟩ `m` (`toggle-enable-multibyte-characters`) active ou désactive le support des caractères multi-octets pour le tampon courant.

Pour désactiver le support des caractères multi-octets par défaut, exécutez l'expression Lisp suivante :

```
(setq-default enable-multibyte-characters nil)
```

Lorsque les caractères multi-octets sont activés, les codes de caractères de 0200 (octal) à 0377 (octal) ne sont pas valides dans le tampon. Les caractères imprimables non ASCII valides ont des codes commençant à 0400.

Si vous tapez un caractère auto-insérable se trouvant dans l'intervalle invalide, Emacs suppose que vous essayez d'utiliser un des jeux de caractères ISO Latin*n*, et le convertit en code Emacs représentant ce caractère ISO Latin-*n*. Vous sélectionnez *quel* jeu de caractères ISO Latin utiliser à travers votre choix d'environnement de langue (see below).

La même chose arrive lorsque vous utilisez `C-q` pour entrer un code octal dans cet intervalle.

18.3 Environnements de Langues

Tous les jeux de caractères supportés sont supportés dans les tampons d'Emacs lorsque les caractères multi-octets sont activés ; il n'est pas nécessaire de sélectionner une langue particulière pour afficher ses caractères dans un tampon d'Emacs. Cependant, il est important de sélectionner un *environnement de langue* dans le but de définir diverses valeurs par défaut. L'environnement de langue représente réellement un choix de script préféré (plus ou moins) plutôt qu'un choix de langue.

L'environnement de langue contrôle quels systèmes d'encodage reconnaître à la lecture d'un texte (see Section 18.7 [Recognize Coding], page 225). Ceci s'applique à des fichiers, messages électroniques arrivés, messages de forums, et tout autre texte que vous lisez sous Emacs. Il peut aussi spécifier le système d'encodage à utiliser par défaut lorsque vous créez un fichier. Chaque environnement de langue spécifie aussi une méthode d'entrée par défaut.

La commande pour sélectionner un environnement de langue est `M-x set-language-environment`. Le tampon courant lorsque vous utilisez cette commande n'est pas important, car les effets s'appliquent en globalité à la session Emacs. Les environnements de langue supportés incluent :

> Chinese-BIG5, Chinese-CNS, Chinese-GB, Cyrillic-Alternativnyj, Cyrillic-ISO, Cyrillic-KOI8, Devanagari, English, Ethiopic, Greek, Hebrew, Japanese, Korean, Lao, Latin-1, Latin-2, Latin-3, Latin-4, Latin-5, Thai, Tibetan, and Vietnamese.

Certains systèmes d'exploitation vous permettent de spécifier la langue que vous utilisez en définissant des variables d'environnement locales. Emacs utilise une d'elles : si votre nom local pour le type de caractères contient la chaîne '`8859-`*n*', Emacs sélectionne automatiquement l'environnement de langue correspondant.

Pour afficher les informations sur les effets d'un environnement de langue donné *lang-env*, utilisez la commande `C-h L` *lang-env* ⟨RET⟩ (`describe-language-environment`). Ceci vous indique pour quelles langues cet environnement de langue est utile, et liste les jeux de caractères, les systèmes d'encodage, et les méthodes d'entrée qui vont avec. Elle montre aussi du texte d'exemple pour illustrer les scripts utilisés dans cet environnement de langue. Par défaut, cette commande décrit l'environnement de langue choisi.

18.4 Méthodes d'Entrée

Une *méthode d'entrée* est une sorte de conversion de caractère spécifiquement conçue pour des entrées interactives. Sous Emacs, typiquement chaque langue a sa propre méthode d'entrée ; parfois plusieurs langues qui utilisent les mêmes caractères peuvent partager une méthode d'entrée. Un certain nombre de langues supportent plusieurs méthodes d'entrée.

Le type le plus simple de méthode d'entrée fonctionne en convertissant des lettres ASCII dans un autre alphabet. C'est de cette manière que fonctionnent les méthodes d'entrée pour le Grec et le Russe.

Une technique plus puissante est la composition : convertir une séquence de caractères en une lettre. Plusieurs méthodes d'entrée européennes utilisent la composition pour produire une seule lettre non ASCII à partir d'une séquence consistant en une lettre suivie d'accents. Par exemple, certaines méthodes convertissent la séquence `'a` en une lettre unique accentuée.

Les méthodes d'entrée pour les scripts syllabiques utilisent typiquement la conversion suivie de la composition. Les méthodes d'entrée pour le Thaï et le Coréen fonctionnent de cette manière. D'abord, les lettres sont transformées en symboles pour des sons ou des marques de tonalité particuliers ; puis, des séquences de ceux-ci créant une syllabe entière sont transformés en un signe syllabique.

Le Chinois et le Japonais demandent des méthodes plus complexes. Dans les méthodes d'entrée pour le Chinois, vous entrez d'abord l'orthographe phonétique d'un mot Chinois (dans la méthode d'entrée `chinese-py`, parmi d'autres), ou une séquence de portions du caractère (méthodes d'entrée `chinese-4corner` et `chinese-sw`, et d'autres). Une orthographe phonétique correspondant typiquement à plusieurs caractères Chinois, vous devez choisir une des alternatives en utilisant des commandes d'Emacs spéciales. Les touches telles que `C-f`, `C-b`, `C-n`, `C-p`, et les chiffres ont des définitions spéciales dans cette situation, utilisées pour choisir parmi les alternatives. ⟨TAB⟩ affiche un tampon montrant toutes les possibilités.

Dans les méthodes d'entrée pour le Japonais, vous entrez d'abord un mot entier en utilisant l'orthographe phonétique ; puis, une fois que le mot est dans le tampon, Emacs le convertit en un ou plusieurs caractères en utilisant un gros dictionnaire. Une orthographe phonétique correspond à plusieurs mots japonais écrits différemment, et vous devez choisir l'un d'eux ; utilisez `C-n` et `C-p` pour parcourir les alternatives.

Il est parfois utile d'interrompre le traitement d'une méthode d'entrée pour que les caractères entrés jusque là ne se combinent pas avec les caractères suivants. Par exemple, dans la méthode d'entrée `latin-1-postfix`, la séquence `e '` se combine pour former un '`e`' avec un accent aigu. Comment faire si vous voulez entrer ces caractères séparément ?

Une manière est de taper l'accent deux fois ; ceci est une fonctionnalité particulière pour entrer la lettre et l'accent séparément. Par exemple, `e ' '` vous donne les deux caractères '`e'`'. Une autre manière est de taper une autre lettre après le `e`—une qui ne se combine pas—puis de la supprimer. Par exemple, vous pouvez taper `e e` ⟨DEL⟩ `'` pour obtenir '`e`' et '`'`' séparément.

Une autre méthode, plus générale mais pas aussi facile à taper, est d'utiliser `C-\ C-\` entre deux caractères pour qu'ils ne se combinent pas. C'est la commande `C-\` (`toggle-input-method`) utilisée deux fois.

`C-\ C-\` est spécialement utile durant une recherche incrémentale, car cette commande arrête l'attente de nouveaux caractères à combiner, et démarre la recherche de ce que vous avez déjà tapé.

Les variables `input-method-highlight-flag` et `input-method-verbose-flag` contrôlent comment les méthodes d'entrée expliquent ce qui arrive. Si `input-method-highlight-flag` est non `nil`, la séquence partielle est mise en surbrillance dans le tampon. Si `input-method-verbose-flag` est non `nil`, la liste des caractères possibles à taper ensuite est affichée dans la zone de répercussion (sauf lorsque vous êtes dans le mini-tampon).

18.5 Sélectionner une Méthode d'Entrée

`C-\` Permet ou non l'utilisation de la méthode d'entrée sélectionnée.

`C-x` ⟨RET⟩ `C-\` *méthode* ⟨RET⟩
: Sélectionne une nouvelle méthode d'entrée pour le tampon courant.

`C-h I` *méthode* ⟨RET⟩
`C-h C-\` *méthode* ⟨RET⟩
: Décrit la méthode d'entrée *méthode* (`describe-input-method`). Par défaut, elle décrit la méthode d'entrée courante (lorsqu'elle est définie).

`M-x list-input-methods`
: Affiche une liste de toutes les méthodes d'entrée supportées.

Pour choisir une méthode d'entrée pour le tampon courant, utilisez `C-x` ⟨RET⟩ `C-\` (`select-input-method`). Cette commande lit le nom de la méthode d'entrée dans le mini-taaampon ; le nom commence normalement par l'environnement de langue qui est utilisé avec. La variable `current-input-method` enregistre quelle méthode d'entrée est sélectionnée.

Les méthodes d'entrée utilisent diverses séquences de caractères ASCII qui correspondent à des caractères non ASCII. Il est parfois utile d'arrêter temporairement la méthode d'entrée. Pour cela, tapez `C-\` (`toggle-input-method`). Pour relancer la méthode d'entrée, tapez `C-\` de nouveau.

Si vous tapez `C-\` et n'avez pas encore sélectionné une méthode d'entrée, il vous est demandé d'en spécifier une. Ceci a le même effet que d'utiliser `C-x` ⟨RET⟩ `C-\` pour spécifier une méthode d'entrée.

La sélection d'un environnement de langue spécifie une méthode d'entrée par défaut à utiliser dans les différents tampons. Lorsque vous avez une méthode d'entrée par défaut, vous pouvez la sélectionner dans le tampon courant en tapant `C-\`. La variable `default-input-method` spécifie la méthode d'entrée par défaut (`nil` indique qu'il n'y en a pas).

Certaines méthodes d'entrée pour les scripts alphabétiques fonctionnent (en pratique) en redéfinissant le clavier pour émuler divers arrangements de claviers communément utilisés pour ces scripts. La manière de redéfinir correctement dépend de l'arrangement de votre clavier actuel. Pour spécifier quel arrangement a votre clavier, utilisez la commande `M-x quail-set-keyboard-layout`.

Pour afficher une liste de toutes les méthodes d'entrée supportées, tapez `M-x list-input-methods`. La liste donne des informations sur chaque méthode d'entrée, dont la chaîne qui apparaît pour elle dans la ligne de mode.

18.6 Systèmres de Codage

Les utilisateurs de langues diverses ont établi plusieurs systèmes de codage plus ou moins standards pour les représenter. Emacs n'utilise pas en

interne ces systèmres de codage ; il convertit plutôt depuis les divers systèmes de codage vers son propre système lorsqu'il lit des données, et convertit le système de codage interne en d'autres systèmes de codage lorsqu'il écrit des données. La conversion est possible pour la lecture ou l'écriture de fichiers, la transmission ou la réception depuis le terminal, et l'échange de données avec des sous-procesus.

Emacs assigne un nom à chaque système de codage. La plupart des systèmes de codage sont utilisés pour une langue, et le nom du système de codage commence avec le nom de la langue. Certains systèmes de codage sont utilisés par plusieurs langues ; leurs noms commencent normalement par '`iso`'. Il y a aussi les systèmes de codage spéciaux `no-conversion`, `raw-text` et `emacs-mule` qui ne convertissent pas du tout les caractères imprimables.

En plus de convertir différentes représentations de caractères non ASCII, un système de codage peut aussi réaliser des conversion de fin-de-ligne. Emacs retient trois conventions différentes pour séparer les lignes dans les fichiers : newline, carriage-return linefeed, et seulement carriage-return.

`C-h C` *codage* ⟨RET⟩
: Décrit le système de codage *codage*.

`C-h C` ⟨RET⟩
: Décrit les systèmes de codage actuellement utilisés.

`M-x list-coding-systems`
: Affiche une liste de tous les systèmes de codage supportés.

La commande `C-h C` (`describe-coding-system`) affiche des informations sur des systèmes de codage particuliers. Vous pouvez spécifier le nom d'un système de codage en argument ; alternativement, avec un argument vide, elle décrit les systèmes de codage couramment sélectionnés pour divers usages, dans le tampon courant et comme défaut, et la liste de priorité pour reconnaître les systèmes de codage (see Section 18.7 [Recognize Coding], page 225).

Pour afficher une liste de tous les systèmes de codage supportés, tapez `M-x list-coding-systems`. La liste donne des informations sur chaque système de codage, dont la lettre qui le représente dans la ligne de mode. (see Section 1.3 [Mode Line], page 26).

Chacun des systèmes de codage qui apparaissent dans cette liste—à l'exception de `no-conversion`, qui veut dire aucune conversion d'aucune sorte—spécifie comment convertir les caractères imprimables, mais laisse le choix de la conversion de fin-de-ligne être décidé selon le contenu de chaque fichier. Par exemple, si le fichier semble utiliser carriage-return linefeed entre les lignes, alors la conversion de fin-de-ligne sera utilisée.

Chacun des systèmes de codage listés a trois variantes qui spécifient exactement quoi faire pour la conversion de fin-de-ligne :

`...-unix` Ne fait aucune conversion de fin de ligne ; suppose que le fichier utilise newline pour séparer les lignes. (C'est la convention normlement utilisée sur les systèmes Unix et GNU.)

`...-dos` Suppose que le fichier utilise carriage-return linefeed pour séparer les lignes, et fait la conversion appropriée. (C'est la convention normalement utilisée sur les systèmes Microsoft.)

`...-mac` Suppose que le fichier utilise carriage-return pour séparer les lignes, et fait la conversion appropriée. (C'est la conversion normalement utilisée sur le système Macintosh.)

Ces variantes de systèmes de codage sont omises de l'affichage de `list-coding-systems` pour raison de brièveté, car elles sont tout à fait prévisibles. Par exemple, le système de codage `iso-latin-1` a des variantes `iso-latin-1-unix`, `iso-latin-1-dos` et `iso-latin-1-mac`.

Le système de codage `raw-text` est utile pour un fichier texte presque entièrement ASCII, mais pouvant contenir des octets dont la valeur est inférieure à 127 et ne servant pas à encoder les caractères non ASCII. Avec `raw-text`, Emacs copie sans les changer ces octets, et met `enable-multibyte-characters` à `nil` dans le tampon courant pour qu'ils soient correctement interprétés. `raw-text` utilise la conversion fin-de-ligne de la manière habituelle, en se basant sur les données rencontrées, et a les trois variantes habituelles pour spécifier le style de conversion de fin-de-ligne à utiliser.

Au contraire, le système de codage `no-conversion` ne spécifie aucun code de caractère de conversion—aucun pour les valeurs d'octets non ASCII et aucun pour les fin-de-ligne. Il est utile pour lire ou écrire des fichiers binaires, des fichiers tar, et d'autres fichiers qui doivent être examinés verbatim. Il met, lui aussi, `enable-multibyte-characters` à `nil`.

La manière la plus facile d'éditer un fichier sans conversion d'aucune sorte est d'utiliser la commande `M-x find-file-literally`. Elle utilise `no-conversion`, et supprime aussi d'autres fonctionnalités d'Emacs qui pourraient convertir le contenu du fichier avant qu'il soit affiché. See Section 14.2 [Visiting], page 145.

Le système de codage `emacs-mule` veut dire que le fichier contient des caractères non ASCII stockés avec l'encodage interne d'Emacs. Il utilise la conversion de fin-de-ligne selon les données rencontrées, et a les trois variantes habituelles pour spécifier le type de conversion de fin-de-ligne.

18.7 Reconnaissance de Systèmes de Codage

La plupart du temps, Emacs peut reconnaître quel système de codage utiliser pour un fichier donné—une fois que vous avez spécifié vos préférences.

Certains systèmes de codae peuvent être reconnus ou distingués par les séquences d'octets apparaissant dans les données. Cependant, des systèmes de codage ne peuvent être distingués, même partiellement. Par exemple, il n'y a pas de moyen de distinguer entre Latin-1 et Latin-2 ; ils utilisent les mêmes valeurs d'octets avec différentes significations.

Emacs traite cette situation grâce à une liste de priorité de systèmes de codage. Lorsqu'Emacs lit un fichier, si vous ne spécifiez pas le système de codage à utiliser, Emacs compare les données avec chaque système de codage, en commençant par le premier en priorité et en descendant la liste, jusqu'à ce qu'il trouve un système de codage s'accordant avec les données. Il convertit alors le contenu du fichier en supposant qu'il est représenté dans ce système de codage.

La liste de priorité de systèmes de codage dépend de l'environnement de langue sélectionné (see Section 18.3 [Language Environments], page 220). Par exemple, si vous utilisez le français, vous voulez certainement qu'Emacs préfère Latin-1 à Latin-2 ; si vous utilisez le tchèque, vous voulez certainement que Latin-2 soit préféré. C'est une des raisons de spécifier un environnement de langue.

Cependant, vous pouvez modifier la liste de priorité en détail avec la commande `M-x prefer-coding-system`. Cette commande lit le nom d'un système de codage depuis le mini-tampon, et l'ajoute au début de la liste de priorité, pour qu'il soit préféré à tous les autres. Si vous utilisez cette commande plusieurs fois, chaque utilisation ajoute un élément au début de la liste de priorité.

Parfois un nom de fichier indique quel système de codage utiliser pour ce fichier. La variable `file-coding-system-alist` spécifie cette correspondance. Il existe une fonction spéciale `modify-coding-system-alist` pour ajouter des éléments à cette liste. Par exemple, pour lire et écrire tous les '`.txt`' en utilisant le système de codage `china-iso-8bit`, vous pouvez utilliser cette expression Lisp :

```
(modify-coding-system-alist 'file "\\.txt\\'" 'china-iso-8bit)
```

Le premier argument doit être `file`, le second argument doit être une expression rationnelle qui détermine à quels fichiers l'appliquer, et le troisième argument indique quel système de codage utiliser pour ces fichiers.

Vous pouvez spécifier le système de codage pour un fichier en particulier en utilisant la construction '`-*-...-*-`' en début de fichier, ou une liste de variables locales à la fin (see Section 31.2.5 [File Variables], page 468). Vous faites cela en définissant une valeur pour une "variable" `coding` ; plutôt que de définir une variable, cela spécifie le système de codage pour le fichier. Par exemple, '`-*-mode: C; coding: latin-1;-*-`' indique d'utiliser le système de codage Latin-1, ainsi que le mode C.

Une fois qu'Emacs a choisi un système de codage pour un tampon, il stocke ce système de codage dans `buffer-file-coding-system` et utilise ce système de codage, par défaut, pour les opérations qui écrivent à partir de ce

tampon dans un fichier. Cela inclut les commandes `save-buffer` et `write-region`. Si vous désirez écrire des fichiers à partir de ce tampon en utilisant un système de codage différent, vous pouvez spécifier un autre système de codage pour ce tampon en utilisant `set-buffer-file-coding-system` (see Section 18.8 [Specify Coding], page 227).

Lorsque vous envoyez un message avec le mode Mail (see Chapter 26 [Sending Mail], page 357), Emacs a quatre manières différentes de déterminer le système de codage à utiliser pour encoder le texte du message. Il essaie la valeur pour le tampon lui-même de `buffer-file-coding-system`, si elle est non `nil`. Autrement, il utilise la valeur de `sendmail-coding-system`, si elle est non `nil`. Le troisième moyen est d'utiliser le système de codage par défaut pour les nouveaux fichiers, qui est contrôlé par votre choix d'environnement de langue, si elle est non `nil`. Si ces trois valeurs sont `nil`, Emacs encode le message à envoyer en utilisant le système de codage Latin-1.

Lorsque des messages arrivent pour vous dans Rmail, chaque message est automatiquement traduit à partir du système de codage dans lequel il est écrit—comme s'il était dans un fichier séparé. Ceci utilise la liste de priorité de systèmes de codage que vous avez spécifié.

Pour lire et écrire les fichiers Rmail eux-mêmes, Emacs utilise le système de codage spécifié par la variable `rmail-file-coding-system`. La valeur par défaut est `nil`, ce qui indique que les fichiers Rmail ne sont pas convertis (ils sont lus et écrits dans le code de caractères interne d'Emacs).

18.8 Specifying a Coding System

Dans les cas où Emacs ne choisit pas automatiquement le bon système de codage, vous pouvez utiliser ces commandes pour en spécifier un :

`C-x` ⟨RET⟩ `f` *codage* ⟨RET⟩
: Utilise le système de codage *codage* pour le fichier visité dans le tampon courant.

`C-x` ⟨RET⟩ `c` *codage* ⟨RET⟩
: Spécifie le système de codage *codage* pour la commande suivant immédiatement.

`C-x` ⟨RET⟩ `k` *codage* ⟨RET⟩
: Utilise le système de codage *codage* pour les entrées au clavier.

`C-x` ⟨RET⟩ `t` *codage* ⟨RET⟩
: Utilise le système de codage *codage* pour les sorties sur le terminal.

`C-x` ⟨RET⟩ `p` *codage* ⟨RET⟩
: Utilise le système de codage *codage* pour les entrées et sorties avec les sous-processus du tampon courant.

La commande `C-x RET f` (`set-buffer-file-coding-system`) spécifie le système de codage pour le fichier du tampon courant—en d'autres mots, quel système de codage utiliser pour enregistrer ou lire le fichier visité. Vous spécifiez le système de codage en utilisant le mini-tampon. Cette commande s'appliquant à un fichier déjà visité, elle affecte seulement la manière dont le fichier est enregistré.

Un autre moment pour spécifier le système de codage pour un fichier est lorsque vous visitez le fichier. Utilisez d'abord la commande `C-x` ⟨RET⟩ `c` (`universal-coding-system-argument`) ; cette commande utilise le mini-tampon pour lire le nom d'un système de codage. Après avoir quitté le mini-tampon, le système de codage spécifié est utilisé pour *la commande suivant immédiatement*.

De ce fait si la commande suivant immédiatement est `C-x C-f`, par exemple, elle lit le fichier en utilisant ce système de codage (et sauvegarde le système de codage pour le moment où le fichier est enregisté). Ou si la commande suivant immédiatement est `C-x C-w`, elle écrit le fichier en utilisant ce système de codage. Les autres commandes de fichiers affectées par un système de codage spécifique incluent `C-x C-i` et `C-x C-v`, ainsi que les variantes autre-fenêtre de `C-x C-f`.

`C-x` ⟨RET⟩ `c` affecte aussi les commandes qui lancent des sous-processus, incluant including `M-x shell` (see Section 30.2 [Shell], page 425).

Cependant, si la commande suivant immédiatement n'utilise pas de système de codage, alors `C-x` ⟨RET⟩ `c` n'a aucun effet.

Une manière facile de visiter un fichier sans aucune conversion est d'utiliser la commande `M-x find-file-literally`. See Section 14.2 [Visiting], page 145.

La variable `default-buffer-file-coding-system` spécifie le choix de système de codage à utiliser lorsque vous créez un nouveau fichier. Elle est utilisée lorsque vous ouvrez un nouveau fichier, et lorsque vous créez un nouveau tampon puis l'enregistrez dans un fichier. Spécifier un environnement de langue met cette variable à un bon choix de système de codage par défaut pour cet environnement de langue.

La commande `C-x` ⟨RET⟩ `t` (`set-terminal-coding-system`) spécifie le système de codage pour les sorties sur le terminal. Si vous spécifiez un code de caractères pour les sorties sur le terminal, tous les caractères envoyés au terminal sont convertis dans ce système de codage.

Cette fonctionnalité est utile pour certains terminaux texte seulement construits pour supporter des langues ou jeux de caractères spécifiques—par exemple, des terminaux européens supportant un des jeux de caractères ISO Latin.

Par défaut, les sorties sur le terminal ne sont pas converties.

La commande `C-x` ⟨RET⟩ `k` (`set-keyboard-coding-system`) spécifie le système de codage pour les entrées au clavier. La conversion par un code de caractères des entrées clavier est utile pour des terminaux avec des touches

qui envoient des caractères graphiques non ASCII—par exemple, certains terminaux construits pour ISO Latin-1 ou un sous-ensemble de celui-ci.

Par défaut, les entrées clavier ne sont pas converties.

Il existe une similarité entre l'utilisation d'un système de codage pour les entrées au clavier, et l'utilisation d'une méthode d'entrée : tous deux définissent des séquences d'entrées au clavier qui sont converties en caractères uniques. Cependant, les méthodes d'entrée sont étudiées pour être utilisées facilement par les humains de façon interactive, et les séquences converties sont généralement des séquences de caractères imprimables ASCII. Les systèmes de codage, quant à eux, convertissent généralement des séquences de caractères non graphiques.

La commande `C-x ⟨RET⟩ p` (`set-buffer-process-coding-system`) spécifie le système de codage pour les entrées et sorties avec un sous-processus. Cette commande s'applique au tampon courant ; normalement, chaque sous-processus a son propre tampon, et vous pouvez ainsi utiliser cette commande pour spécifier la conversion à partir de et vers un sous-processus en particulier en exécutant cette commande dans le tampon correspondant.

Par défaut, les entrées et sorties avec un processus ne sont pas converties.

La variable `file-name-coding-system` spécifie un système de codage utilisé pour encoder les noms de fichiers. Si vous définissez cette variable comme un nom de système de codage (comme symbole Lisp ou comme chaîne), Emacs encode les noms de fichiers en utilisant ce système de codage pour toutes les opérations sur les fichiers. Cela rend possible l'utilisation des caractères non ASCII dans les noms de fichiers—ou, au moins, les caractères non ASCII que le système de codage spécifié peut encoder. Par défaut, cette variable est à `nil`, ce qui implique que vous ne pouvez pas utiliser de caractères non ASCII dans les noms de fichiers.

18.9 Jeux de Polices

Une police pour X Window définit typiquement des formes graphiques pour un seul alphabet ou un script. Par conséquent, l'affichage de l'éventail entier de scripts supportés par Emacs nécessite une collection de plusieurs polices. Sous Emacs, une telle collection est appelée un *jeu de polices*. Un jeu de polices est défini par une liste de polices, chacune représentant un éventail de codes de caractères.

Chaque jeu de polices a un nom, comme une police. Les polices X disponibles sont définies par le serveur X ; les jeux de polices, cependant, sont définis sous Emacs lui-même. Une fois que vous avez défini un jeu de polices, vous pouvez l'utiliser sous Emacs en utilisant son nom, partout où vous pourriez utiliser une police unique. Bien sûr, les jeux de polices d'Emacs peuvent seulement utiliser les polices que le serveur X supporte ; si

certains caractères apparaissent à l'écran comme des rectangles vides, cela veut dire que le jeu de police en usage n'a pas de police pour ces caractères.

Emacs crée automatiquement deux jeux de polices : le *jeu de polices standard* et le *jeu de polices de départ*. Le jeu de police standard a le plus de chance d'avoir des polices pour une grande variété de caractères non ASCII ; cependant, ce n'est pas celui qu'Emacs utilise par défaut. (Par défaut, Emacs essaie de trouver une police qui a des variantes grasses et italiques.) Vous pouvez spécifier l'utilisation du jeu de police standard en démarrant Emacs de cette manière :

```
emacs -fn fontset-standard
```

Un jeu de polices ne spécifie pas nécessairement une police pour tous les codes de caractères. Si un jeu de polices ne spécifie pas de polices pour un certain caractère, ou s'il spécifie une police qui n'existe pas sur votre système, il ne peut alors afficher ce caractère. Il affichera un rectangle vide à la place.

La largeur et la hauteur d'un jeu de polices sont déterminés par les caractères ASCII (c'est-à-dire, par la police utilisée pour les caractères ASCII dans ce jeu de polices). Si une autre police dans le jeu a une hauteur ou une largeur différente, alors les caractères assignés à cette police sont ramenés à la taille du jeu de polices. Si `highlight-wrong-size-font` est non `nil`, un rectangle est affiché autour de ces caractères de mauvaise taille.

18.10 Définir des Jeux de Polices

Emacs crée automatiquement un jeu de polices standard selon la valeur de `standard-fontset-spec`. Le nom de ce jeu de polices est

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

ou seulement '`fontset-standard`' pour simplifier.

Les variantes grasses, italiques, et grasses-italiques du jeu de polices standard sont créés automatiquement. Leurs noms ont '`bold`' à la place de '`medium`', ou '`i`' à la place de '`r`', ou les deux.

Si vous spécifiez une police ASCII par défaut avec la ressource '`Font`' ou l'argument '`-fn`', Emacs génère automatiquement un jeu de polices à partir de celle-ci. C'est le *jeu de polices de départ* et son nom est `fontset-startup`. Il fait ceci en remplaçant les champs *fonderie*, *famille*, *style*, et *largeur_moyenne* de la police avec '`*`', en remplaçant le champ *registre_jeu_de_caractères* avec '`fontset`', en remplaçant le champ *encodage_jeu_de_caractères* avec '`startup`', puis en utilisant la chaîne résultante pour spécifier un jeu de polices.

Ainsi, si vous démarrez Emacs de cette manière,

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

Emacs génère le jeu de polices suivant et l'utilise pour la fenêtre X du cadre initial :

```
-*-*-medium-r-normal-*-14-140-*-*-*-*-fontset-startup
```

Avec la ressource X '`Emacs.Font`', vous pouvez spécifier un nom de jeu de polices exactement comme pour un nom de police. Mais faites attention de ne pas spécifier un nom de jeu de polices dans une ressource générique comme '`Emacs*Font`'—cette spécification générique s'applique à diverses autres utilisations, comme les menus, et les menus ne peuvent pas accepter de jeux de polices.

Vous pouvez spécifier des jeux de polices additionnels en utilisant les ressources X appelées '`Fontset-`*n*', où *n* est un entier commençant à 0. La valeur de la ressource doit avoir cette forme :

motif_police`,` [*nom_jeu_de_caractères*`:`*nom_polices*]...

motif_police doit avoir la forme d'un nom de police X standard, sauf pour les deux derniers champs. Ils doivent avoir la forme '`fontset-`*alias*'.

Le jeu de polices a deux noms, un long et un court. Le nom long est *motif_police*. Le nom court est '`fontset-`*alias*'. Vous pouvez vous référer au jeu de caractères par l'un ou l'autre nom.

La construction '*jeu_de_caractères*`:`*police*' spécifie quelle police utiliser (dans ce jeu de polices) pour un jeu de caractères particulier. Ici, *jeu_de_caractères* est le nom d'un jeu de caractères, et *police* est la police à utiliser pour ce jeu de caractères. Vous pouvez utiliser cette construction un nombre quelconque de fois dans la définition d'un jeu de polices.

Pour les autres jeux de caractères, Emacs choisit une police basée sur *motif_police*. Il remplace '`fontset-`*alias*' avec les valeurs décrivant le jeu de caractères. Pour la police de caractères ASCII, '`fontset-`*alias*' est remplacé par '`ISO8859-1`'.

De plus, lorsque plusieurs champs consécutifs sont des jokers, Emacs les rassemble en un seul joker. Ceci est prévu pour éviter l'usage de polices auto-redimensionnées. Des polices créées en redimensionannt des polices plus grandes ne sont pas utilisables pour l'édition, et redimensionner des polices plus petites n'est pas utile car il est préférable d'utiliser la petite police dans sa taille réelle, ce que fait Emacs.

Par exemple, si *motif_police* est,

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

la spécification de police pour des caractères ASCII serait :

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

et la spécification de police pour les caractères chinois GB2312 serait :

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

Vous pouvez n'avoir aucune police chinoise correspondant à la spécification de police précédente. La plupart des distributions X incluent seulement des polices chinoises ayant '`song ti`' ou '`fangsong ti`' dans le champ *famille* Dans un tel cas, '`Fontset-`*n*' peut être spécifié ainsi :

```
Emacs.Fontset-0: -*-fixed-medium-r-normal-*-24-*-*-*-*-*-fontset-24,\
        chinese-gb2312:-*-*-medium-r-normal-*-24-*-gb2312*-*
```

Ainsi, les spécifications de polices pour tous les caractères sauf ceux chinois ont '`fixed`' dans le champ *famille*, et la spécification de police pour les caractèrs chinois GB2312 a un joker '`*`' dans le champ *famille*.

La fonction qui traite la valeur de la ressource Fontset pour créer le jeu de polices est appelée `create-fontset-from-fontset-spec`. Vous pouvez aussi appeler explicitement cette fonction pour créer un jeu de polices.

See Section B.7 [Font X], page 516, pour plus d'informations sur les noms de polices X.

18.11 Support de Caractères Européens Mono-Octets.

Les jeux de caractères ISO 8859 Latin-*n* définissent les codes de caractères dans l'intervalle 160 à 255 pour supporter les lettres accentuées et la ponctuation nécessaires pour diverses langues européennes. Si vous interdisez l'usage de caractères multi-octets, Emacs peut toujours supporter *un* de ces jeux de caractères à la fois. Pour spécifier *lequel* de ces codes utiliser, appelez `set-language-environment` et spécifiez '`Latin-`*n*'.

Emacs peut afficher ces caractères dans la mesure où le terminal ou la police en usage le permet. Utilisez la commande `M-x standard-display-european` pour permettre ou interdire le mode d'affichage européen. Avec un argument numérique, `M-x standard-display-european` permet l'affichage de caractères européens si et seulement si l'argument est positif.

Si votre terminal ne permet pas l'affichage du jeu de caractères Latin-1, Emacs peut afficher ces caractères sous la forme de séquences ASCII qui vous donnent au moins une idée claire des caractères réels. Pour cela, chargez la bibliothèque `iso-ascii`. Des bibliothèques similaires pour d'autres jeux de caractères Latin-*n* pourraient être développés, mais cela n'a pas encore été fait.

Il existe trois moyens différents de saisir des caractères Latin-*n* mono-octets :

- Si votre clavier peut générer des codes de caractères 128 et supérieurs, représentant les caractères ISO Latin-*n* characters, exécutez l'expression suivante pour permettre à Emacs de les comprendre :

```
(set-input-mode (car (current-input-mode))
                (nth 1 (current-input-mode))
                0)
```

- Pour Latin-1 seulement, vous pouvez charger la bibliothèque `iso-transl` pour que la touche `C-x 8` soit un préfixe "caractère composé" pour entrer des caractères imprimables Latin-1 non ASCII. `C-x 8` fonctionne pour l'insertion (dans le mini-tampon aussi bien que dans less autres tampons), pour la recherche, et dans tout autre contexte où les séquences de touches sont permises. La touche modificatrice ⟨ALT⟩, si

vous en avez une, est utilisé comme `C-x 8` ; utilisez ⟨ALT⟩ avec un carctère accent pour modifier la lettre suivante.

- Vous pouvez utilisez le mode Accents ISO. Lorsque ce mode mineur est lancé, les caractères ‘`‘`’, ‘`’`’, ‘`"`’, ‘`^`’, ‘`/`’, ‘`,`’ et ‘`~`’ modifient la lettre suivante en y ajoutant, si possible, le signe diacritique correspondant. Pour lancer ou arrêter le mode Accents ISO, utilisez la commande `M-x iso-accents-mode`. Cette commande affecte seulement le tampon courant.

  Pour insérer un de ces caractères accent depuis le mode Accents ISO, tapez le caractère, suivi d'un espace. Certains de ces caractères ont un caractère accent “touche morte” correspondant dans certains jeux de carctères ISO Latin ; pour entrer un caractère “touche morte”, tapez le caractère ASCII correspondant deux fois. Par exemple, dans Latin-1, `''` insére le caractère Latin-1 accent aigu (code octal 0264).

  L'utilisation du mode Accents ISO est disponible lorsqu'une séquence de touches est attendue : pour l'insertion ordinaire, pour la recherche, pour le mini-tampon, et pour certains arguments de commandes.

  En plus des lettres accentuées, vous pouvez utiliser ces séquences spéciales dans le mode Accents ISO pour insérer certains autres caractères ISO Latin-1 :

  `/A`, ‘`A`’ avec anneau. `~C`, ‘`C`’ cédille. `~D`, ‘`D`’ coupé. `/E`, ‘`AE`’ liés.
  `/a`, ‘`a`’ avec anneau. `~c`, ‘`c`’ cédille. `~d`, ‘`d`’ coupé. `/e`, ‘`ae`’ liés.
  `"s`, ‘`s`’ allemand. `~<`, guillemet ouvrant. `~>`, guillemet fermant.
  `~!`, point d'exclamation inversé. `~?`, point d'interrogation inversé.

19 Modes Majeurs

Emacs propose plusieurs *modes majeurs* alternatifs, chacun personnalisant Emacs pour éditer du texte d'un type particulier. Les modes majeurs sont mutuellement exclusifs, et chaque tampon a à chaque moment un mode majeur. La ligne de mode indique normalement le nom du mode majeur en cours, entre parenthèses (see Section 1.3 [Mode Line], page 26).

Le mode majeur le moins spécialisé est appelé *Fundamental mode*. Ce mode n'a aucune redéfinition spécifique au mode ou définition de variable, ainsi chaque commande d'Emacs fonctionne de la manière la plus générale, et chaque option est dans son état par défaut. Pour éditer du texte d'un type spécifique connu par Emacs, comme du code Lisp ou du texte français, vous devez passer dans le mode approprié, comme le mode Lisp ou le mode Texte.

La sélection d'un mode majeur change la signification de quelques touches pour qu'elles soient plus particulièrement adaptées au langage édité. Celles modifiées fréquemment sont ⟨TAB⟩, ⟨DEL⟩, et `C-j`. La touche préfixe `C-c` contient normalement les commandes spécifiques au mode. De plus, les commandes qui travaillent sur les commentaires utilisent le mode pour déterminer comment les commentaires doivent être délimités. Plusieurs modes majeurs redéfinissent les propriétés syntaxiques des caractères apparaissant dans le tampon. See Section 31.6 [Syntax], page 485.

Les modes majeurs tombent dans trois groupes majeurs. Le mode Lisp (qui a plusieurs variantes), le mode C, le mode Fortran et autres sont utilisés pour des langages de programmation spécifiques. Le mode Texte, le mode Nroff, le mode SGML, le mode TEX et le mode Profil sont pour du texte normal, brut ou balisé. Les autres modes majeurs ne sont pas prévus pour être utilisés sur des fichiers d'utilisateurs ; ils sont utilisés dans des tampons créés par Emacs pour un usage spécifique, comme le mode Dired pour les tampons créés par Dired (see Chapter 28 [Dired], page 387), le mode Mail pour les tampons créés par `C-x m` (see Chapter 26 [Sending Mail], page 357), et le mode Shell pour les tampons utilisés pour communiquer avec un processus shell fils (see Section 30.2.2 [Interactive Shell], page 427).

La plupart des modes majeurs pour langage de programmation spécifient que seules les lignes vierges séparent les paragraphes. Cela permet de rendre utilisables les commandes relatives aux paragraphes. (See Section 21.3 [Paragraphs], page 246.) Ils indiquent aussi au mode Auto Fill d'utiliser la définition de ⟨TAB⟩ pour indenter les nouvelles lignes qu'il crée. Cela car la plupart des lignes d'un programme sont habituellement indentées (See Chapter 20 [Indentation], page 239.)

19.1 Comment les Modes Majeurs Sont Choisis

Vous pouvez explicitement sélectionner un mode majeur pour le tampon courant, mais la plupart du temps Emacs détermine quel mode utiliser en se basant sur le nom du fichier ou sur du texte spécial dans le fichier.

La sélection explicite d'un nouveau mode majeur est fait avec la commande `M-x`. Au nom du mode majeur, ajoutez `-mode` pour obtenir le nom de la commande sélectionnant ce mode. Ainsi, vous pouvez entrer dans le mode Lisp en exécutant `M-x lisp-mode`.

Lorsque vous visitez un fichier, Emacs choisit habituellement le bon mode majeur en se basant sur le nom du fichier. Par exemple, les fichiers dont les noms finissent par '`.c`' sont édités dans le mode C. Les correspondances entre les noms de fichiers et les modes majeurs sont contrôlées par la variable `auto-mode-alist`. Sa valeur est une liste dans laquelle chaque élément a cette forme :

(*regexp* `.` *mode-function*)

ou cette forme :

(*regexp* *mode-function* *flag*)

Par exemple, un élément généralement touvé dans cette liste a la forme `("\\.c\\'" . c-mode)`, et sélectionne le mode C pour les fichiers dont les noms finissent par '`.c`'. (Notez que '`\\`' est nécessaire dans la syntaxe Lisp pour inclure un '`\`' dans une chaîne, qui est nécessaire pour supprimer la signification spéciale de '`.`' dans les expressions rationnelles.) Si l'élément a la forme (*regexp mode-function flag*) et *flag* est non `nil`, alors après avoir appelé *mode-function*, le suffixe correspondant à *regexp* est enlevé et la liste est de nouveau parcourue pour rechercher une nouvelle correspondance.

Vous pouvez spécifier quel mode majeur doit être utilisé pour éditer un certain fichier par un texte particulier sur la première ligne non vierge de ce fichier. Le nom du mode doit apparaître sur cette ligne précédé et suivi de '`-*-`'. Du texte supplémentaire peut apparaître sur cette même ligne. Par exemple,

```
;-*-Lisp-*-
```

indique à Emacs d'utiliser le mode Lisp. Une telle spécification explicite est prioritaire par rapport à tout défaut basé sur le nom du fichier. Notez l'utilisation du point-virgule pour que Lisp traite cette ligne comme un commentaire.

Un autre format de spécification de mode est

`-*- mode:` *nom_mode*`;-*-`

qui vous permet aussi de spécifier des variables locales, de cette manière :

`-*- mode:` *nom_mode*`;` *var*`:` *valeur*`; ... -*-`

See Section 31.2.5 [File Variables], page 468, pour plus d'informations sur ce sujet.

Lorsque le contenu d'un fichier commence par '`#!`', il peut être utilisé comme une commande shell exécutable, qui fonctionne en exécutant un in-

terpréteur nommé dans la première ligne du fichier. La suite du fichier est utilisée comme entrée de l'interpréteur.

Lorsque vous visitez un tel fichier avec Emacs, si le nom du fichier ne spécifie pas un mode majeur, Emacs utilise le nom de l'interpréteur de la première ligne pour choisir un mode. Si la première ligne est le nom d'un interpréteur reconnu, comme '`perl`' ou '`tcl`', Emacs utilise le mode approprié pour cet interpréteur. La variable `interpreter-mode-alist` spécifie les correspondances entre les noms d'interpréteurs et les modes majeurs.

Lorsque la première ligne commence avec '`#!`', vous ne pouvez pas (sur un grand nombre de systèmes) utiliser la fionctionnalité '`-*-`' sur la première ligne, car le système ferait une erreur en lançant l'interpréteur. Emacs recherche alors '`-*-`' sur la seconde ligne aussi bien que sur la première dans de tels fichiers.

Losque vous visitez un fichier qui ne spécifie pas le mode majeur à utiliser, ou lorsque vous créez un nouveau tampon avec `C-x b`, la variable `default-major-mode` spécifie quel mode majeur utiliser. Normalement, sa valeur est `fundamental-mode`, qui indique le mode Fondamental. Si `default-major-mode` est `nil`, le mode majeur est le même que celui du tampon précédemment sélectionné.

Si vous changez le mode majeur d'un tampon, vous pouvez revenir au mode majeur qu'Emacs choisit automatiquement : utilisez la commande `M-x normal-mode` pour ce faire. C'est la même fonction que `find-file` appelle pour choisir un mode majeur. Elle analyse aussi les variables locales au fichier, s'il y en a.

Les commandes `C-x C-w` et `set-visited-file-name` modifient le mode majeur si le nom du fichier spécifient un nouveau mode (see Section 14.3 [Saving], page 148). Cependant, ceci n'arrive pas lorsque le contenu du tampon spécifie un mode majeur, et certains modes majeurs "spéciaux" ne permettent pas au mode de changer. Vous pouvez annuler cette fonctionnalité de changement de mode en mettant la variable `change-major-mode-with-file-name` à `nil`.

20 Indentation

Ce chapitre décrit les commandes Emacs pour ajouter, supprimer, ou ajuster l'indentation.

⟨TAB⟩ Indente la ligne courante de manière "appropriée" selon le mode.

`C-j` Exécute ⟨RET⟩ suivi de ⟨TAB⟩ (`newline-and-indent`).

`M-^` Réunit deux lignes (`delete-indentation`). Ceci peut annuler l'effet de `C-j`.

`C-M-o` Coupe la ligne au point ; le texte de la ligne après le point devient une nouvelle ligne indentée sur la colonne où commence actuellement ce texte (`split-line`).

`M-m` Se déplace (en avant ou en arrière) sur le premier caractère non vierge de la ligne courante (`back-to-indentation`).

`C-M-\` Indente plusieurs lignes sur la même colonne (`indent-region`).

`C-x` ⟨TAB⟩ Déplace un bloc de lignes de façon rigide vers la droite ou vers la gauche (`indent-rigidly`).

`M-i` Indente depuis le point jusqu'à la prochaine colonne de tabulation prédéfinie. (`tab-to-tab-stop`).

`M-x indent-relative`
Indente depuis le point jusqu'à sous une indentation de la ligne précédente.

La plupart des langages de programmation ont une convention d'indentation. Pour le code Lisp, les lignes sont indentées selon leur profondeur dans les parenthèses. La même idée générale est utilisée pour le code C, bien qu'un grand nombre de détails soient différents.

Quel que soit le langage, pour indenter une ligne, utilisez la commande ⟨TAB⟩. Chaque mode majeur définit cette commande pour exécuter le type d'indentation appropriée au langage particulier. Dans le mode Lisp, ⟨TAB⟩ aligne la ligne selon sa profondeur dans les parenthèses. Aucune importance de l'endroit où vous vous trouvez dans la ligne lorsque vous tapez ⟨TAB⟩, la ligne est indentée en intégralité. Dans le mode C, ⟨TAB⟩ implémente un style d'indentation subtile et sophistiquée, qui tient compte d'un grand nombre d'aspects de la syntaxe C.

Dans le mode Texte, ⟨TAB⟩ lance la commande `tab-to-tab-stop`, qui indente la ligne sur la prochaine colonne de tabulation. Vous pouvez placer les tabulations avec `M-x edit-tab-stops`.

20.1 Commandes et Techniques d'Indentation

Pour vous placer sur l'indentation d'une ligne, faites `M-m` (`back-to-indentation`). Cette commande, n'importe où sur la ligne, positionne le point sur le premier caractère non vierge de la ligne.

Pour insérer une ligne indentée avant la ligne courante, faites `C-a C-o` ⟨TAB⟩. Pour une ligne indentée après la ligne courante, utilisez `C-e C-j`.

Si vous désirez juste insérer un caractère tab dans le tampon, vous pouvez taper `C-q` ⟨TAB⟩.

`C-M-o` (`split-line`) déplace le texte à partir du point jusqu'à la fin de la ligne verticalement vers le bas, de manière que la ligne courante devienne deux lignes. `C-M-o` déplace d'abord le point en avant à travers les espaces et tabulations. Il insére alors après le point un caractère newline et l'indentation nécessaire pour atteindre la colonne sur laquelle se trouve le point. Le point reste avant la ligne insérée ; de ce point de vue, `C-M-o` ressemble à `C-o`.

Pour joindre deux lignes correctement, utilisez la commande `M-^` (`delete-indentation`). Elle supprime l'indentation au début de la ligne courante, ainsi que les limites de la ligne, en les remplaçant par un espace unique. Comme cas particulier (utile pour le code Lisp), l'espace est omis si les carctères joints sont des parenthèses ouvrantes ou fermantes consécutives, ou si la jonction suit un autre caractère newline. Pour supprimer l'indentation d'une ligne, allez en début de ligne et utilisez `M-\` (`delete-horizontal-space`), qui supprime tous les espaces et tabulations autour du curseur.

Si vous avez un préfixe de remplissage, `M-^` supprime le préfixe de remplissage s'il apparaît après le caractère newline supprimé. See Section 21.5.3 [Fill Prefix], page 251.

Il existe aussi des commandes pour modifier l'indentation de plusieurs lignes d'un coup. `C-M-\` (`indent-region`) s'applique à toutes les lignes commençant dans la région ; elle indente chaque ligne de la manière "usuelle", comme si vous aviez tapé ⟨TAB⟩ au début de chaque ligne. Un argument numérique spécifie la colonne d'indentation, et chaque ligne est déplacée vers la droite ou vers la gauche pour que son premier caractère non vierge apparaisse sur cette colonne. `C-x` ⟨TAB⟩ (`indent-rigidly`) déplace toutes les lignes de la région vers la droite de la valeur de l'argument (vers la gauche pour un argument négatif). Le groupe entier de lignes est déplacé de façon rigide, d'où le nom de la commande.

`M-x indent-relative` indente au point d'après la ligne précédente (plus exactement, la dernière ligne non vierge). Elle insére des espaces au point, pour déplacer le point, jusqu'à ce qu'il se trouve sous un point d'indentation de la ligne précédente. Un point d'indentation est la fin d'une séquence de caractères vierges ou la fin de la ligne. Si le point se trouve au delà de tout point d'indentation de le ligne précédente, le caractère vierge avant le point

est supprimé et le premier point d'indentation alors applicable est utilisé. Si aucun point d'indentation n'est toujours applicable, `indent-relative` lance `tab-to-tab-stop` (see next section).

`indent-relative` est la définition de ⟨TAB⟩ de le mode Texte Indenté. See Chapter 21 [Text], page 243.

See Section 21.11.6 [Format Indentation], page 269, pour d'autres moyens de spécifier l'indentation pour des parties de votre texte.

20.2 Arrêts de Tabulation

Pour créer des tables, vous pouvez utiliser la définition de ⟨TAB⟩ pour le mode Texte, `tab-to-tab-stop`. Cette commande insére l'indentation avant le point, assez pour atteindre le prochain arrêt de tabulation. Si vous n'êtes pas en mode Texte, cette commande peut être trouvée sur la touche `M-i`.

Vous pouvez spécifier les arrêts de tabulation utilisés par `M-i`. Ile sont stockés dans une variable appelée `tab-stop-list`, comme une liste de numéros de colonne dans l'ordre croissant.

Le moyen convenable de placer les arrêts de tabulation est d'utiliser `M-x edit-tab-stops`, qui crée et sélectionne un tampon contenant une description des arrêts de tabulation. Vous pouvez éditer ce tampon pour spécifier des arrêts de tabulation différents, puis taper `C-c C-c` pour rendre actifs ces nouveaux arrêts de tabulation. `edit-tab-stops` enregistre quel tampon était courant lorsque vous l'avez invoqué, et stocke les arrêts de tabulation dans ce tampon ; normalement tous les tampons partagent les mêmes arrêts de tabulation et les changer dans un tampon affecte tous les tampons, mais si vous rendez `tab-stop-list` local à un tampon alors `edit-tab-stops` dans ce tampon éditera les définitions locales.

Voici à quoi ressemble le texte représentant des arrêts de tabulation ordinaires toutes les huit colonnes.

```
        :       :       :       :       :       :
0         1         2         3         4
0123456789012345678901234567890123456789012345678
To install changes, type C-c C-c
```

La premièr ligne contient un double-point à chaque arrêt de tabulation. Les lignes suivantes sont présentes seulement pour vous aider à voir où sont placés les double-points.

Notez que les arrêts de tabulation qui contrôlent `tab-to-tab-stop` n'ont rien à voir avec les caractères tab dans le tampon. See Section 11.12 [Display Custom], page 115, pour plus d'informations là-dessus.

20.3 Tabulations contre Espaces

Emacs utilise normalement aussi bien des caractères tab que des espaces pour indenter les lignes. Si vous préférez, toute indentation peut être faite avec des espaces uniquement. Pour demander cela, mettez `indent-tabs-mode` à `nil`. C'est une variable locale à chaque tampon ; altérer cette variable affecte seulement le tampon courant, mais il existe une valeur par défaut que vous pouvez aussi changer. See Section 31.2.4 [Locals], page 466.

Il existe aussi des commandes pour convertir des tabulations en espaces et vice versa, préservant toujours les colonnes de tout texte non vierge. `M-x tabify` recherche dans la région des séquences d'espaces, et convertit les séquences d'au moins trois espaces en tab si cela peut être fait sans modifier l'indentation. `M-x untabify` change tous les tabs de la région en un nombre approprié d'espaces.

21 Commandes pour Langages Humains

Le terme *texte* a deux significations bien différentes dans notre domaine de l'informatique. La première est une donnée sous la forme d'une séquence de caractères. Tout fichier que vous éditez avec Emacs est du texte, dans ce sens du terme. La seconde signification est plus restrictive : une séquence de caractères dans un langage humain que les humains peuvent lire (parfois après avoir été transformé par un formateur de texte), en opposition à un programme ou des commandes pour un programme.

Les langages humains ont des conventions syntaxiques/stylistiques qui peuvent être supportées ou utilisées avantageusement par des commandes d'éditeurs : les conventions mettant en jeu mots, phrases, paragraphes, et lettres capitales. Ce chapitre décrit les commandes Emacs pour toutes ces choses. Il existe aussi des commandes pour *remplir*, ce qui veut dire réarranger les lignes d'un paragraphe pour qu'elles aient approximativement la même longueur. Les commandes pour se déplacer à travers ou supprimer les mots, phrases ou paragraphes, bien que d'abord prévues pour éditer du texte, sont aussi souvent utilisées pour éditer des programmes.

Emacs a plusieurs modes majeurs pour éditer du texte de langage humain. Si le fichier contient du texte pur et simple, utilisez le mode Texte, qui personnalise Emacs pour les conventions syntaxiques de texte. Le mode Profil fournit des commandes spéciales pour travailler sur du texte avec une structure profilée. See Section 21.8 [Outline Mode], page 255.

Pour le texte contenant des commandes pour les formatteurs de texte, Emacs a d'autres modes majeurs, chacun pour un formateur de texte particulier. Ainsi, pour du texte à passer à TEX, vous pouvez utiliser le mode TEX (see Section 21.9 [TeX Mode], page 260). Pour du texte à passer à nroff, utilisez le mode Nroff.

Plutôt que d'utiliser un formateur de texte, vous pouvez éditer du texte formaté dans le style WYSIWYG ("what you see is what you get"), avec le mode Enrichi. De cette manière, le formatage apparaît à l'écran sous Emacs alors que vous éditez. See Section 21.11 [Formatted Text], page 265.

21.1 Mots

Emacs a des commandes pour vous déplacer ou travailler sur les mots. Par convention, les touches pour celles-ci sont toutes des Meta-caractères.

`M-f`	Avance d'un mot (`forward-word`).
`M-b`	Recule d'un mot (`backward-word`).
`M-d`	Efface la fin d'un mot (`kill-word`).
`M-⟨DEL⟩`	Efface le début d'un mot (`backward-kill-word`).

`M-@` Marque la fin du prochain mot (`mark-word`).

`M-t` Transpose deux mots ou déplace un mot à travers d'autres mots (`transpose-words`).

Notez comment ces touches forment une série parallèle à celles basées sur les caractères `C-f`, `C-b`, `C-d`, ⟨DEL⟩ et `C-t`. `M-@` est analogue à `C-@`, qui est un alias pour `C-`⟨SPC⟩.

Les commandes `M-f` (`forward-word`) et `M-b` (`backward-word`) déplacent en avant et en arrière à travers les mots. Ces Meta-caractères sont analogues aux caractères contrôle correspondants, `C-f` and `C-b`, qui déplacent à travers de simples caractères dans le texte. L'anlogie s'étend aux arguments numériques, qui servent de compteurs de répétition. `M-f` avec un argument négatif déplace en arrière, et `M-b` avec un argument négatif déplace en avant. Le déplacement en avant s'arrête juste après la dernière lettre du mot, alors que le déplacement en arrière s'arrête juste avant la première lettre.

`M-d` (`kill-word`) efface le mot après le point. Pour être précis, elle efface tout depuis le point jusqu'à l'endroit où `M-f` s'arrêterait. Ainsi, si le point est au milieu d'un mot, `M-d` efface seulement la partie après le point. Si des signes de ponctuation apparaissent entre le point et le mot suivant, ils sont effacés en même temps que le mot. (Si vous désirez seulement effacer le prochain mot mais non la ponctuation qui le précède, faites `M-f` pour vous rendre à la fin, puis effacez le mot en arrière avec `M-`⟨DEL⟩.) `M-d` accepte des arguments exactement comme `M-f`.

`M-`⟨DEL⟩ (`backward-kill-word`) efface le mot avant le point. Elle efface tout entre le point et l'endroit avant le point où `M-b` aurait déplacé le point. Si le point est après l'espace dans '`FOO, BAR`', alors '`FOO, `' est effacé. (Si vous désirez seulement effacer '`FOO`', et non la virgule et l'espace, utilisez `M-b M-d` plutôt que `M-`⟨DEL⟩.)

`M-t` (`transpose-words`) échange le mot avant ou contenant le point avec le mot suivant. Les caractères de délimitation entre les mots ne sont pas déplacés. Par exemple, '`FOO, BAR`' se transpose en '`BAR, FOO`' plutôt que '`BAR FOO,`'. See Section 13.2 [Transpose], page 137, pour plus d'informations sur la transposition et les arguments des commandes de transposition.

Pour opérer sur les *n* mots suivants avec une opération s'appliquant entre le point et la marque, vous pouvez soit poser la marque au point puis vous déplacer à travers les mots, ou vous pouvez utiliser la commande `M-@` (`mark-word`) qui ne déplace pas le point, mais place la marque où `M-f` vous aurait déplacé. `M-@` accepte un argument numérique indiquant le nombre de mots à passer pour poser la marque. Dans le mode de Marque Transitoire, cette commande active la marque.

La manière dont les commandes sur les mots comprennent la syntaxe est complètement contrôlée par la table de syntaxe. Tout caractère peut, par exemple, être déclaré comme étant un délimiteur de mots. See Section 31.6 [Syntax], page 485.

21.2 Phrases

Les commandes Emacs manipulant les phrases et les paragraphes sont pour la plupart sur des touches Meta, comme le sont les commandes sur les mots.

`M-a` Recule jusqu'au début de la phrase (`backward-sentence`).

`M-e` Avance jusqu'à la fin de la phrase (`forward-sentence`).

`M-k` Efface jusqu'à la fin de la phrase (`kill-sentence`).

`C-x` ⟨DEL⟩ Efface depuis le début de la phrase `backward-kill-sentence`).

Les commandes `M-a` et `M-e` (`backward-sentence` et `forward-sentence`) déplacent le point au début et à la fin de la phrase courante, respectivement. Elles ont été choisies pour ressembler à `C-a` et `C-e`, qui déplacent le point au début et à la fin de la ligne. Au contraire de celles-ci, `M-a` and `M-e` répétées ou avec un argument numérique déplacent le point à travers les phrases successives.

Reculer au début de la phrase place le point juste avant le premier caractère de la phrase ; avancer place le point juste après la ponctuation terminant la phrase. Aucune d'elles ne se déplace à travers l'espace séparant les deux phrases.

Exactement comme `C-a` et `C-e` ont une commande de coupe, `C-k`, pour aller avec, `M-a` et `M-e` ont une commande de coupe `M-k` (`kill-sentence`) qui coupe depuis le point jusqu'à la fin de la phrase. Avec moins un comme argument, elle coupe depuis le début de la phrase. Les arguments plus grands servent de compte de répétition. Il existe aussi une commande, `C-x` ⟨DEL⟩ (`backward-kill-sentence`), pour couper depuis le début de la phrase. Cette commande est utile lorsque vous changez d'avis en pleine composition de texte.

Les commandes sur les phrases assument que vous suivez la convention américaine de placer deux espaces à la fin d'une phrase ; elles considèrent qu'une phrase se finit après un '`.`', '`?`' ou '`!`' suivi d'une fin de ligne ou de deux espaces, avec un nombre quelconque de caractères '`)`', '`]`', '`'`', ou '`"`' permis entre eux. Une phrase commence ou finnit aussi lorsq'un paragraphe commence ou finit.

La variable `sentence-end` contrôle la reconnaissance de la fin d'une phrase. C'est une expression rationnelle qui correspond aux derniers caractères d'une phrase, y compris les espaces suivant la phrase. Sa valeur par défaut est

```
"[.?!][]\"')]*\\($\\|\t\\|  \\)[ \t\n]*"
```

Cet exemple est expliqué dans la section sur les expressions rationnelles. See Section 12.5 [Regexps], page 125.

Si vous désirez utiliser seulement un espace entre les phrases, vous devez mettre `sentence-end` à cette valeur :

```
"[.?!][]\"')]*\\($\\|\t\\| \\)[ \t\n]*"
```

Vous devez aussi mettre la variable `sentence-end-double-space` à `nil` pour que les commandes de remplissage n'attendent et ne laissent qu'un seul espace à la fin d'une phrase. Notez que ceci rend impossible de distinguer les points terminant les phrases de ceux indiquant les abbréviations.

21.3 Paragraphes

Les commandes Emacs pour manipuler des paragraphes sont aussi des touches Meta.

`M-{` Recule jusqu'au début du paragraphe (`backward-paragraph`).

`M-}` Avance jusqu'à la fin du paragraphe (`forward-paragraph`).

`M-h` Place le point et la marque autour du paragraphe courant ou suivant (`mark-paragraph`).

`M-{` déplace le point au début du paragraphe courant ou précédent, alors que `M-}` déplace le point à la fin du paragraphe courant ou suivant. Les lignes vierges et les lignes de commande de formatage de texte séparent les paragraphes et ne sont pas considérées comme faisant partie d'un paragraphe. Dans le mode Fondamental, mais non dans le mode Texte, une ligne indentée commence aussi un nouveau paragraphe. (Si un paragraphe est précédé d'une ligne vierge, ces commandes traitent cette ligne vierge comme étant le début du paragraphe.)

Dans les modes majeurs pour les programmes, les paragraphes commencent et finissent uniquement avec une ligne vierge. Ceci rend les commandes de paragraphe toujours utilisables, même s'il n'y a plus de réels paragraphes.

Lorsqu'un préfixe de remplissage est défini, alors les paragraphes sont délimités par toute ligne ne commençant pas par le préfixe de remplissage. See Section 21.5 [Filling], page 248.

Lorsque vous désirez opérer sur un paragraphe, vous pouvez utiliser la commande `M-h` (`mark-paragraph`) pour placer la région autour de celui-ci. Ainsi, par exemple, `M-h C-w` coupe le paragraphe autour ou après le point. La commande `M-h` place le point au début et la marque à la fin du paragraphe contenant le point. Dans le mode de Marque Transitoire, elle active la marque. Si le point se trouve entre deux paragraphes (dans une suite de lignes vierges ou à la frontière), le paragraphe suivant le point est entouré par le point et la marque. Si des lignes vierges précèdent la première ligne du paragraphe, une de ces lignes vierges est incluse dans la région.

La définition précise d'une frontière de paragraphe est contrôlée par les variables `paragraph-separate` et `paragraph-start`. La valeur de `paragraph-start` est une expression rationnelle qui doit correspondre à toute ligne commençant ou séparant des paragraphes. La valeur de

`paragraph-separate` est une autre expression rationnelle qui doit correspondre seulement aux lignes qui séparent les paragraphes sans être contenu dans aucun paragraphe (par exemple, les lignes vierges). Les lignes començant un nouveau paragraphe et contenu dans celui-ci doivent correspondre seulement à `paragraph-start`, et non à `paragraph-separate`. Par exemple, dans le mode Fondamental, `paragraph-start` est `" [ \t\n\f]"` et `paragraph-separate` est `" [ \t\f]*$"`.

Normalement, il est désirable que les frontières de pages séparent les paragraphes. Les valeurs par défaut pour ces variables reconnaissent le séparateur de pages habituel.

21.4 Pages

Les fichiers sont souvent plus plaisants lorsqu'ils sont divisés en plusieurs *pages* par le caractère *formfeed* (ASCII control-L, code octal 014). Lorsque vous imprimez un fichier, ce caractère force un saut de page ; ainsi, chaque page du fichier se trouve sur une page différente sur le papier. La plupart des commandes Emacs traitent le caractère de séparation de page comme tout autre caractère : vous pouvez l'insérer avec `C-q C-l`, et le supprimer avec ⟨DEL⟩. Ainsi, vous êtes de libre de paginer ou non votre fichier. Cependant, les pages étant un moyen pratique de diviser un fichier, Emacs fournit des commandes pour vous déplacer à travers elles ou opérer sur elles.

`C-x [` Déplace le point sur la frontière de page précédente (`backward-page`).

`C-x ]` Déplace le point sur la frontière de page suivante (`forward-page`).

`C-x C-p` Place le point et la marque autour de cette page (ou d'une autre page) (`mark-page`).

`C-x l` Compte les lignes de cette page (`count-lines-page`).

La commande `C-x [` (`backward-page`) déplace le point immédiatement après le délimiteur de page précédent. Si le point est déjà placé juste après un délimiteur de page, il saute celui-ci et s'arrête au précédent. Un argument numérique sert de compte de répétition. La commande `C-x ]` (`forward-page`) déplace le point juste après le délimiteur de page suivant.

La commande `C-x C-p` (`mark-page`) place le point au début de la page courante et la marque à la fin. Le délimiteur de page de fin est inclus (la marque est placée juste après). Le délimiteur de page de début est exclu (le point est placé juste après). `C-x C-p C-w` est un moyen rapide de couper une page pour la coller ailleurs. Si vous vous déplacez vers un autre délimiteur de page avec `C-x [` et `C-x ]`, puis collez la page coupée, toutes les pages seront de nouveau bien délimitées. La raison pour laquelle `C-x C-p` inclut seulement le délimiteur de page final dans la région est de s'assurer de cela.

Un argument numérique à `C-x C-p` est utilisé pour spécifier quelle page sélectionner, par rapport à la page courante. Zéro indique la page courante. Un indique la page suivante, et −1 indique la page précédente.

La commande `C-x l` (`count-lines-page`) est pratique pour décider à quel endroit couper une page en deux. Elle indique dans la zone de répercussion le nombre total de lignes de la page courante, puis indique le nombre de celles précédant la ligne courante et celles la suivant, comme dans

```
Page has 96 (72+25) lines
```

Notez que la somme est supérieure de un ; ceci est correct si le point n'est pas au début de la ligne.

La variable `page-delimiter` contrôle où les pages commencent. Sa valeur est une expression rationnelle qui correspond à un début de ligne séparant deux pages. La valeur normale de cette variable est `"^\f"`, qui correspond à un caractère formfeed en début de ligne.

21.5 Remplissage de Texte

Remplir du texte veut dire le découper en lignes d'une certaine longueur. Emacs remplit le texte de deux manières différentes. Dans le mode Remplissage Automatique, l'insertion de texte avec des caractères auto-insérables remplit aussi le texte automatiquement. Il existe aussi des commandes de remplissage explicites que vous pouvez utiliser lorsque l'édition du texte ne le remplit pas automatiquement. Lorsque vous éditez du texte formaté, vous pouvez spécifier un style de remplissage pour chaque portion du texte (see Section 21.11 [Formatted Text], page 265).

21.5.1 Mode Remplissage Automatique

Le mode *Remplissage Automatique* est un mode mineur dans lequel les lignes sont automatiquement coupées lorsqu'elles deviennent trop longues. La coupure survient seulement lorsque vous tapez ⟨SPC⟩ ou ⟨RET⟩.

`M-x auto-fill-mode`
: Permet ou interdit le mode Remplissage Automatique.

⟨SPC⟩
⟨RET⟩
: Dans le mode Remplissage Automatique, coupe la ligne si nécessaire.

`M-x auto-fill-mode` démarre le mode Remplissage Automatique s'il était arrêté, ou l'arrête s'il était démarré. Avec un argument numérique positif il démarre toujours le mode Remplissage Automatique, et avec un

argument négatif l'arrête toujours. Vous pouvez voir si le mode Remplissage Automatique est effectif par la présence du mot '`Fill`' dans la ligne de mode, entre parenthèses. Le mode Remplissage Automatique est un mode mineur qui est activé ou désactivé individuellement pour chaque tampon. See Section 31.1 [Minor Modes], page 455.

Dans le mode Remplissage Automatique, les lignes sont automatiquement coupées à un espace lorsqu'elles deviennent plus longues que la longueur spécifiée. La coupure de la ligne et le réarrangement s'effectuent seulement lorsque vous tapez ⟨SPC⟩ ou ⟨RET⟩. si vous désirez insérer un espace ou un caractère newline sans permettre la coupure de la ligne, tapez `C-q` ⟨SPC⟩ ou `C-q C-j` (souvenez-vous qu'un caractère newline est en vérité control-J). De plus, `C-o` iinsére un caractère newline sans couper la ligne.

Le remplissage automatique fonctionne correctement avec les modes de langages de programmation, car il indente les nouvelles lignes avec ⟨TAB⟩. Si une ligne de commentaire devient trop longue, le texte du commentaire est coupé en deux lignes de commentaires. Optionnellement, de nouveaux délimiteurs de commentaires sont ajoutés à la fin de la première ligne et au début de la seconde de manière que chacune des lignes soit un commentaire séparé ; la variable `comment-multi-line` contrôle le choix (see Section 22.7 [Comments], page 291).

Le remplissage adaptatif (voir la section suivante) fonctionne aussi bien pour le mode Remplissage Automatique que pour les commandes explicites de remplissage. Il prend automatiquement un préfixe de remplissage sur la seconde ou première ligne du paragraphe.

Le mode Remplissage Automatique ne remplit pas des paragraphes entiers ; il peut couper des lignes mais ne peut pas les fusionner. Ainsi l'édition au milieu d'un paragraphe peut rendre incorrect le remplissage du paragraphe. La manière la plus facile de rendre le paragraphe correctement rempli est d'utiliser les commandes explicites de remplissage.

Un grand nombre d'utilisateurs apprécient le mode Remplissage Automatique et veulent l'utiliser pour tous les fichiers texte. La section sur les fichiers d'initialisation indique comment l'utiliser en permanence. See Section 31.7 [Init File], page 486.

21.5.2 Commandes Explicites de Remplissage

`M-q` Remplit le paragraphe courant (`fill-paragraph`).

`C-x f` Place la colonne de remplissage (`set-fill-column`).

`M-x fill-region`
Remplit chacun des paragraphes de la région (`fill-region`).

`M-x fill-region-as-paragraph.`
Remplit la région, la considérant comme un unique paragraphe.

`M-s` Centre une ligne.

Pour remplir un paragraphe, utilisez la commande `M-q` (`fill-paragraph`). Cette commande opère sur le paragraphe contenant le point, ou celui suivant le point si ce dernier se trouve entre deux paragraphes. Le remplissage est effectué en supprimant toutes les coupures de ligne puis en en plaçant de nouveaux aux endroits nécessaires.

Pour remplir plusieurs paragraphes, utilisez `M-x fill-region`, qui divise la région en paragraphes et remplit chacun d'eux.

`M-q` et `fill-region` utilisent le même critère que `M-h` pour trouver les limites de paragraphes (see Section 21.3 [Paragraphs], page 246). Pour plus de contrôle, vous pouvez utiliser `M-x fill-region-as-paragraph`, qui remplit tout entre le point et la marque. Cette commande supprime toutes les lignes vierges de la région, de sorte que les blocs séparés de texte terminent combinés en un seul bloc.

Un argument numérique à `M-q` permet de *justifier* le texte en plus de le remplir. Cela veut dire que des espaces supplémentaires sont insérés pour placer la ligne de marge droite exactement à la colonne de remplissage. Pour supprimer les espaces supplémentaires, utilisez `M-q` sans argument. (De même pour `fill-region`.) Un autre moyen de contrôler la justification et choisir d'autres styles de remplissage est d'utiliser la propriété `justification` de texte ; voir Section 21.11.7 [Format Justification], page 270.

La commande `M-s` (`center-line`) centre la ligne courante par rapport à la colonne de remplissage courante. Avec un argument *n*, elle centre *n* lignes individuellement et place le point après celles-ci.

La longueur maximale d'une ligne pour le remplissage est définie dans la variable `fill-column`. La modification de la valeur de `fill-column` rend cette variable locale au tampon courant ; avant cela, la valeur par défaut est effective. La valeur par défaut est initialement 70. See Section 31.2.4 [Locals], page 466. Le plus facile pour définir `fill-column` est d'utiliser la commande `C-x f` (`set-fill-column`). Avec un argument numérique, elle utilise cet argument comme nouvelle colonne de remplissage. Avec seulement `C-u` comme argument, elle place `fill-column` à la position horizontale courante du point.

Les commandes Emacs considèrent normalement un point suivi de deux espaces ou d'un caractère newline comme la fin d'une phrase ; un point suivi seulement d'un espace indique une abbréviation et non la fin d'une phrase. Pour préserver la distinction entre les deux manières d'utiliser un point, les commandes de remplissage ne coupent pas les lignes après un point suivi seulement d'un espace.

Si la variable `sentence-end-double-space` est `nil`, les commandes de remplissage attendent et laissent seulement un espace à la fin des phrases. D'ordinaire cette variable est `t`, et les commandes de remplissage insistent

sur les deux espaces à la fin des phrases, comme expliqué ci-dessus. See Section 21.2 [Sentences], page 245.

Si la variable `colon-double-space` est non `nil`, les commandes de remplissage insérent deux espaces après un caractère deux-points.

21.5.3 Le Préfixe de Remplissage

Pour remplir un paragraphe dans lequel chaque ligne commence avec une marque spéciale (qui peut être quelques espaces, donnant un paragraphe indenté), vous pouvez utiliser la caractéristique du *préfixe de remplissage*. Le préfixe de remplissage est une chaîne qu'Emacs attend à chaque début de ligne, et qui n'est pas inclue dans le remplissage. Vous pouvez spécifier explicitement un préfixe de remplissage ; Emacs peut aussi déduire automatiquement le préfixe de remplissage (see Section 21.5.4 [Adaptive Fill], page 252).

`C-x .` Définit le préfixe de remplissage (`set-fill-prefix`).

`M-q` Remplit un paragraphe en utilisant le préfixe de remplissage courant (`fill-paragraph`).

`M-x fill-individual-paragraphs`
Remplit la région, en considérant chaque changement d'indentation comme le début d'un nouveau paragraphe.

`M-x fill-nonuniform-paragraphs`
Remplit la région, considérant seulement les séparateurs de paragraphes comme le début d'un nouveau paragraphe.

Pour spécifier un préfixe de remplissage, allez sur une ligne commençant par le préfixe désiré, placez le point à la fin du préfixe, et appelez la commande `C-x .` (`set-fill-prefix`). C'est un point après `C-x`. Pour annuler le préfixe de remplissage, spécifiez un préfixe vide : tapez `C-x .` avec le point en début de ligne.

Lorsque le préfixe de remplissage est effectif, les commandes de remplissage suppriment le préfixe de remplissage de chaque ligne avant de les remplir et l'insére au début de chaque ligne après le remlpissage. Le mode de remplissage automatique insére aussi automatiquement le préfixe de remplissage lorsqu'il crée une nouvelle ligne. La commande `C-o` insére le préfixe de remplissage aux nouvelles lignes qu'elle crée, lorsque vous l'utilisez en début de ligne. (see Section 4.7 [Blank Lines], page 48). De même, la commande `M-^` supprime le préfixe (s'il est présent) après le caractère newline qu'elle supprime (see Chapter 20 [Indentation], page 239).

Par exemple, si `fill-column` est 40 et que vous définissez le préfixe de remplissage à '`;; `', alors `M-q` dans le texte suivant

```
;; Voici un
```

```
;; exemple de paragraphe
;; dans un commentaire de type Lisp.
```

produit ceci :

```
;; Voici un exemple de paragraphe dans
;; un commentaire de type Lisp.
```

Les lignes ne commençant pas avec le préfixe de remplissage sont considérées commencer un paragraphe, aussi bien pour `M-q` que pour les commandes sur les paragraphes ; ceci donne de bons résultats pour des paragraphes indentés (chaque ligne est indentée à part la première). Les lignes qui deviennent vides ou indentées une fois que le préfixe est supprimé séparent ou commencent aussi des paragraphes ; c'est ce qui est utilisé lorsque vous écrivez des commentaires de plusieurs paragraphes avec des délimiteurs de commentaire sur chaque ligne.

Vous pouvez utiliser `M-x fill-individual-paragraphs` pour définir le préfixe de remplissage pour chaque paragraphe automatiquement. Cette commande divise la région en paragraphes, traitant chaque changement de niveau d'indentation en début de ligne comme le début d'un nouveau paragraphe, et remplit chacun de ces paragraphes. Ainsi, chaque ligne dans un "paragraphe" a le même niveau d'indentation. Cette indentation est utilisé comme préfixe de remplissage pour ce paragraphe.

`M-x fill-nonuniform-paragraphs` est une commande similaire qui divise la région en paragraphes d'une manière différente. Elle considère seulement les lignes de séparation de paragraphe (définies par `paragraph-separate`) comme commençant un nouveau paragraphe. Les lignes d'un paragraphe pouvant de ce fait avoir différentes indentations, le préfixe de remplissage utilisé est le plus petit niveau d'indentation des lignes du paragraphe. Ceci donne de bons résultats pour des styles qui indentent la première ligne d'un paragraphe plus ou moins que le reste du paragraphe.

Le préfixe de remplissage est stocké dansla variable `fill-prefix`. Sa valeur est une chaîne, ou `nil` lorsqu'il n'y a pas de préfixe de remplissage. C'est une variable locale aux tampons ; la modification de la variable affecte seulement le tampon courant, mais il existe une valeu par défaut que vous pouvez aussi modifier. See Section 31.2.4 [Locals], page 466.

La propriété d'`indentation` de texte fournit un autre moyen de contrôler le niveau d'indentation reçu par les paragraphes. See Section 21.11.6 [Format Indentation], page 269.

21.5.4 Remplissage Adaptatif

Les commandes de remplissage peuvent déduire automatiquement le préfixe de remplissage correct pour un paragraphe dans certains cas : les espaces et certains caractères de ponctuation en début de ligne sont propagés sur toutes les lignes du paragraphe.

Si le paragraphe a deux lignes ou plus, le préfixe de remplissage est obtenu de la seconde ligne du paragraphe, mais seulement s'il apparaît aussi sur la première ligne.

Si un paragraphe a seulement une ligne, les commandes de remplissage *peuvent* obtenir un préfixe de cette ligne. La décision est compliquée car il y a trois choses raisonnables à faire dans un tel cas :

- Utiliser le préfixe de la première ligne pour toutes les lignes du paragraphe.
- Indenter les lignes suivantes avec des espaces, pour qu'elles s'alignent avec le texte suivant le préfixe sur la première ligne, mais ne recopie pas le préfixe de la première ligne.
- Ne rien faire de spécial avec la seconde ligne et les suivantes.

Ces trois styles de formatage sont couramment utilisés. Les commandes de remplissage essaient alors de déterminer ce que vous préférez, selon le préfixe qui apparaît et sur le mode majeur. Voici comment elles procèdent.

Si le préfixe trouvé sur la première ligne correspond à `adaptive-fill-first-line-regexp`, ou s'il semble être une séquence de début de commentaire (qui dépend du mode majeur), alors le préfixe trouvé est utilisé pour remplir le paragraphe, dans la mesure où il n'agit pas comme un début de paragraphe pour les lignes suivantes.

Autrement, le préfixe trouvé est converti en un nombre équivalent d'espaces, et ces espaces sont utilisés comme préfixe de remplissage pour les lignes suivantes, dans la mesure où il n'agit pas comme un début de paragraphe sur les lignes suivantes.

En mode Texte, et dans les autres modes où seules les lignes vides et les délimiteurs de page séparent les paragraphes, le préfixe choisi par le remplissage adaptatif n'agit jamais comme un début de paragraphe, et peut toujours être utilisé pour le remplissage.

La variable `adaptive-fill-regexp` détermine quels types de débuts de lignes peuvent servir de préfixe de remplissage : tous les caractères en début de ligne correspondant à cette expression rationnelle sont utilisés. Si vous définissez la variable `adaptive-fill-mode` à `nil`, le préfixe de remplissage n'est jamais choisi automatiquement.

Vous pouvez spécifier des moyens plus complexes de choisir un préfixe de remplissage automatiquement en définissant la variable `adaptive-fill-function` comme une fonction. Cette fonction est appelée avec le point après la marge gauche d'une ligne, et doit retourner le préfixe de remplissage approprié selon cette ligne. Si elle retourne `nil`, cela veut dire qu'elle ne voit aucun préfixe de remplissage dans cette ligne.

21.6 Commandes de Conversion de Casse

Emacs a des commandes pour convertir soit un seul mot, soit une quantité arbitraire de texte en majuscule ou en minuscule.

`M-l` Convertit le mot suivant en minuscule (`downcase-word`).

`M-u` Convertit le mot suivant en majuscule (`upcase-word`).

`M-c` Capitalise le mot suivant (`capitalize-word`).

`C-x C-l` Convertit la région en minuscule (`downcase-region`).

`C-x C-u` Convertit la région en majuscule (`upcase-region`).

Les commandes de conversion de mots sont les plus utiles. `M-l` (`downcase-word`) convertit le mot après le point en minuscule, puis place le point après celui-ci. Ainsi, la répétition de `M-l` convertit successivement plusieurs mots. `M-u` (`upcase-word`) convertit le mot entier en majuscules, alors que `M-c` (`capitalize-word`) convertit la première lettre du mot en majuscule et le reste du mot en minuscule. Toutes ces commandes convertissent plusieurs mots à la fois s'il leur est fourni un argument. Elles sont tout particulièrement pratiques pour convertir une grande quantité de texte tout en majuscule en casse mixée, car vous pouvez vous déplacer à travers le texte en utilisant `M-l`, `M-u` ou `M-c` sur chaque mot tel qu'approprié, en utilisant occasionnellement `M-f` pour sauter un mot.

Lorsqu'un argument négatif leur est fourni, les commandes de conversion de casse de mots s'appliquent au nombre approprié de mots avant le point, mais ne déplace pas le point. Ceci est pratique lorsque vous venez juste de taper un mot avec une mauvaise casse : vous pouvez lancer la commande de conversion de casse et continuer à taper.

Si une commande de conversion de casse de mots est lancée avec le point au milieu d'un mot, elle s'applique seulement à la partie du mot suivant le point. Exactement de la même manière que `M-d` (`kill-word`). Avec un argument négatif, la conversion de casse s'applique uniquement à la partie du mot avant le point.

Les autres commandes de conversion de casse sont `C-x C-u` (`upcase-region`) et `C-x C-l` (`downcase-region`), qui convertissent tout entre le point et la marque dans la casse spécifiée. Le point et la marque ne sont pas déplacés.

Les commandes de conversion de casse de régions `upcase-region` et `downcase-region` sont normalement interdites. Cela veut dire qu'elles demandent confirmation si vous essayez de les utiliser. Lorsque vous confirmez, vous pouvez permettre la commande, ce qui veut dire qu'elle ne demandera plus ed confirmation. See Section 31.4.11 [Disabling], page 484.

21.7 Mode Texte

Lorsque vous éditez des fichiers de texte dans un langage humain, il est plus commode d'utiliser le mode Texte plutôt que le mode Fondamental. Pour entrer en mode Texte, tapez `M-x text-mode`.

En mode Texte, seules les lignes vierges et les délimiteurs de page séparent les paragraphes. Résultat, les paragraphes peuvent être indentés, et le remplissage adaptatif détermine quelle indentation utiliser pour remplir un paragraphe. See Section 21.5.4 [Adaptive Fill], page 252.

Le mode Texte définit ⟨TAB⟩ pour exécuter `indent-relative` (see Chapter 20 [Indentation], page 239), pour que vous puissiez indenter une ligne comme la ligne précédente. Lorsque la ligne précédente n'est pas indentée, `indent-relative` exécute `tab-to-tab-stop`, qui utilise les arrêts de tabulation d'Emacs (see Section 20.2 [Tab Stops], page 241).

Le mode Texte n'utilise pas les fonctionnalités ayant rapport avec les commentaires, exepté lorsque vous les invoquez explicitement. Il change la table de syntaxe pour que les points ne soient pas considérés comme faisant partie d'un mot, alors que les apostrophes, caractères backspace et souligné sont considérés comme faisant partie des mots.

Si vous indentez les premières lignes des paragraphes, alors vous devriez utiliser le mode Texte Paragraph-Indent plutôt que le mode Texte. Dans ce mode, vous n'avez pas besoin de séparer les paragraphes avec des lignes vierges, car l'indentation de la première ligne est suffisante pour démarrer un paragraphe ; cependant les paragraphes dans lesquels chaque ligne est indentée ne sont pas supportés. Use `M-x paragraph-indent-text-mode` pour entrer dans ce mode.

Le mode Texte, et tous les modes basés sur celui-ci, définissent `M-`⟨TAB⟩ comme la commande `ispell-complete-word`, qui réalise la complétion de la partie du mot dans le tampon avant le point, en utilisant le dictionnaire de mots comme espace de mots possibles. See Section 13.4 [Spelling], page 139.

L'entrée dans le mode Texte exécute le crochet `text-mode-hook`. Les autres modes majeurs relatifs au mode Texte exécutent aussi ce crochet, puis leurs propres crochets ; ceux-ci incluent le mode Texte Paragraph-Indent, le mode Nroff, le mode TEX, le mode Canevas, et le mode Mail. Les fonctions crochet de `text-mode-hook` peuvent utiliser la valeur de `major-mode` pour déterminer lequel de ces modes est actuellement entré. See Section 31.2.3 [Hooks], page 465.

21.8 Outline Mode

Outline mode is a major mode much like Text mode but intended for editing outlines. It allows you to make parts of the text temporarily invisible so that you can see the outline structure. Type `M-x outline-mode` to switch to Outline mode as the major mode of the current buffer.

When Outline mode makes a line invisible, the line does not appear on the screen. The screen appears exactly as if the invisible line were deleted, except that an ellipsis (three periods in a row) appears at the end of the previous visible line (only one ellipsis no matter how many invisible lines follow).

Editing commands that operate on lines, such as `C-n` and `C-p`, treat the text of the invisible line as part of the previous visible line. Killing an entire visible line, including its terminating newline, really kills all the following invisible lines along with it.

Outline minor mode provides the same commands as the major mode, Outline mode, but you can use it in conjunction with other major modes. Type `M-x outline-minor-mode` to enable the Outline minor mode in the current buffer. You can also specify this in the text of a file, with a file local variable of the form '`mode: outline-minor`' (see Section 31.2.5 [File Variables], page 468).

The major mode, Outline mode, provides special key bindings on the `C-c` prefix. Outline minor mode provides similar bindings with `C-c @` as the prefix; this is to reduce the conflicts with the major mode's special commands. (The variable `outline-minor-mode-prefix` controls the prefix used.)

Entering Outline mode runs the hook `text-mode-hook` followed by the hook `outline-mode-hook` (see Section 31.2.3 [Hooks], page 465).

21.8.1 Format of Outlines

Outline mode assumes that the lines in the buffer are of two types: *heading lines* and *body lines*. A heading line represents a topic in the outline. Heading lines start with one or more stars; the number of stars determines the depth of the heading in the outline structure. Thus, a heading line with one star is a major topic; all the heading lines with two stars between it and the next one-star heading are its subtopics; and so on. Any line that is not a heading line is a body line. Body lines belong with the preceding heading line. Here is an example:

```
* 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.
```

A heading line together with all following body lines is called collectively an *entry*. A heading line together with all following deeper heading lines and their body lines is called a *subtree*.

You can customize the criterion for distinguishing heading lines by setting the variable `outline-regexp`. Any line whose beginning has a match for this regexp is considered a heading line. Matches that start within a line (not at the left margin) do not count. The length of the matching text determines the level of the heading; longer matches make a more deeply nested level. Thus, for example, if a text formatter has commands '`@chapter`', '`@section`' and '`@subsection`' to divide the document into chapters and sections, you could make those lines count as heading lines by setting `outline-regexp` to '`"@chap\\|@\\(sub\\)*section"`'. Note the trick: the two words '`chapter`' and '`section`' are equally long, but by defining the regexp to match only '`chap`' we ensure that the length of the text matched on a chapter heading is shorter, so that Outline mode will know that sections are contained in chapters. This works as long as no other command starts with '`@chap`'.

It is possible to change the rule for calculating the level of a heading line by setting the variable `outline-level`. The value of `outline-level` should be a function that takes no arguments and returns the level of the current heading. Some major modes such as C, Nroff, and Emacs Lisp mode set this variable in order to work with Outline minor mode.

21.8.2 Outline Motion Commands

Outline mode provides special motion commands that move backward and forward to heading lines.

`C-c C-n` Move point to the next visible heading line (`outline-next-visible-heading`).

`C-c C-p` Move point to the previous visible heading line (`outline-previous-visible-heading`).

`C-c C-f` Move point to the next visible heading line at the same level as the one point is on (`outline-forward-same-level`).

`C-c C-b` Move point to the previous visible heading line at the same level (`outline-backward-same-level`).

`C-c C-u` Move point up to a lower-level (more inclusive) visible heading line (`outline-up-heading`).

`C-c C-n` (`outline-next-visible-heading`) moves down to the next heading line. `C-c C-p` (`outline-previous-visible-heading`) moves similarly backward. Both accept numeric arguments as repeat counts. The names emphasize that invisible headings are skipped, but this is not really a special feature. All editing commands that look for lines ignore the invisible lines automatically.

More powerful motion commands understand the level structure of headings. `C-c C-f` (`outline-forward-same-level`) and `C-c C-b` (`outline-backward-same-level`) move from one heading line to another visible heading at the same depth in the outline. `C-c C-u` (`outline-up-heading`) moves backward to another heading that is less deeply nested.

21.8.3 Outline Visibility Commands

The other special commands of outline mode are used to make lines visible or invisible. Their names all start with `hide` or `show`. Most of them fall into pairs of opposites. They are not undoable; instead, you can undo right past them. Making lines visible or invisible is simply not recorded by the undo mechanism.

`C-c C-t` Make all body lines in the buffer invisible (`hide-body`).

`C-c C-a` Make all lines in the buffer visible (`show-all`).

`C-c C-d` Make everything under this heading invisible, not including this heading itself3 (`hide-subtree`).

`C-c C-s` Make everything under this heading visible, including body, subheadings, and their bodies (`show-subtree`).

`C-c C-l` Make the body of this heading line, and of all its subheadings, invisible (`hide-leaves`).

`C-c C-k` Make all subheadings of this heading line, at all levels, visible (`show-branches`).

`C-c C-i` Make immediate subheadings (one level down) of this heading line visible (`show-children`).

`C-c C-c` Make this heading line's body invisible (`hide-entry`).

`C-c C-e` Make this heading line's body visible (`show-entry`).

`C-c C-q` Hide everything except the top *n* levels of heading lines (`hide-sublevels`).

`C-c C-o` Hide everything except for the heading or body that point is in, plus the headings leading up from there to the top level of the outline (`hide-other`).

Two commands that are exact opposites are `C-c C-c` (`hide-entry`) and `C-c C-e` (`show-entry`). They are used with point on a heading line, and apply only to the body lines of that heading. Subheadings and their bodies are not affected.

Two more powerful opposites are `C-c C-d` (`hide-subtree`) and `C-c C-s` (`show-subtree`). Both expect to be used when point is on a heading line, and both apply to all the lines of that heading's *subtree*: its body, all its subheadings, both direct and indirect, and all of their bodies. In other words, the subtree contains everything following this heading line, up to and not including the next heading of the same or higher rank.

Intermediate between a visible subtree and an invisible one is having all the subheadings visible but none of the body. There are two commands for doing this, depending on whether you want to hide the bodies or make the subheadings visible. They are `C-c C-l` (`hide-leaves`) and `C-c C-k` (`show-branches`).

A little weaker than `show-branches` is `C-c C-i` (`show-children`). It makes just the direct subheadings visible—those one level down. Deeper subheadings remain invisible, if they were invisible.

Two commands have a blanket effect on the whole file. `C-c C-t` (`hide-body`) makes all body lines invisible, so that you see just the outline structure. `C-c C-a` (`show-all`) makes all lines visible. These commands can be thought of as a pair of opposites even though `C-c C-a` applies to more than just body lines.

The command `C-c C-q` (`hide-sublevels`) hides all but the top level headings. With a numeric argument *n*, it hides everything except the top *n* levels of heading lines.

The command `C-c C-o` (`hide-other`) hides everything except the heading or body text that point is in, plus its parents (the headers leading up from there to top level in the outline).

You can turn off the use of ellipses at the ends of visible lines by setting `selective-display-ellipses` to `nil`. Then there is no visible indication of the presence of invisible lines.

21.8.4 Viewing One Outline in Multiple Views

You can display two views of a single outline at the same time, in different windows. To do this, you must create an indirect buffer using `M-x make-indirect-buffer`. The first argument of this command is the existing outline buffer name, and its second argument is the name to use for the new indirect buffer. See Section 15.6 [Indirect Buffers], page 191.

Once the indirect buffer exists, you can display it in a window in the normal fashion, with `C-x 4 b` or other Emacs commands. The Outline mode commands to show and hide parts of the text operate on each buffer inde-

pendently; as a result, each buffer can have its own view. If you want more than two views on the same outline, create additional indirect buffers.

21.9 TEX Mode

TEX is a powerful text formatter written by Donald Knuth; it is also free, like GNU Emacs. LaTEX is a simplified input format for TEX, implemented by TEX macros; it comes with TEX. SliTEX is a special form of LaTEX.

Emacs has a special TEX mode for editing TEX input files. It provides facilities for checking the balance of delimiters and for invoking TEX on all or part of the file.

TEX mode has three variants, Plain TEX mode, LaTEX mode, and SliTEX mode (these three distinct major modes differ only slightly). They are designed for editing the three different formats. The command `M-x tex-mode` looks at the contents of the buffer to determine whether the contents appear to be either LaTEX input or SliTEX input; if so, it selects the appropriate mode. If the file contents do not appear to be LaTEX or SliTEX, it selects Plain TEX mode. If the contents are insufficient to determine this, the variable `tex-default-mode` controls which mode is used.

When `M-x tex-mode` does not guess right, you can use the commands `M-x plain-tex-mode`, `M-x latex-mode`, and `M-x slitex-mode` to select explicitly the particular variants of TEX mode.

21.9.1 TEX Editing Commands

Here are the special commands provided in TEX mode for editing the text of the file.

`"` Insert, according to context, either ‘‘‘’ or ‘"’ or ‘’’’ (`tex-insert-quote`).

`C-j` Insert a paragraph break (two newlines) and check the previous paragraph for unbalanced braces or dollar signs (`tex-terminate-paragraph`).

`M-x validate-tex-region`
Check each paragraph in the region for unbalanced braces or dollar signs.

`C-c {` Insert ‘`{}`’ and position point between them (`tex-insert-braces`).

`C-c }` Move forward past the next unmatched close brace (`up-list`).

In TEX, the character ‘"’ is not normally used; we use ‘‘‘’ to start a quotation and ‘’’’ to end one. To make editing easier under this formatting

convention, TEX mode overrides the normal meaning of the key " with a command that inserts a pair of single-quotes or backquotes (tex-insert-quote). To be precise, this command inserts ‘‘‘’ after whitespace or an open brace, ‘"’ after a backslash, and ‘’’’ after any other character.

If you need the character ‘"’ itself in unusual contexts, use C-q to insert it. Also, " with a numeric argument always inserts that number of ‘"’ characters. You can turn off the feature of " expansion by eliminating that binding in the local map (See Section 31.4 [Key Bindings], page 474.)

In TEX mode, ‘$’ has a special syntax code which attempts to understand the way TEX math mode delimiters match. When you insert a ‘$’ that is meant to exit math mode, the position of the matching ‘$’ that entered math mode is displayed for a second. This is the same feature that displays the open brace that matches a close brace that is inserted. However, there is no way to tell whether a ‘$’ enters math mode or leaves it; so when you insert a ‘$’ that enters math mode, the previous ‘$’ position is shown as if it were a match, even though they are actually unrelated.

TEX uses braces as delimiters that must match. Some users prefer to keep braces balanced at all times, rather than inserting them singly. Use C-c { (tex-insert-braces) to insert a pair of braces. It leaves point between the two braces so you can insert the text that belongs inside. Afterward, use the command C-c } (up-list) to move forward past the close brace.

There are two commands for checking the matching of braces. C-j (tex-terminate-paragraph) checks the paragraph before point, and inserts two newlines to start a new paragraph. It prints a message in the echo area if any mismatch is found. M-x validate-tex-region checks a region, paragraph by paragraph. When it finds a paragraph that contains a mismatch, it displays point at the beginning of the paragraph for a few seconds and sets the mark at that spot. Scanning continues until the whole buffer has been checked or until you type another key. Afterward, you can use the mark ring to find the last several paragraphs that had mismatches (see Section 8.5 [Mark Ring], page 81).

Note that Emacs commands count square brackets and parentheses in TEX mode, not just braces. This is not strictly correct for the purpose of checking TEX syntax. However, parentheses and square brackets are likely to be used in text as matching delimiters and it is useful for the various motion commands and automatic match display to work with them.

21.9.2 LaTEX Editing Commands

LaTEX mode, and its variant, SliTEX mode, provide a few extra features not applicable to plain TEX.

C-c C-o Insert ‘\begin’ and ‘\end’ for LaTEX block and position point on a line between them. (tex-latex-block).

`C-c C-e` Close the innermost LaTeX block not yet closed (`tex-close-latex-block`).

In LaTeX input, '`\begin`' and '`\end`' commands are used to group blocks of text. To insert a '`\begin`' and a matching '`\end`' (on a new line following the '`\begin`'), use `C-c C-o` (`tex-latex-block`). A blank line is inserted between the two, and point is left there. You can use completion when you enter the block type; to specify additional block type names beyond the standard list, set the variable `latex-block-names`. For example, here's how to add '`theorem`', '`corollary`', and '`proof`':

```
(setq latex-block-names '("theorem" "corollary" "proof"))
```

In LaTeX input, '`\begin`' and '`\end`' commands must balance. You can use `C-c C-e` (`tex-close-latex-block`) to insert automatically a matching '`\end`' to match the last unmatched '`\begin`'. It indents the '`\end`' to match the corresponding '`\begin`'. It inserts a newline after '`\end`' if point is at the beginning of a line.

21.9.3 TeX Printing Commands

You can invoke TeX as an inferior of Emacs on either the entire contents of the buffer or just a region at a time. Running TeX in this way on just one chapter is a good way to see what your changes look like without taking the time to format the entire file.

`C-c C-r` Invoke TeX on the current region, together with the buffer's header (`tex-region`).

`C-c C-b` Invoke TeX on the entire current buffer (`tex-buffer`).

`C-c ⟨TAB⟩` Invoke BibTeX on the current file (`tex-bibtex-file`).

`C-c C-f` Invoke TeX on the current file (`tex-file`).

`C-c C-l` Recenter the window showing output from the inferior TeX so that the last line can be seen (`tex-recenter-output-buffer`).

`C-c C-k` Kill the TeX subprocess (`tex-kill-job`).

`C-c C-p` Print the output from the last `C-c C-r`, `C-c C-b`, or `C-c C-f` command (`tex-print`).

`C-c C-v` Preview the output from the last `C-c C-r`, `C-c C-b`, or `C-c C-f` command (`tex-view`).

`C-c C-q` Show the printer queue (`tex-show-print-queue`).

You can pass the current buffer through an inferior TeX by means of `C-c C-b` (`tex-buffer`). The formatted output appears in a temporary file; to print it, type `C-c C-p` (`tex-print`). Afterward, you can use `C-c C-q` (`tex-show-print-queue`) to view the progress of your output towards being

printed. If your terminal has the ability to display TEX output files, you can preview the output on the terminal with `C-c C-v` (`tex-view`).

You can specify the directory to use for running TEX by setting the variable `tex-directory`. `"."` is the default value. If your environment variable `TEXINPUTS` contains relative directory names, or if your files contains '`\input`' commands with relative file names, then `tex-directory` *must* be `"."` or you will get the wrong results. Otherwise, it is safe to specify some other directory, such as `"/tmp"`.

If you want to specify which shell commands are used in the inferior TEX, you can do so by setting the values of the variables `tex-run-command`, `latex-run-command`, `slitex-run-command`, `tex-dvi-print-command`, `tex-dvi-view-command`, and `tex-show-queue-command`. You *must* set the value of `tex-dvi-view-command` for your particular terminal; this variable has no default value. The other variables have default values that may (or may not) be appropriate for your system.

Normally, the file name given to these commands comes at the end of the command string; for example, '`latex` *filename*'. In some cases, however, the file name needs to be embedded in the command; an example is when you need to provide the file name as an argument to one command whose output is piped to another. You can specify where to put the file name with '`*`' in the command string. For example,

```
(setq tex-dvi-print-command "dvips -f * | lpr")
```

The terminal output from TEX, including any error messages, appears in a buffer called '`*tex-shell*`'. If TEX gets an error, you can switch to this buffer and feed it input (this works as in Shell mode; see Section 30.2.2 [Interactive Shell], page 427). Without switching to this buffer you can scroll it so that its last line is visible by typing `C-c C-l`.

Type `C-c C-k` (`tex-kill-job`) to kill the TEX process if you see that its output is no longer useful. Using `C-c C-b` or `C-c C-r` also kills any TEX process still running.

You can also pass an arbitrary region through an inferior TEX by typing `C-c C-r` (`tex-region`). This is tricky, however, because most files of TEX input contain commands at the beginning to set parameters and define macros, without which no later part of the file will format correctly. To solve this problem, `C-c C-r` allows you to designate a part of the file as containing essential commands; it is included before the specified region as part of the input to TEX. The designated part of the file is called the *header*.

To indicate the bounds of the header in Plain TEX mode, you insert two special strings in the file. Insert '`%**start of header`' before the header, and '`%**end of header`' after it. Each string must appear entirely on one line, but there may be other text on the line before or after. The lines containing the two strings are included in the header. If '`%**start of header`' does not appear within the first 100 lines of the buffer, `C-c C-r` assumes that there is no header.

In LaTeX mode, the header begins with '`\documentstyle`' and ends with '`\begin{document}`'. These are commands that LaTeX requires you to use in any case, so nothing special needs to be done to identify the header.

The commands (`tex-buffer`) and (`tex-region`) do all of their work in a temporary directory, and do not have available any of the auxiliary files needed by TeX for cross-references; these commands are generally not suitable for running the final copy in which all of the cross-references need to be correct.

When you want the auxiliary files for cross references, use `C-c C-f` (`tex-file`) which runs TeX on the current buffer's file, in that file's directory. Before running TeX, it offers to save any modified buffers. Generally, you need to use (`tex-file`) twice to get the cross-references right.

Large TeX documents are often split into several files—one main file, plus subfiles. Running TeX on a subfile typically does not work; you have to run it on the main file. In order to make `tex-file` useful when you are editing a subfile, you can set the variable `tex-main-file` to the name of the main file. Then `tex-file` runs TeX on that file.

The most convenient way to use `tex-main-file` is to specify it in a local variable list in each of the subfiles. See Section 31.2.5 [File Variables], page 468.

For LaTeX files, you can use BibTeX to process the auxiliary file for the current buffer's file. BibTeX looks up bibliographic citations in a data base and prepares the cited references for the bibliography section. The command `C-c TAB` (`tex-bibtex-file`) runs the shell command (`tex-bibtex-command`) to produce a '`.bbl`' file for the current buffer's file. Generally, you need to do `C-c C-f` (`tex-file`) once to generate the '`.aux`' file, then do `C-c TAB` (`tex-bibtex-file`), and then repeat `C-c C-f` (`tex-file`) twice more to get the cross-references correct.

Entering any kind of TeX mode runs the hooks `text-mode-hook` and `tex-mode-hook`. Then it runs either `plain-tex-mode-hook` or `latex-mode-hook`, whichever is appropriate. For SliTeX files, it calls `slitex-mode-hook`. Starting the TeX shell runs the hook `tex-shell-hook`. See Section 31.2.3 [Hooks], page 465.

21.10 Nroff Mode

Nroff mode is a mode like Text mode but modified to handle nroff commands present in the text. Invoke `M-x nroff-mode` to enter this mode. It differs from Text mode in only a few ways. All nroff command lines are considered paragraph separators, so that filling will never garble the nroff commands. Pages are separated by '`.bp`' commands. Comments start with backslash-doublequote. Also, three special commands are provided that are not in Text mode:

`M-n` Move to the beginning of the next line that isn't an nroff command (`forward-text-line`). An argument is a repeat count.

`M-p` Like `M-n` but move up (`backward-text-line`).

`M-?` Prints in the echo area the number of text lines (lines that are not nroff commands) in the region (`count-text-lines`).

The other feature of Nroff mode is that you can turn on Electric Nroff mode. This is a minor mode that you can turn on or off with `M-x electric-nroff-mode` (see Section 31.1 [Minor Modes], page 455). When the mode is on, each time you use ⟨RET⟩ to end a line that contains an nroff command that opens a kind of grouping, the matching nroff command to close that grouping is automatically inserted on the following line. For example, if you are at the beginning of a line and type `. ( b` ⟨RET⟩, this inserts the matching command '`.)b`' on a new line following point.

If you use Outline minor mode with Nroff mode (see Section 21.8 [Outline Mode], page 255), heading lines are lines of the form '`.H`' followed by a number (the header level).

Entering Nroff mode runs the hook `text-mode-hook`, followed by the hook `nroff-mode-hook` (see Section 31.2.3 [Hooks], page 465).

21.11 Editing Formatted Text

Enriched mode is a minor mode for editing files that contain formatted text in WYSIWYG fashion, as in a word processor. Currently, formatted text in Enriched mode can specify fonts, colors, underlining, margins, and types of filling and justification. In the future, we plan to implement other formatting features as well.

Enriched mode is a minor mode (see Section 31.1 [Minor Modes], page 455). Typically it is used in conjunction with Text mode (see Section 21.7 [Text Mode], page 255). However, you can also use it with other major modes such as Outline mode and Paragraph-Indent Text mode.

Potentially, Emacs can store formatted text files in various file formats. Currently, only one format is implemented: *text/enriched* format, which is defined by the MIME protocol. See section "Format Conversion" in *the Emacs Lisp Reference Manual*, for details of how Emacs recognizes and converts file formats.

The Emacs distribution contains a formatted text file that can serve as an example. Its name is '`etc/enriched.doc`'. It contains samples illustrating all the features described in this section. It also contains a list of ideas for future enhancements.

21.11.1 Requesting to Edit Formatted Text

Whenever you visit a file that Emacs saved in the text/enriched format, Emacs automatically converts the formatting information in the file into Emacs's own internal format (text properties), and turns on Enriched mode.

To create a new file of formatted text, first visit the nonexistent file, then type `M-x enriched-mode` before you start inserting text. This command turns on Enriched mode. Do this before you begin inserting text, to ensure that the text you insert is handled properly.

More generally, the command `enriched-mode` turns Enriched mode on if it was off, and off if it was on. With a prefix argument, this command turns Enriched mode on if the argument is positive, and turns the mode off otherwise.

When you save a buffer while Enriched mode is enabled in it, Emacs automatically converts the text to text/enriched format while writing it into the file. When you visit the file again, Emacs will automatically recognize the format, reconvert the text, and turn on Enriched mode again.

Normally, after visiting a file in text/enriched format, Emacs refills each paragraph to fit the specified right margin. You can turn off this refilling, to save time, by setting the variable `enriched-fill-after-visiting` to `nil` or to `ask`.

However, when visiting a file that was saved from Enriched mode, there is no need for refilling, because Emacs saves the right margin settings along with the text.

You can add annotations for saving additional text properties, which Emacs normally does not save, by adding to `enriched-translations`. Note that the text/enriched standard requires any non-standard annotations to have names starting with '`x-`', as in '`x-read-only`'. This ensures that they will not conflict with standard annotations that may be added later.

21.11.2 Hard and Soft Newlines

In formatted text, Emacs distinguishes between two different kinds of newlines, *hard* newlines and *soft* newlines.

Hard newlines are used to separate paragraphs, or items in a list, or anywhere that there should always be a line break regardless of the margins. The `RET` command (`newline`) and `C-o` (`open-line`) insert hard newlines.

Soft newlines are used to make text fit between the margins. All the fill commands, including Auto Fill, insert soft newlines—and they delete only soft newlines.

Although hard and soft newlines look the same, it is important to bear the difference in mind. Do not use ⟨RET⟩ to break lines in the middle of filled paragraphs, or else you will get hard newlines that are barriers to further filling. Instead, let Auto Fill mode break lines, so that if the text or

the margins change, Emacs can refill the lines properly. See Section 21.5.1 [Auto Fill], page 248.

On the other hand, in tables and lists, where the lines should always remain as you type them, you can use ⟨RET⟩ to end lines. For these lines, you may also want to set the justification style to `unfilled`. See Section 21.11.7 [Format Justification], page 270.

21.11.3 Editing Format Information

There are two ways to alter the formatting information for a formatted text file: with keyboard commands, and with the mouse.

The easiest way to add properties to your document is by using the Text Properties menu. You can get to this menu in two ways: from the Edit menu in the menu bar, or with `C-mouse-2` (hold the ⟨CTRL⟩ key and press the middle mouse button).

Most of the items in the Text Properties menu lead to other submenus. These are described in the sections that follow. Some items run commands directly:

`Remove Properties`
: Delete from the region all the text properties that the Text Properties menu works with (`facemenu-remove-props`).

`Remove All`
: Delete *all* text properties from the region (`facemenu-remove-all`).

`List Properties`
: List all the text properties of the character following point (`list-text-properties-at`).

`Display Faces`
: Display a list of all the defined faces.

`Display Colors`
: Display a list of all the defined colors.

21.11.4 Faces in Formatted Text

The Faces submenu lists various Emacs faces including `bold`, `italic`, and `underline`. Selecting one of these adds the chosen face to the region. See Section 11.1 [Faces], page 103. You can also specify a face with these keyboard commands:

`M-g d`
: Set the region, or the next inserted character, to the `default` face (`facemenu-set-default`).

`M-g b` Set the region, or the next inserted character, to the `bold` face (`facemenu-set-bold`).

`M-g i` Set the region, or the next inserted character, to the `italic` face (`facemenu-set-italic`).

`M-g l` Set the region, or the next inserted character, to the `bold-italic` face (`facemenu-set-bold-italic`).

`M-g u` Set the region, or the next inserted character, to the `underline` face (`facemenu-set-underline`).

`M-g o` *face* ⟨RET⟩
Set the region, or the next inserted character, to the face *face* (`facemenu-set-face`).

If you use these commands with a prefix argument—or, in Transient Mark mode, if the region is not active—then these commands specify a face to use for your next self-inserting input. See Section 8.2 [Transient Mark], page 78. This applies to both the keyboard commands and the menu commands.

Enriched mode defines two additional faces: `excerpt` and `fixed`. These correspond to codes used in the text/enriched file format.

The `excerpt` face is intended for quotations. This face is the same as `italic` unless you customize it (see Section 31.2.2.3 [Face Customization], page 463).

The `fixed` face is meant to say, "Use a fixed-width font for this part of the text." Emacs currently supports only fixed-width fonts; therefore, the `fixed` annotation is not necessary now. However, we plan to support variable width fonts in future Emacs versions, and other systems that display text/enriched format may not use a fixed-width font as the default. So if you specifically want a certain part of the text to use a fixed-width font, you should specify the `fixed` face for that part.

The `fixed` face is normally defined to use a different font from the default. However, systems have different fonts installed, you may need to customize this.

If your terminal cannot display different faces, you will not be able to see them, but you can still edit documents containing faces. You can even add faces and colors to documents. They will be visible when the file is viewed on a terminal that can display them.

21.11.5 Colors in Formatted Text

You can specify foreground and background colors for portions of the text. There is a menu for specifying the foreground color and a menu for specifying the background color. Each color menu lists all the colors that you have used in Enriched mode in the current Emacs session.

If you specify a color with a prefix argument—or, in Transient Mark mode, if the region is not active—then it applies to your next self-inserting input. See Section 8.2 [Transient Mark], page 78. Otherwise, the command applies to the region.

Each color menu contains one additional item: '`Other`'. You can use this item to specify a color that is not listed in the menu; it reads the color name with the minibuffer. To display list of available colors and their names, use the '`Display Colors`' menu item in the Text Properties menu (see Section 21.11.3 [Editing Format Info], page 267).

Any color that you specify in this way, or that is mentioned in a formatted text file that you read in, is added to both color menus for the duration of the Emacs session.

There are no key bindings for specifying colors, but you can do so with the extended commands `M-x facemenu-set-foreground` and `M-x facemenu-set-background`. Both of these commands read the name of the color with the minibuffer.

21.11.6 Indentation in Formatted Text

When editing formatted text, you can specify different amounts of indentation for the right or left margin of an entire paragraph or a part of a paragraph. The margins you specify automatically affect the Emacs fill commands (see Section 21.5 [Filling], page 248) and line-breaking commands.

The Indentation submenu provides a convenient interface for specifying these properties. The submenu contains four items:

`Indent More`
: Indent the region by 4 columns (`increase-left-margin`). In Enriched mode, this command is also available on `C-x` ⟨TAB⟩; if you supply a numeric argument, that says how many columns to add to the margin (a negative argument reduces the number of columns).

`Indent Less`
: Remove 4 columns of indentation from the region.

`Indent Right More`
: Make the text narrower by indenting 4 columns at the right margin.

`Indent Right Less`
: Remove 4 columns of indentation from the right margin.

You can use these commands repeatedly to increase or decrease the indentation.

The most common way to use these commands is to change the indentation of an entire paragraph. However, that is not the only use. You can

change the margins at any point; the new values take effect at the end of the line (for right margins) or the beginning of the next line (for left margins).

This makes it possible to format paragraphs with *hanging indents*, which means that the first line is indented less than subsequent lines. To set up a hanging indent, increase the indentation of the region starting after the first word of the paragraph and running until the end of the paragraph.

Indenting the first line of a paragraph is easier. Set the margin for the whole paragraph where you want it to be for the body of the paragraph, then indent the first line by inserting extra spaces or tabs.

Sometimes, as a result of editing, the filling of a paragraph becomes messed up—parts of the paragraph may extend past the left or right margins. When this happens, use `M-q` (`fill-paragraph`) to refill the paragraph.

The variable `standard-indent` specifies how many columns these commands should add to or subtract from the indentation. The default value is 4.

The overall default right margin for Enriched mode is controlled by the variable `fill-column`, as usual.

21.11.7 Justification in Formatted Text

When editing formatted text, you can specify various styles of justification for a paragraph. The style you specify automatically affects the Emacs fill commands.

The Justification submenu provides a convenient interface for specifying the style. The submenu contains five items:

`Flush Left`
: This is the most common style of justification (at least for English). Lines are aligned at the left margin but left uneven at the right.

`Flush Right`
: This aligns each line with the right margin. Spaces and tabs are added on the left, if necessary, to make lines line up on the right.

`Full`
: This justifies the text, aligning both edges of each line. Justified text looks very nice in a printed book, where the spaces can all be adjusted equally, but it does not look as nice with a fixed-width font on the screen. Perhaps a future version of Emacs will be able to adjust the width of spaces in a line to achieve elegant justification.

`Center`
: This centers every line between the current margins.

`None`
: This turns off filling entirely. Each line will remain as you wrote it; the fill and auto-fill functions will have no effect on text which has this setting. You can, however, still indent the left margin.

In unfilled regions, all newlines are treated as hard newlines (see Section 21.11.2 [Hard and Soft Newlines], page 266) .

In Enriched mode, you can also specify justification from the keyboard using the `M-j` prefix character:

`M-j l` Make the region left-filled (`set-justification-left`).

`M-j r` Make the region right-filled (`set-justification-right`).

`M-j f` Make the region fully-justified (`set-justification-full`).

`M-j c`
`M-S` Make the region centered (`set-justification-center`).

`M-j u` Make the region unfilled (`set-justification-none`).

Justification styles apply to entire paragraphs. All the justification-changing commands operate on the paragraph containing point, or, if the region is active, on all paragraphs which overlap the region.

The default justification style is specified by the variable `default-justification`. Its value should be one of the symbols `left`, `right`, `full`, `center`, or `none`.

21.11.8 Setting Other Text Properties

The Other Properties menu lets you add or remove three other useful text properties: `read-only`, `invisible` and `intangible`. The `intangible` property disallows moving point within the text, the `invisible` text property hides text from display, and the `read-only` property disallows alteration of the text.

Each of these special properties has a menu item to add it to the region. The last menu item, '`Remove Special`', removes all of these special properties from the text in the region.

Currently, the `invisible` and `intangible` properties are *not* saved in the text/enriched format. The `read-only` property is saved, but it is not a standard part of the text/enriched format, so other editors may not respect it.

21.11.9 Forcing Enriched Mode

Normally, Emacs knows when you are editing formatted text because it recognizes the special annotations used in the file that you visited. However, there are situations in which you must take special actions to convert file contents or turn on Enriched mode:

- When you visit a file that was created with some other editor, Emacs may not recognize the file as being in the text/enriched format. In this case, when you visit the file you will see the formatting commands rather than the formatted text. Type `M-x format-decode-buffer` to translate it.
- When you *insert* a file into a buffer, rather than visiting it. Emacs does the necessary conversions on the text which you insert, but it does not enable Enriched mode. If you wish to do that, type `M-x enriched-mode`.

The command `format-decode-buffer` translates text in various formats into Emacs's internal format. It asks you to specify the format to translate from; however, normally you can type just ⟨RET⟩, which tells Emacs to guess the format.

If you wish to look at text/enriched file in its raw form, as a sequence of characters rather than as formatted text, use the `M-x find-file-literally` command. This visits a file, like `find-file`, but does not do format conversion. It also inhibits character code conversion (see Section 18.6 [Coding Systems], page 223) and automatic uncompression (see Section 14.11 [Compressed Files], page 182). To disable format conversion but allow character code conversion and/or automatic uncompression if appropriate, use `format-find-file` with suitable arguments.

22 Editing Programs

Emacs has many commands designed to understand the syntax of programming languages such as Lisp and C. These commands can

- Move over or kill balanced expressions or *sexps* (see Section 22.2 [Lists], page 274).
- Move over or mark top-level expressions—*defuns*, in Lisp; functions, in C (see Section 22.4 [Defuns], page 277).
- Show how parentheses balance (see Section 22.6 [Matching], page 290).
- Insert, kill or align comments (see Section 22.7 [Comments], page 291).
- Follow the usual indentation conventions of the language (see Section 22.5 [Program Indent], page 278).

The commands for words, sentences and paragraphs are very useful in editing code even though their canonical application is for editing human language text. Most symbols contain words (see Section 21.1 [Words], page 243); sentences can be found in strings and comments (see Section 21.2 [Sentences], page 245). Paragraphs per se don't exist in code, but the paragraph commands are useful anyway, because programming language major modes define paragraphs to begin and end at blank lines (see Section 21.3 [Paragraphs], page 246). Judicious use of blank lines to make the program clearer will also provide useful chunks of text for the paragraph commands to work on.

The selective display feature is useful for looking at the overall structure of a function (see Section 11.9 [Selective Display], page 113). This feature hides the lines that are indented more than a specified amount. Programming modes often support Outline minor mode (see Section 21.8 [Outline Mode], page 255). The Foldout package provides folding-editor features (see ⟨undefined⟩ [Foldout], page ⟨undefined⟩).

The "automatic typing" features may be useful for writing programs. See section "" in *Autotyping*.

22.1 Major Modes for Programming Languages

Emacs also has major modes for the programming languages Lisp, Scheme (a variant of Lisp) and the Scheme-based DSSSL expression language, Ada, Awk, C, C++, Delphi (Object Pascal), Fortran (free and fixed format), Icon, IDLWAVE, Java, Metafont (TEX's companion for font creation), Modula2, Objective-C, Octave, Pascal, Perl, Pike, PostScript, Prolog, Simula, VHDL, CORBA IDL, and Tcl. There is also a major mode for makefiles, called Makefile mode. An alternative mode for Perl is called CPerl mode. Modes

are available for scripts for the common Unix shells, VMS DCL and MS-DOS/MS-Windows '`BAT`' files. In a similar fashion to programming languages, modes are provided for editing various sorts of configuration files.

Separate manuals are available for the modes for Ada (see section "Ada Mode" in *Ada Mode*), C/C++/Objective C/Java/Corba IDL (see section "CC Mode" in *CC Mode*) and the IDLWAVE modes (see section "IDLWAVE" in *IDLWAVE User Manual*).

Ideally, a major mode should be implemented for each programming language that you might want to edit with Emacs; but often the mode for one language can serve for other syntactically similar languages. The language modes that exist are those that someone decided to take the trouble to write.

There are several forms of Lisp mode, which differ in the way they interface to Lisp execution. See Section 23.6 [Executing Lisp], page 339.

Each of the programming language major modes defines the ⟨TAB⟩ key to run an indentation function that knows the indentation conventions of that language and updates the current line's indentation accordingly. For example, in C mode ⟨TAB⟩ is bound to `c-indent-line`. `C-j` is normally defined to do ⟨RET⟩ followed by ⟨TAB⟩; thus, it too indents in a mode-specific fashion.

In most programming languages, indentation is likely to vary from line to line. So the major modes for those languages rebind ⟨DEL⟩ to treat a tab as if it were the equivalent number of spaces (using the command `backward-delete-char-untabify`). This makes it possible to rub out indentation one column at a time without worrying whether it is made up of spaces or tabs. Use `C-b C-d` to delete a tab character before point, in these modes.

Programming language modes define paragraphs to be separated only by blank lines, so that the paragraph commands remain useful. Auto Fill mode, if enabled in a programming language major mode, indents the new lines which it creates.

Turning on a major mode runs a normal hook called the *mode hook*, which is the value of a Lisp variable. Each major mode has a mode hook, and the hook's name is always made from the mode command's name by adding '`-hook`'. For example, turning on C mode runs the hook `c-mode-hook`, while turning on Lisp mode runs the hook `lisp-mode-hook`. See Section 31.2.3 [Hooks], page 465.

22.2 Lists and Sexps

By convention, Emacs keys for dealing with balanced expressions are usually Control-Meta characters. They tend to be analogous in function to their Control and Meta equivalents. These commands are usually thought of as pertaining to expressions in programming languages, but can be useful

with any language in which some sort of parentheses exist (including human languages).

These commands fall into two classes. Some deal only with *lists* (parenthetical groupings). They see nothing except parentheses, brackets, braces (whichever ones must balance in the language you are working with), and escape characters that might be used to quote those.

The other commands deal with expressions or *sexps*. The word "sexp" is derived from *s-expression*, the ancient term for an expression in Lisp. But in Emacs, the notion of "sexp" is not limited to Lisp. It refers to an expression in whatever language your program is written in. Each programming language has its own major mode, which customizes the syntax tables so that expressions in that language count as sexps.

Sexps typically include symbols, numbers, and string constants, as well as anything contained in parentheses, brackets or braces.

In languages that use prefix and infix operators, such as C, it is not possible for all expressions to be sexps. For example, C mode does not recognize '`foo + bar`' as a sexp, even though it *is* a C expression; it recognizes '`foo`' as one sexp and '`bar`' as another, with the '`+`' as punctuation between them. This is a fundamental ambiguity: both '`foo + bar`' and '`foo`' are legitimate choices for the sexp to move over if point is at the '`f`'. Note that '`(foo + bar)`' is a single sexp in C mode.

Some languages have obscure forms of expression syntax that nobody has bothered to make Emacs understand properly.

22.3 List And Sexp Commands

`C-M-f`	Move forward over a sexp (`forward-sexp`).
`C-M-b`	Move backward over a sexp (`backward-sexp`).
`C-M-k`	Kill sexp forward (`kill-sexp`).
`C-M-`⟨DEL⟩	Kill sexp backward (`backward-kill-sexp`).
`C-M-u`	Move up and backward in list structure (`backward-up-list`).
`C-M-d`	Move down and forward in list structure (`down-list`).
`C-M-n`	Move forward over a list (`forward-list`).
`C-M-p`	Move backward over a list (`backward-list`).
`C-M-t`	Transpose expressions (`transpose-sexps`).
`C-M-@`	Put mark after following expression (`mark-sexp`).

To move forward over a sexp, use `C-M-f` (`forward-sexp`). If the first significant character after point is an opening delimiter ('`(`' in Lisp; '`(`',

‘[’ or ‘{’ in C), C-M-f moves past the matching closing delimiter. If the character begins a symbol, string, or number, C-M-f moves over that.

The command C-M-b (backward-sexp) moves backward over a sexp. The detailed rules are like those above for C-M-f, but with directions reversed. If there are any prefix characters (single-quote, backquote and comma, in Lisp) preceding the sexp, C-M-b moves back over them as well. The sexp commands move across comments as if they were whitespace in most modes.

C-M-f or C-M-b with an argument repeats that operation the specified number of times; with a negative argument, it moves in the opposite direction.

Killing a whole sexp can be done with C-M-k (kill-sexp) or C-M-⟨DEL⟩ (backward-kill-sexp). C-M-k kills the characters that C-M-f would move over, and C-M-⟨DEL⟩ kills the characters that C-M-b would move over.

The *list commands* move over lists, as the sexp commands do, but skip blithely over any number of other kinds of sexps (symbols, strings, etc.). They are C-M-n (forward-list) and C-M-p (backward-list). The main reason they are useful is that they usually ignore comments (since the comments usually do not contain any lists).

C-M-n and C-M-p stay at the same level in parentheses, when that’s possible. To move *up* one (or *n*) levels, use C-M-u (backward-up-list). C-M-u moves backward up past one unmatched opening delimiter. A positive argument serves as a repeat count; a negative argument reverses direction of motion and also requests repetition, so it moves forward and up one or more levels.

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

A somewhat random-sounding command which is nevertheless handy is C-M-t (transpose-sexps), which drags the previous sexp across the next one. An argument serves as a repeat count, and a negative argument drags backwards (thus canceling out the effect of C-M-t with a positive argument). An argument of zero, rather than doing nothing, transposes the sexps ending after point and the mark.

To set the region around the next sexp in the buffer, use C-M-@ (mark-sexp), which sets mark at the same place that C-M-f would move to. C-M-@ takes arguments like C-M-f. In particular, a negative argument is useful for putting the mark at the beginning of the previous sexp.

The list and sexp commands’ understanding of syntax is completely controlled by the syntax table. Any character can, for example, be declared to be an opening delimiter and act like an open parenthesis. See Section 31.6 [Syntax], page 485.

22.4 Defuns

In Emacs, a parenthetical grouping at the top level in the buffer is called a *defun*. The name derives from the fact that most top-level lists in a Lisp file are instances of the special form `defun`, but any top-level parenthetical grouping counts as a defun in Emacs parlance regardless of what its contents are, and regardless of the programming language in use. For example, in C, the body of a function definition is a defun.

`C-M-a` Move to beginning of current or preceding defun (`beginning-of-defun`).

`C-M-e` Move to end of current or following defun (`end-of-defun`).

`C-M-h` Put region around whole current or following defun (`mark-defun`).

The commands to move to the beginning and end of the current defun are `C-M-a` (`beginning-of-defun`) and `C-M-e` (`end-of-defun`).

If you wish to operate on the current defun, use `C-M-h` (`mark-defun`) which puts point at the beginning and mark at the end of the current or next defun. For example, this is the easiest way to get ready to move the defun to a different place in the text. In C mode, `C-M-h` runs the function `c-mark-function`, which is almost the same as `mark-defun`; the difference is that it backs up over the argument declarations, function name and returned data type so that the entire C function is inside the region. See Section 8.4 [Marking Objects], page 81.

Emacs assumes that any open-parenthesis found in the leftmost column is the start of a defun. Therefore, **never put an open-parenthesis at the left margin in a Lisp file unless it is the start of a top-level list. Never put an open-brace or other opening delimiter at the beginning of a line of C code unless it starts the body of a function.** The most likely problem case is when you want an opening delimiter at the start of a line inside a string. To avoid trouble, put an escape character (‘\’, in C and Emacs Lisp, ‘/’ in some other Lisp dialects) before the opening delimiter. It will not affect the contents of the string.

In the remotest past, the original Emacs found defuns by moving upward a level of parentheses until there were no more levels to go up. This always required scanning all the way back to the beginning of the buffer, even for a small function. To speed up the operation, Emacs was changed to assume that any ‘`(`’ (or other character assigned the syntactic class of opening-delimiter) at the left margin is the start of a defun. This heuristic is nearly always right and avoids the costly scan; however, it mandates the convention described above.

22.5 Indentation for Programs

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

Emacs also provides a Lisp pretty-printer in the library `pp`. This program reformats a Lisp object with indentation chosen to look nice.

22.5.1 Basic Program Indentation Commands

⟨TAB⟩ Adjust indentation of current line.

`C-j` Equivalent to ⟨RET⟩ followed by ⟨TAB⟩ (`newline-and-indent`).

The basic indentation command is ⟨TAB⟩, which gives the current line the correct indentation as determined from the previous lines. The function that ⟨TAB⟩ runs depends on the major mode; it is `lisp-indent-line` in Lisp mode, `c-indent-line` in C mode, etc. These functions understand different syntaxes for different languages, but they all do about the same thing. ⟨TAB⟩ in any programming-language major mode inserts or deletes whitespace at the beginning of the current line, independent of where point is in the line. If point is inside the whitespace at the beginning of the line, ⟨TAB⟩ leaves it at the end of that whitespace; otherwise, ⟨TAB⟩ leaves point fixed with respect to the characters around it.

Use `C-q` ⟨TAB⟩ to insert a tab at point.

When entering lines of new code, use `C-j` (`newline-and-indent`), which is equivalent to a ⟨RET⟩ followed by a ⟨TAB⟩. `C-j` creates a blank line and then gives it the appropriate indentation.

⟨TAB⟩ indents the second and following lines of the body of a parenthetical grouping each under the preceding one; therefore, if you alter one line's indentation to be nonstandard, the lines below will tend to follow it. This behavior is convenient in cases where you have overridden the standard result of ⟨TAB⟩ because you find it unaesthetic for a particular line.

Remember that an open-parenthesis, open-brace or other opening delimiter at the left margin is assumed by Emacs (including the indentation routines) to be the start of a function. Therefore, you must never have an opening delimiter in column zero that is not the beginning of a function, not even inside a string. This restriction is vital for making the indentation commands fast; you must simply accept it. See Section 22.4 [Defuns], page 277, for more information on this.

22.5.2 Indenting Several Lines

When you wish to reindent several lines of code which have been altered or moved to a different level in the list structure, you have several commands available.

`C-M-q` Reindent all the lines within one list (`indent-sexp`).

`C-u` ⟨TAB⟩ Shift an entire list rigidly sideways so that its first line is properly indented.

`C-M-\` Reindent all lines in the region (`indent-region`).

You can reindent the contents of a single list by positioning point before the beginning of it and typing `C-M-q` (`indent-sexp` in Lisp mode, `c-indent-exp` in C mode; also bound to other suitable commands in other modes). The indentation of the line the sexp starts on is not changed; therefore, only the relative indentation within the list, and not its position, is changed. To correct the position as well, type a ⟨TAB⟩ before the `C-M-q`.

If the relative indentation within a list is correct but the indentation of its first line is not, go to that line and type `C-u` ⟨TAB⟩. ⟨TAB⟩ with a numeric argument reindents the current line as usual, then reindents by the same amount all the lines in the grouping starting on the current line. In other words, it reindents the whole grouping rigidly as a unit. It is clever, though, and does not alter lines that start inside strings, or C preprocessor lines when in C mode.

Another way to specify the range to be reindented is with the region. The command `C-M-\` (`indent-region`) applies ⟨TAB⟩ to every line whose first character is between point and mark.

22.5.3 Customizing Lisp Indentation

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

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

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

The standard pattern is overridden for certain functions. Functions whose names start with `def` always indent the second line by `lisp-body-indent` extra columns beyond the open-parenthesis starting the expression.

The standard pattern can be overridden in various ways for individual functions, according to the `lisp-indent-function` property of the function name. There are four possibilities for this property:

`nil` This is the same as no property; the standard indentation pattern is used.

`defun` The pattern used for function names that start with `def` is used for this function also.

a number, *number*
: The first *number* arguments of the function are *distinguished* arguments; the rest are considered the *body* of the expression. A line in the expression is indented according to whether the first argument on it is distinguished or not. If the argument is part of the body, the line is indented `lisp-body-indent` more columns than the open-parenthesis starting the containing expression. If the argument is distinguished and is either the first or second argument, it is indented *twice* that many extra columns. If the argument is distinguished and not the first or second argument, the standard pattern is followed for that line.

a symbol, *symbol*
: *symbol* should be a function name; that function is called to calculate the indentation of a line within this expression. The function receives two arguments:

  state The value returned by `parse-partial-sexp` (a Lisp primitive for indentation and nesting computation) when it parses up to the beginning of this line.

  pos The position at which the line being indented begins.

  It should return either a number, which is the number of columns of indentation for that line, or a list whose car is such a number. The difference between returning a number and returning a list is that a number says that all following lines at the same nesting level should be indented just like this one; a list says that following lines might call for different indentations. This makes a difference when the indentation is being computed by `C-M-q`; if the value is a number, `C-M-q` need not recalculate indentation for the following lines until the end of the list.

22.5.4 Commands for C Indentation

Here are the commands for indentation in C mode and related modes:

`C-c C-q` Reindent the current top-level function definition or aggregate type declaration (`c-indent-defun`).

`C-M-q` Reindent each line in the balanced expression that follows point (`c-indent-exp`). A prefix argument inhibits error checking and warning messages about invalid syntax.

⟨TAB⟩ Reindent the current line, and/or in some cases insert a tab character (`c-indent-command`).

If `c-tab-always-indent` is `t`, this command always reindents the current line and does nothing else. This is the default.

If that variable is `nil`, this command reindents the current line only if point is at the left margin or in the line's indentation; otherwise, it inserts a tab (or the equivalent number of spaces, if `indent-tabs-mode` is `nil`).

Any other value (not `nil` or `t`) means always reindent the line, and also insert a tab if within a comment, a string, or a preprocessor directive.

`C-u` ⟨TAB⟩ Reindent the current line according to its syntax; also rigidly reindent any other lines of the expression that starts on the current line. See Section 22.5.2 [Multi-line Indent], page 279.

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

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

22.5.5 Customizing C Indentation

C mode and related modes use a simple yet flexible mechanism for customizing indentation. The mechanism works in two steps: first it classifies the line syntactically according to its contents and context; second, it associates each kind of syntactic construct with an indentation offset which you can customize.

22.5.5.1 Step 1—Syntactic Analysis

In the first step, the C indentation mechanism looks at the line before the one you are currently indenting and determines the syntactic components of the construct on that line. It builds a list of these syntactic components,

each of which contains a *syntactic symbol* and sometimes also a buffer position. Some syntactic symbols describe grammatical elements, for example `statement` and `substatement`; others describe locations amidst grammatical elements, for example `class-open` and `knr-argdecl`.

Conceptually, a line of C code is always indented relative to the indentation of some line higher up in the buffer. This is represented by the buffer positions in the syntactic component list.

Here is an example. Suppose we have the following code in a C++ mode buffer (the line numbers don't actually appear in the buffer):

```
1: void swap (int& a, int& b)
2: {
3:   int tmp = a;
4:   a = b;
5:   b = tmp;
6: }
```

If you type `C-c C-s` (which runs the command `c-show-syntactic-information`) on line 4, it shows the result of the indentation mechanism for that line:

```
((statement . 32))
```

This indicates that the line is a statement and it is indented relative to buffer position 32, which happens to be the '`i`' in `int` on line 3. If you move the cursor to line 3 and type `C-c C-s`, it displays this:

```
((defun-block-intro . 28))
```

This indicates that the `int` line is the first statement in a block, and is indented relative to buffer position 28, which is the brace just after the function header.

Here is another example:

```
1: int add (int val, int incr, int doit)
2: {
3:   if (doit)
4:     {
5:       return (val + incr);
6:     }
7:   return (val);
8: }
```

Typing `C-c C-s` on line 4 displays this:

```
((substatement-open . 43))
```

This says that the brace *opens* a substatement block. By the way, a *substatement* indicates the line after an `if`, `else`, `while`, `do`, `switch`, `for`, `try`, `catch`, `finally`, or `synchronized` statement.

Within the C indentation commands, after a line has been analyzed syntactically for indentation, the variable `c-syntactic-context` contains a list that describes the results. Each element in this list is a *syntactic component*:

a cons cell containing a syntactic symbol and (optionally) its corresponding buffer position. There may be several elements in a component list; typically only one element has a buffer position.

22.5.5.2 Step 2—Indentation Calculation

The C indentation mechanism calculates the indentation for the current line using the list of syntactic components, `c-syntactic-context`, derived from syntactic analysis. Each component is a cons cell that contains a syntactic symbol and may also contain a buffer position.

Each component contributes to the final total indentation of the line in two ways. First, the syntactic symbol identifies an element of `c-offsets-alist`, which is an association list mapping syntactic symbols into indentation offsets. Each syntactic symbol's offset adds to the total indentation. Second, if the component includes a buffer position, the column number of that position adds to the indentation. All these offsets and column numbers, added together, give the total indentation.

The following examples demonstrate the workings of the C indentation mechanism:

```
1: void swap (int& a, int& b)
2: {
3:   int tmp = a;
4:   a = b;
5:   b = tmp;
6: }
```

Suppose that point is on line 3 and you type ⟨TAB⟩ to reindent the line. As explained above (see Section 22.5.5.1 [Syntactic Analysis], page 281), the syntactic component list for that line is:

```
((defun-block-intro . 28))
```

In this case, the indentation calculation first looks up `defun-block-intro` in the `c-offsets-alist` alist. Suppose that it finds the integer 2; it adds this to the running total (initialized to zero), yielding a updated total indentation of 2 spaces.

The next step is to find the column number of buffer position 28. Since the brace at buffer position 28 is in column zero, this adds 0 to the running total. Since this line has only one syntactic component, the total indentation for the line is 2 spaces.

```
1: int add (int val, int incr, int doit)
2: {
3:   if (doit)
4:     {
5:       return(val + incr);
6:     }
```

```
7:   return(val);
8: }
```

If you type ⟨TAB⟩ on line 4, the same process is performed, but with different data. The syntactic component list for this line is:

```
((substatement-open . 43))
```

Here, the indentation calculation's first job is to look up the symbol `substatement-open` in `c-offsets-alist`. Let's assume that the offset for this symbol is 2. At this point the running total is 2 (0 + 2 = 2). Then it adds the column number of buffer position 43, which is the '`i`' in `if` on line 3. This character is in column 2 on that line. Adding this yields a total indentation of 4 spaces.

If a syntactic symbol in the analysis of a line does not appear in `c-offsets-alist`, it is ignored; if in addition the variable `c-strict-syntax-p` is non-`nil`, it is an error.

22.5.5.3 Changing Indentation Style

There are two ways to customize the indentation style for the C-like modes. First, you can select one of several predefined styles, each of which specifies offsets for all the syntactic symbols. For more flexibility, you can customize the handling of individual syntactic symbols. See Section 22.5.5.4 [Syntactic Symbols], page 285, for a list of all defined syntactic symbols.

`M-x c-set-style` ⟨RET⟩ *style* ⟨RET⟩
: Select predefined indentation style *style*. Type `?` when entering *style* to see a list of supported styles; to find out what a style looks like, select it and reindent some C code.

`C-c C-o` *symbol* ⟨RET⟩ *offset* ⟨RET⟩
: Set the indentation offset for syntactic symbol *symbol* (`c-set-offset`). The second argument *offset* specifies the new indentation offset.

The `c-offsets-alist` variable controls the amount of indentation to give to each syntactic symbol. Its value is an association list, and each element of the list has the form (*syntactic-symbol* . *offset*). By changing the offsets for various syntactic symbols, you can customize indentation in fine detail. To change this alist, use `c-set-offset` (see below).

Each offset value in `c-offsets-alist` can be an integer, a function or variable name, a list, or one of the following symbols: `+`, `-`, `++`, `--`, `*`, or `/`, indicating positive or negative multiples of the variable `c-basic-offset`. Thus, if you want to change the levels of indentation to be 3 spaces instead of 2 spaces, set `c-basic-offset` to 3.

Using a function as the offset value provides the ultimate flexibility in customizing indentation. The function is called with a single argument con-

taining the `cons` of the syntactic symbol and the buffer position, if any. The function should return an integer offset.

If the offset value is a list, its elements are processed according to the rules above until a non-`nil` value is found. That value is then added to the total indentation in the normal manner. The primary use for this is to combine the results of several functions.

The command `C-c C-o` (`c-set-offset`) is the easiest way to set offsets, both interactively or in your '`~/.emacs`' file. First specify the syntactic symbol, then the offset you want. See Section 22.5.5.4 [Syntactic Symbols], page 285, for a list of valid syntactic symbols and their meanings.

22.5.5.4 Syntactic Symbols

Here is a table of valid syntactic symbols for indentation in C and related modes, with their syntactic meanings. Normally, most of these symbols are assigned offsets in `c-offsets-alist`.

`string` Inside a multi-line string.

`c` Inside a multi-line C style block comment.

`defun-open`
On a brace that opens a function definition.

`defun-close`
On a brace that closes a function definition.

`defun-block-intro`
In the first line in a top-level defun.

`class-open`
On a brace that opens a class definition.

`class-close`
On a brace that closes a class definition.

`inline-open`
On a brace that opens an in-class inline method.

`inline-close`
On a brace that closes an in-class inline method.

`extern-lang-open`
On a brace that opens an external language block.

`extern-lang-close`
On a brace that closes an external language block.

`func-decl-cont`
The region between a function definition's argument list and the defun opening brace (excluding K&R function definitions).

In C, you cannot put anything but whitespace and comments between them; in C++ and Java, `throws` declarations and other things can appear in this context.

`knr-argdecl-intro`
On the first line of a K&R C argument declaration.

`knr-argdecl`
In one of the subsequent lines in a K&R C argument declaration.

`topmost-intro`
On the first line in a topmost construct definition.

`topmost-intro-cont`
On the topmost definition continuation lines.

`member-init-intro`
On the first line in a member initialization list.

`member-init-cont`
On one of the subsequent member initialization list lines.

`inher-intro`
On the first line of a multiple inheritance list.

`inher-cont`
On one of the subsequent multiple inheritance lines.

`block-open`
On a statement block open brace.

`block-close`
On a statement block close brace.

`brace-list-open`
On the opening brace of an `enum` or `static` array list.

`brace-list-close`
On the closing brace of an `enum` or `static` array list.

`brace-list-intro`
On the first line in an `enum` or `static` array list.

`brace-list-entry`
On one of the subsequent lines in an `enum` or `static` array list.

`brace-entry-open`
On one of the subsequent lines in an `enum` or `static` array list, when the line begins with an open brace.

`statement`
On an ordinary statement.

`statement-cont`
On a continuation line of a statement.

`statement-block-intro`
: On the first line in a new statement block.

`statement-case-intro`
: On the first line in a `case` "block."

`statement-case-open`
: On the first line in a `case` block starting with brace.

`inexpr-statement`
: On a statement block inside an expression. This is used for a GNU extension to the C language, and for Pike special functions that take a statement block as an argument.

`inexpr-class`
: On a class definition inside an expression. This is used for anonymous classes and anonymous array initializers in Java.

`substatement`
: On the first line after an `if`, `while`, `for`, `do`, or `else`.

`substatement-open`
: On the brace that opens a substatement block.

`case-label`
: On a `case` or `default` label.

`access-label`
: On a C++ `private`, `protected`, or `public` access label.

`label`
: On any ordinary label.

`do-while-closure`
: On the `while` that ends a `do-while` construct.

`else-clause`
: On the `else` of an `if-else` construct.

`catch-clause`
: On the `catch` and `finally` lines in `try...catch` constructs in C++ and Java.

`comment-intro`
: On a line containing only a comment introduction.

`arglist-intro`
: On the first line in an argument list.

`arglist-cont`
: On one of the subsequent argument list lines when no arguments follow on the same line as the arglist opening parenthesis.

`arglist-cont-nonempty`
: On one of the subsequent argument list lines when at least one argument follows on the same line as the arglist opening parenthesis.

`arglist-close`
: On the closing parenthesis of an argument list.

`stream-op`
: On one of the lines continuing a stream operator construct.

`inclass`
: On a construct that is nested inside a class definition. The indentation is relative to the open brace of the class definition.

`inextern-lang`
: On a construct that is nested inside an external language block.

`inexpr-statement`
: On the first line of statement block inside an expression. This is used for the GCC extension to C that uses the syntax `({ ... })`. It is also used for the special functions that takes a statement block as an argument in Pike.

`inexpr-class`
: On the first line of a class definition inside an expression. This is used for anonymous classes and anonymous array initializers in Java.

`cpp-macro`
: On the start of a cpp macro.

`friend`
: On a C++ `friend` declaration.

`objc-method-intro`
: On the first line of an Objective-C method definition.

`objc-method-args-cont`
: On one of the lines continuing an Objective-C method definition.

`objc-method-call-cont`
: On one of the lines continuing an Objective-C method call.

`inlambda`
: Like `inclass`, but used inside lambda (i.e. anonymous) functions. Only used in Pike.

`lambda-intro-cont`
: On a line continuing the header of a lambda function, between the `lambda` keyword and the function body. Only used in Pike.

22.5.5.5 Variables for C Indentation

This section describes additional variables which control the indentation behavior of C mode and related mode.

`c-offsets-alist`
: Association list of syntactic symbols and their indentation offsets. You should not set this directly, only with `c-set-offset`.

See Section 22.5.5.3 [Changing Indent Style], page 284, for details.

`c-style-alist`
: Variable for defining indentation styles; see below.

`c-basic-offset`
: Amount of basic offset used by `+` and `-` symbols in `c-offsets-alist`.

`c-special-indent-hook`
: Hook for user-defined special indentation adjustments. This hook is called after a line is indented by C mode and related modes.

The variable `c-style-alist` specifies the predefined indentation styles. Each element has form (*name variable-setting*`...`), where *name* is the name of the style. Each *variable-setting* has the form (*variable* `.` *value*); *variable* is one of the customization variables used by C mode, and *value* is the value for that variable when using the selected style.

When *variable* is `c-offsets-alist`, that is a special case: *value* is appended to the front of the value of `c-offsets-alist` instead of replacing that value outright. Therefore, it is not necessary for *value* to specify each and every syntactic symbol—only those for which the style differs from the default.

The indentation of lines containing only comments is also affected by the variable `c-comment-only-line-offset` (see Section 22.19.5 [Comments in C], page 322).

22.5.5.6 C Indentation Styles

A *C style* is a collection of indentation style customizations. Emacs comes with several predefined indentation styles for C and related modes, including `gnu`, `k&r`, `bsd`, `stroustrup`, `linux`, `python`, `java`, `whitesmith`, `ellemtel`, `cc-mode`, and `user`.

To choose the style you want, use the command `M-x c-set-style`. Specify a style name as an argument (case is not significant in C style names). The chosen style only affects newly visited buffers, not those you are already editing. You can also set the variable `c-default-style` to specify the style for various major modes. Its value should be an alist, in which each element specifies one major mode and which indentation style to use for it. For example,

```
(setq c-default-style
      '((java-mode . "java") (other . "gnu")))
```

specifies an explicit choice for Java mode, and the default ‘`gnu`’ style for the other C-like modes.

The style `gnu` defines the formatting recommend by the GNU Project; it is the default, so as to encourage the indentation we recommend. However, if you make changes in variables such as `c-basic-offset` and `c-offsets-alist` in your '`~/.emacs`' file, your changes override the what `gnu` style says.

To define a new C indentation style, call the function `c-add-style`:

`(c-add-style` *name values use-now*`)`

Here *name* is the name of the new style (a string), and *values* is an alist whose elements have the form `(`*variable* `.` *value*`)`. The variables you specify should be among those documented in Section 22.5.5.5 [Variables for C Indent], page 288.

If *use-now* is non-`nil`, `c-add-style` selects the new style after defining it.

22.6 Automatic Display Of Matching Parentheses

The Emacs parenthesis-matching feature is designed to show automatically how parentheses match in the text. Whenever you type a self-inserting character that is a closing delimiter, the cursor moves momentarily to the location of the matching opening delimiter, provided that is on the screen. If it is not on the screen, some text near it is displayed in the echo area. Either way, you can tell what grouping is being closed off.

In Lisp, automatic matching applies only to parentheses. In C, it applies to braces and brackets too. Emacs knows which characters to regard as matching delimiters based on the syntax table, which is set by the major mode. See Section 31.6 [Syntax], page 485.

If the opening delimiter and closing delimiter are mismatched—such as in '`[x)`'—a warning message is displayed in the echo area. The correct matches are specified in the syntax table.

Three variables control parenthesis match display. `blink-matching-paren` turns the feature on or off; `nil` turns it off, but the default is `t` to turn match display on. `blink-matching-delay` says how many seconds to wait; the default is 1, but on some systems it is useful to specify a fraction of a second. `blink-matching-paren-distance` specifies how many characters back to search to find the matching opening delimiter. If the match is not found in that far, scanning stops, and nothing is displayed. This is to prevent scanning for the matching delimiter from wasting lots of time when there is no match. The default is 12,000.

Show Paren mode provides a more powerful kind of automatic parenthesis matching. Whenever point is after a close parenthesis, the close parenthesis and its matching open parenthesis are both highlighted; otherwise, if point is before an open parenthesis, the matching close parenthesis is highlighted. (There is no need to highlight the open parenthesis after point

because the cursor appears on top of that character.) Use the command `M-x show-paren-mode` to enable or disable this mode.

By default, `show-paren-mode` uses colors to highlight the parentheses. However, if your display doesn't support colors, you can customize the faces `show-paren-match-face` and `show-paren-mismatch-face` to use other attributes, such as bold or underline. See Section 31.2.2.3 [Face Customization], page 463.

22.7 Manipulating Comments

Because comments are such an important part of programming, Emacs provides special commands for editing and inserting comments.

22.7.1 Comment Commands

The comment commands in this table insert, kill and align comments. They are described in this section and following sections.

`M-;` Insert or realign comment on current line; alternatively, comment or uncomment the region (`comment-dwim`).

`C-u M-;` Kill comment on current line (`comment-kill`).

`C-x ;` Set comment column (`set-comment-column`).

`C-M-j` Like ⟨RET⟩ followed by inserting and aligning a comment (`indent-new-comment-line`).

`M-x comment-region`
Add or remove comment delimiters on all the lines in the region.

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

If there is no comment already on the line, `M-;` inserts a new comment, aligned at a specific column called the *comment column*. The new comment begins with the string Emacs thinks comments should start with (the value of `comment-start`; see below). Point is after that string, so you can insert the text of the comment right away. If the major mode has specified a string to terminate comments, `M-;` inserts that too, to keep the syntax valid.

If the text of the line extends past the comment column, then the comment start string is indented to a suitable boundary (usually, at least one space is inserted).

You can also use `M-;` to align an existing comment. If a line already contains the comment-start string, `M-;` reindents it to the conventional alignment and moves point after it. (Exception: comments starting in column 0 are not moved.) Even when an existing comment is properly aligned, `M-;` is still useful for moving directly to the start of the text inside the comment.

`C-u M-;` kills any comment on the current line, along with the whitespace before it. To reinsert the comment on another line, move to the end of that line, do `C-y`, and then do `M-;` to realign it.

Note that `C-u M-;` is not a distinct key; it is `M-;` (`comment-dwim`) with a prefix argument. That command is programmed so that when it receives a prefix argument it calls `comment-kill`. However, `comment-kill` is a valid command in its own right, and you can bind it directly to a key if you wish.

`M-;` does two other jobs when used with an active region in Transient Mark mode (see Section 8.2 [Transient Mark], page 78). Then it either adds or removes comment delimiters on each line of the region. (If every line is a comment, it removes comment delimiters from each; otherwise, it adds comment delimiters to each.) If you are not using Transient Mark mode, then you should use the commands `comment-region` and `uncomment-region` to do these jobs (see Section 22.7.2 [Multi-Line Comments], page 292). A prefix argument used in these circumstances specifies how many comment delimiters to add or how many to delete.

Some major modes have special rules for indenting certain kinds of comments in certain contexts. For example, in Lisp code, comments which start with two semicolons are indented as if they were lines of code, instead of at the comment column. Comments which start with three semicolons are supposed to start at the left margin. Emacs understands these conventions by indenting a double-semicolon comment using ⟨TAB⟩, and by not changing the indentation of a triple-semicolon comment at all.

```
;; 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.
```

In C code, a comment preceded on its line by nothing but whitespace is indented like a line of code.

22.7.2 Multiple Lines of Comments

If you are typing a comment and wish to continue it on another line, you can use the command `C-M-j` (`indent-new-comment-line`). This terminates the comment you are typing, creates a new blank line afterward, and begins a new comment indented under the old one. When Auto Fill mode is on,

going past the fill column while typing a comment causes the comment to be continued in just this fashion. If point is not at the end of the line when `C-M-j` is typed, the text on the rest of the line becomes part of the new comment line.

To turn existing lines into comment lines, use the `M-x comment-region` command. It adds comment delimiters to the lines that start in the region, thus commenting them out. With a negative argument, it does the opposite—it deletes comment delimiters from the lines in the region.

With a positive argument, `comment-region` duplicates the last character of the comment start sequence it adds; the argument specifies how many copies of the character to insert. Thus, in Lisp mode, `C-u 2 M-x comment-region` adds '`;;`' to each line. Duplicating the comment delimiter is a way of calling attention to the comment. It can also affect how the comment is indented. In Lisp, for proper indentation, you should use an argument of two, if between defuns, and three, if within a defun.

22.7.3 Options Controlling Comments

The comment column is stored in the variable `comment-column`. You can set it to a number explicitly. Alternatively, the command `C-x ;` (`set-comment-column`) sets the comment column to the column point is at. `C-u C-x ;` sets the comment column to match the last comment before point in the buffer, and then does a `M-;` to align the current line's comment under the previous one.

The variable `comment-column` is per-buffer: setting the variable in the normal fashion affects only the current buffer, but there is a default value which you can change with `setq-default`. See Section 31.2.4 [Locals], page 466. Many major modes initialize this variable for the current buffer.

The comment commands recognize comments based on the regular expression that is the value of the variable `comment-start-skip`. Make sure this regexp does not match the null string. It may match more than the comment starting delimiter in the strictest sense of the word; for example, in C mode the value of the variable is `"/\\*+ *"`, which matches extra stars and spaces after the '`/*`' itself. (Note that '`\\`' is needed in Lisp syntax to include a '`\`' in the string, which is needed to deny the first star its special meaning in regexp syntax. See Section 12.5 [Regexps], page 125.)

When a comment command makes a new comment, it inserts the value of `comment-start` to begin it. The value of `comment-end` is inserted after point, so that it will follow the text that you will insert into the comment. In C mode, `comment-start` has the value `"/* "` and `comment-end` has the value `" */"`.

The variable `comment-padding` specifies how many spaces `comment-region` should insert on each line between the comment delimiter and the line's original text. The default is 1.

The variable `comment-multi-line` controls how `C-M-j` (`indent-new-comment-line`) behaves when used inside a comment. If `comment-multi-line` is `nil`, as it normally is, then the comment on the starting line is terminated and a new comment is started on the new following line. If `comment-multi-line` is not `nil`, then the new following line is set up as part of the same comment that was found on the starting line. This is done by not inserting a terminator on the old line, and not inserting a starter on the new line. In languages where multi-line comments work, the choice of value for this variable is a matter of taste.

The variable `comment-indent-function` should contain a function that will be called to compute the indentation for a newly inserted comment or for aligning an existing comment. It is set differently by various major modes. The function is called with no arguments, but with point at the beginning of the comment, or at the end of a line if a new comment is to be inserted. It should return the column in which the comment ought to start. For example, in Lisp mode, the indent hook function bases its decision on how many semicolons begin an existing comment, and on the code in the preceding lines.

22.8 Editing Without Unbalanced Parentheses

`M-(` Put parentheses around next sexp(s) (`insert-parentheses`).

`M-)` Move past next close parenthesis and reindent (`move-past-close-and-reindent`).

The commands `M-(` (`insert-parentheses`) and `M-)` (`move-past-close-and-reindent`) are designed to facilitate a style of editing which keeps parentheses balanced at all times. `M-(` inserts a pair of parentheses, either together as in '`()`', or, if given an argument, around the next several sexps. It leaves point after the open parenthesis. The command `M-)` moves past the close parenthesis, deleting any indentation preceding it, and indenting with `C-j` after it.

For example, instead of typing `( F O O )`, you can type `M-( F O O`, which has the same effect except for leaving the cursor before the close parenthesis.

`M-(` may insert a space before the open parenthesis, depending on the syntax class of the preceding character. Set `parens-require-spaces` to `nil` value if you wish to inhibit this.

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

22.9 Completion for Symbol Names

Usually completion happens in the minibuffer. But one kind of completion is available in all buffers: completion for symbol names.

The character `M-`⟨TAB⟩ runs a command to complete the partial symbol before point against the set of meaningful symbol names. Any additional characters determined by the partial name are inserted at point.

If the partial name in the buffer has more than one possible completion and they have no additional characters in common, a list of all possible completions is displayed in another window.

In most programming language major modes, `M-`⟨TAB⟩ runs the command `complete-symbol`, which provides two kinds of completion. Normally it does completion based on a tags table (see Section 22.16 [Tags], page 301); with a numeric argument (regardless of the value), it does completion based on the names listed in the Info file indexes for your language. Thus, to complete the name of a symbol defined in your own program, use `M-`⟨TAB⟩ with no argument; to complete the name of a standard library function, use `C-u M-`⟨TAB⟩. Of course, Info-based completion works only if there is an Info file for the standard library functions of your language, and only if it is installed at your site.

In Emacs-Lisp mode, the name space for completion normally consists of nontrivial symbols present in Emacs—those that have function definitions, values or properties. However, if there is an open-parenthesis immediately before the beginning of the partial symbol, only symbols with function definitions are considered as completions. The command which implements this is `lisp-complete-symbol`.

In Text mode and related modes, `M-`⟨TAB⟩ completes words based on the spell-checker's dictionary. See Section 13.4 [Spelling], page 139.

22.10 Which Function Mode

Which Function mode is a minor mode that displays the current function name in the mode line, as you move around in a buffer.

To enable (or disable) Which Function mode, use the command `M-x which-function-mode`. This command is global; it applies to all buffers, both existing ones and those yet to be created. However, this only affects certain major modes, those listed in the value of `which-func-modes`. (If the value is `t`, then Which Function mode applies to all major modes that know how to support it—which are the major modes that support Imenu.)

22.11 Hideshow minor mode

Hideshow minor mode provides selective display of portions of a file, known as *blocks*. You can use `M-x hs-minor-mode` to enable or disable this mode, or add `hs-minor-mode` to the mode hook for certain major modes in order to enable it automatically for those modes.

Just what constitutes a block depends on the major mode. In C mode or C++ mode, they are delimited by braces, while in Lisp mode and similar modes they are delimited by parentheses. Multi-line comments also count as blocks.

`C-c C-h` Hide the current block (`hs-hide-block`).

`C-c C-s` Show the current block (`hs-show-block`).

`C-c C-c` Either hide or show the current block (`hs-toggle-hiding`)

`S-Mouse-2`
: Either hide or show the block you click on (`hs-mouse-toggle-hiding`)

`C-c C-M-h`
: Hide all top-level blocks (`hs-hide-all`).

`C-c C-M-s`
: Show everything in the buffer (`hs-show-all`).

`C-c C-l` Hide all blocks *n* levels below this block (`hs-hide-level`).

These user options exist for customizing Hideshow mode.

`hs-hide-comments-when-hiding-all`
: Non-`nil` says that `hs-hide-all` should hide comments too.

`hs-show-hidden-short-form`
: Non-`nil` says to omit the last line in a form (saving screen space).

`hs-isearch-open`
: Specifies what kind of hidden blocks to open in Isearch mode.

`hs-special-modes-alist`
: Specifies Initializes Hideshow variables for different modes.

22.12 Glasses minor mode

Glasses minor mode makes '`unreadableIdentifiersLikeThis`' readable by altering the display. It can do this in two different ways: by displaying underscores between an lower-case letter and the following capital letter, or by emboldening the capital letters. It does not alter the buffer text, only the way they display, so you can use it even on read-only buffers. You can use

the command `M-x glasses-mode` to enable or disable the mode; you can also add `glasses-mode` to the mode hook of appropriate programming language major modes.

22.13 Documentation Commands

As you edit Lisp code to be run in Emacs, the commands `C-h f` (`describe-function`) and `C-h v` (`describe-variable`) can be used to print documentation of functions and variables that you want to call. These commands use the minibuffer to read the name of a function or variable to document, and display the documentation in a window.

For extra convenience, these commands provide default arguments based on the code in the neighborhood of point. `C-h f` sets the default to the function called in the innermost list containing point. `C-h v` uses the symbol name around or adjacent to point as its default.

For Emacs Lisp code, you can also use Eldoc mode. This minor mode constantly displays in the echo area the argument list for the function being called at point. (In other words, it finds the function call that point is contained in, and displays the argument list of that function.) Eldoc mode applies in Emacs Lisp and Lisp Interaction modes only. Use the command `M-x eldoc-mode` to enable or disable this feature.

For C, Lisp, and other languages, you can use `C-h C-i` (`info-lookup-symbol`) to view the Info documentation for a symbol. You specify the symbol with the minibuffer; by default, it uses the symbol that appears in the buffer at point. The major mode determines where to look for documentation for the symbol—which Info files and which indices. You can also use `M-x info-lookup-file` to look for documentation for a file name. Currently this supports the following modes: Awk, Autoconf, Bison, C, Emacs Lisp, LaTeX, M4, Makefile, Octave, Perl, Scheme and Texinfo, provided you have installed the relevant Info files, which are typically available with the appropriate GNU package.

You can read the "man page" for an operating system command, library function, or system call, with the `M-x manual-entry` command. It runs the `man` program to format the man page, and runs it asynchronously if your system permits, so that you can keep on editing while the page is being formatted. (MS-DOS and MS-Windows 3 do not permit asynchronous subprocesses, so on these systems you cannot edit while Emacs waits for `man` to exit.) The result goes in a buffer named '`*Man` *topic*`*`'. These buffers use a special major mode, Man mode, that facilitates scrolling and examining other manual pages. For details, type `C-h m` while in a man page buffer.

Man pages are classified into *sections*; sometimes there are man pages with the same name in different sections. To read a man page from a specific section, type '*topic*`(`*section*`)`' or '*section topic*' when `M-x manual-entry` prompts for the topic. For example, to read the man page for the C library

function `chmod` (as opposed to a command by the same name), type `M-x manual-entry` ⟨RET⟩ `chmod(2v)` ⟨RET⟩ (assuming `chmod` is in section '`2v`').

If you do not specify a section, the results depend on how the `man` command works on your system. Some of them display only the first man page they find. Others display all man pages that have the specified name, so you can page between them with the `M-n` and `M-p` keys. The mode line shows how many manual pages are available in the Man buffer.

For a long man page, setting the faces properly can take substantial time. By default, Emacs uses faces in man pages if Emacs can display different fonts or colors. You can turn off use of faces in man pages by setting the variable `Man-fontify-manpage-flag` to `nil`.

If you insert the text of a man page into an Emacs buffer in some other fashion, you can use the command `M-x Man-fontify-manpage` to perform the same conversions that `M-x manual-entry` does.

An alternative way of reading manual pages is the `M-x woman` command[1]. Unlike `M-x man`, it does not run any external programs to format and display the man pages; instead it does the job in Emacs Lisp, so it works on systems such as MS-Windows, where the `man` program and other the programs it needs are not readily available. `M-x woman` prompts for a name of a manual page, and provides completion based on the list of manual pages that are installed on your machine; the list of available manual pages is computed automatically the first time you invoke `woman`. The word at point in the current buffer is used to suggest the default name of the manual page.

With a numeric argument, `M-x woman` recomputes the list of the manual pages used for completion. This is useful if you add or delete manual pages.

If you type a name of a manual page and `M-x woman` finds that several manual pages by the same name exist in different sections, it pops up a window with possible candidates asking you to choose one of them.

By default, `M-x woman` looks up the manual pages in directories listed by the `MANPATH` environment variable. (If `MANPATH` is not set, `woman` uses a suitable default value, which can be customized.) More precisely, `woman` looks for subdirectories that match the shell wildcard '`man*`' in each one of these directories, and tries to find the manual pages in those subdirectories. When first invoked, `M-x woman` converts the value of `MANPATH` to a list of directory names and stores that list in the `woman-manpath` variable. By changing the value of this variable, you can customize the list of directories where `woman` looks for manual pages.

In addition, you can augment the list of directories searched by `woman` by setting the value of the `woman-path` variable. This variable should hold a list of specific directories which `woman` should search, in addition to those in `woman-manpath`. Unlike `woman-manpath`, the directories in `woman-path` are searched for the manual pages, not for '`man*`' subdirectories.

[1] The name of the command, `woman`, is an acronym for "w/o (without) man," since it doesn't use the `man` program.

Occasionally, you might need to display manual pages that are not in any of the directories listed by `woman-manpath` and `woman-path`. The `M-x woman-find-file` command prompts for a name of a manual page file, with completion, and then formats and displays that file like `M-x woman` does.

First time you invoke `M-x woman`, it defines the Dired `W` key to run the `woman-find-file` command on the current line's file. You can disable this by setting the variable `woman-dired-keys` to `nil`. See Chapter 28 [Dired], page 387. In addition, the Tar-mode `w` key is bound to `woman-find-file` on the current line's archive member.

For more information about setting up and using `M-x woman`, see section "Browse UN*X Manual Pages WithOut Man" in *The WoMan Manual*.

Eventually the GNU project hopes to replace most man pages with better-organized manuals that you can browse with Info. See Section 7.7 [Misc Help], page 75. Since this process is only partially completed, it is still useful to read manual pages.

22.14 Change Logs

The Emacs command `C-x 4 a` adds a new entry to the change log file for the file you are editing (`add-change-log-entry-other-window`). If that file is actually a backup file, it makes an entry appropriate for the file's parent—that is useful for making log entries for functions that have been deleted in the current version.

A change log file contains a chronological record of when and why you have changed a program, consisting of a sequence of entries describing individual changes. Normally it is kept in a file called '`ChangeLog`' in the same directory as the file you are editing, or one of its parent directories. A single '`ChangeLog`' file can record changes for all the files in its directory and all its subdirectories.

A change log entry starts with a header line that contains your name, your email address (taken from the variable `user-mail-address`), and the current date and time. Aside from these header lines, every line in the change log starts with a space or a tab. The bulk of the entry consists of *items*, each of which starts with a line starting with whitespace and a star. Here are two entries, both dated in May 1993, each with two items:

```
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.
```

One entry can describe several changes; each change should have its own item. Normally there should be a blank line between items. When items are related (parts of the same change, in different places), group them by leaving no blank line between them. The second entry above contains two items grouped in this way.

`C-x 4 a` visits the change log file and creates a new entry unless the most recent entry is for today's date and your name. It also creates a new item for the current file. For many languages, it can even guess the name of the function or other object that was changed.

When the option `add-log-keep-changes-together` is non-`nil`, `C-x 4 a` adds to any existing entry for the file rather than starting a new entry.

If the value of the variable `change-log-version-info-enabled` is non-`nil`, `C-x 4 a` ads the file's version number to the change log entry. It finds the version number by searching the first ten percent of the file, using regular expressions from the variable `change-log-version-number-regexp-list`.

The change log file is visited in Change Log mode. In this major mode, each bunch of grouped items counts as one paragraph, and each entry is considered a page. This facilitates editing the entries. `C-j` and auto-fill indent each new line like the previous line; this is convenient for entering the contents of an entry.

You can use the command `M-x change-log-merge` to merge other log files into a buffer in Change Log Mode, preserving the date ordering of entries.

Versions of Emacs before 20.1 used a different format for the time of the change log entry:

```
Fri May 25 11:23:23 1993 Richard Stallman  <rms@gnu.org>
```

The `M-x change-log-redate` command converts all the old-style date entries in the change log file visited in the current buffer to the new format, to make the file uniform in style. This is handy when entries are contributed by many different people, some of whom use old versions of Emacs.

Version control systems are another way to keep track of changes in your program and keep a change log. See ⟨undefined⟩ [Log Buffer], page ⟨undefined⟩.

22.15 'AUTHORS' files

Programs which have many contributors usually include a file named '`AUTHORS`' in their distribution, which lists the individual contributions. Emacs has a special command for maintaining the '`AUTHORS`' file that is part of the Emacs distribution.

The `M-x authors` command prompts for the name of the root of the Emacs source directory. It then scans '`ChageLog`' files and Lisp source files under that directory for information about authors of individual packages and people who made changes in source files, and puts the information it gleans into a buffer named '`*Authors*`'. You can then edit the contents of that buffer and merge it with the exisiting '`AUTHORS`' file.

Do not assume that this command finds all the contributors; don't assume that a person not listed in the output was not a contributor. If you merged in someone's contribution and did not put his name in the change log, he won't show up in `M-x authors` either.

22.16 Tags Tables

A *tags table* is a description of how a multi-file program is broken up into files. It lists the names of the component files and the names and positions of the functions (or other named subunits) in each file. Grouping the related files makes it possible to search or replace through all the files with one command. Recording the function names and positions makes possible the `M-.` command which finds the definition of a function by looking up which of the files it is in.

Tags tables are stored in files called *tags table files*. The conventional name for a tags table file is '`TAGS`'.

Each entry in the tags table records the name of one tag, the name of the file that the tag is defined in (implicitly), and the position in that file of the tag's definition.

Just what names from the described files are recorded in the tags table depends on the programming language of the described file. They normally include all file names, functions and subroutines, and may also include global variables, data types, and anything else convenient. Each name recorded is called a *tag*.

See also the Ebrowse facility, which is tailored for C++. See section "" in *Ebrowse User's Manual*.

22.16.1 Source File Tag Syntax

Here is how tag syntax is defined for the most popular languages:

- In C code, any C function or typedef is a tag, and so are definitions of `struct`, `union` and `enum`. `#define` macro definitions and `enum` constants are also tags, unless you specify '`--no-defines`' when making the tags table. Similarly, global variables are tags, unless you specify '`--no-globals`'. Use of '`--no-globals`' and '`--no-defines`' can make the tags table file much smaller.

You can tag function declarations and external variables in addition to function definitions by giving the '`--declarations`' option to `etags`.

- In C++ code, in addition to all the tag constructs of C code, member functions are also recognized, and optionally member variables if you use the '`--members`' option. Tags for variables and functions in classes are named '*class*`::`*variable*' and '*class*`::`*function*'. `operator` definitions have tag names like '`operator+`'.
- In Java code, tags include all the constructs recognized in C++, plus the `interface`, `extends` and `implements` constructs. Tags for variables and functions in classes are named '*class*`.`*variable*' and '*class*`.`*function*'.
- In LaTeX text, the argument of any of the commands `\chapter`, `\section`, `\subsection`, `\subsubsection`, `\eqno`, `\label`, `\ref`, `\cite`, `\bibitem`, `\part`, `\appendix`, `\entry`, or `\index`, is a tag.

  Other commands can make tags as well, if you specify them in the environment variable `TEXTAGS` before invoking `etags`. The value of this environment variable should be a colon-separated list of command names. For example,

  ```
  TEXTAGS="def:newcommand:newenvironment"
  export TEXTAGS
  ```

  specifies (using Bourne shell syntax) that the commands '`\def`', '`\newcommand`' and '`\newenvironment`' also define tags.
- In Lisp code, any function defined with `defun`, any variable defined with `defvar` or `defconst`, and in general the first argument of any expression that starts with '`(def`' in column zero, is a tag.
- In Scheme code, tags include anything defined with `def` or with a construct whose name starts with '`def`'. They also include variables set with `set!` at top level in the file.

Several other languages are also supported:

- In Ada code, functions, procedures, packages, tasks, and types are tags. Use the '`--packages-only`' option to create tags for packages only.

  In Ada, the same name can be used for different kinds of entity (e.g., for a procedure and for a function). Also, for things like packages, procedures and functions, there is the spec (i.e. the interface) and the body (i.e. the implementation). To make it easier to pick the definition you want, Ada tag name have suffixes indicating the type of entity:

  '`/b`' package body.

  '`/f`' function.

  '`/k`' task.

  '`/p`' procedure.

  '`/s`' package spec.

‘`/t`’ type.

Thus, `M-x find-tag` ⟨RET⟩ `bidule/b` ⟨RET⟩ will go directly to the body of the package `bidule`, while `M-x find-tag` ⟨RET⟩ `bidule` ⟨RET⟩ will just search for any tag `bidule`.

- In assembler code, labels appearing at the beginning of a line, followed by a colon, are tags.
- In Bison or Yacc input files, each rule defines as a tag the nonterminal it constructs. The portions of the file that contain C code are parsed as C code.
- In Cobol code, tags are paragraph names; that is, any word starting in column 8 and followed by a period.
- In Erlang code, the tags are the functions, records, and macros defined in the file.
- In Fortran code, functions, subroutines and blockdata are tags.
- In makefiles, targets are tags.
- In Objective C code, tags include Objective C definitions for classes, class categories, methods, and protocols.
- In Pascal code, the tags are the functions and procedures defined in the file.
- In Perl code, the tags are the procedures defined by the `sub`, `my` and `local` keywords. Use ‘`--globals`’ if you want to tag global variables.
- In PostScript code, the tags are the functions.
- In Prolog code, a tag name appears at the left margin.
- In Python code, `def` or `class` at the beginning of a line generate a tag.

You can also generate tags based on regexp matching (see Section 22.16.3 [Etags Regexps], page 305) to handle other formats and languages.

22.16.2 Creating Tags Tables

The `etags` program is used to create a tags table file. It knows the syntax of several languages, as described in the previous section. Here is how to run `etags`:

`etags` *inputfiles*`...`

The `etags` program reads the specified files, and writes a tags table named ‘`TAGS`’ in the current working directory.

If the specified files don’t exist, `etags` looks for compressed versions of them and uncompresses them to read them. Under MS-DOS, `etags` also looks for file names like ‘`mycode.cgz`’ if it is given ‘`mycode.c`’ on the command line and ‘`mycode.c`’ does not exist.

`etags` recognizes the language used in an input file based on its file name and contents. You can specify the language with the '`--language=`*name*' option, described below.

If the tags table data become outdated due to changes in the files described in the table, the way to update the tags table is the same way it was made in the first place. But it is not necessary to do this very often.

If the tags table fails to record a tag, or records it for the wrong file, then Emacs cannot possibly find its definition. However, if the position recorded in the tags table becomes a little bit wrong (due to some editing in the file that the tag definition is in), the only consequence is a slight delay in finding the tag. Even if the stored position is very wrong, Emacs will still find the tag, but it must search the entire file for it.

So you should update a tags table when you define new tags that you want to have listed, or when you move tag definitions from one file to another, or when changes become substantial. Normally there is no need to update the tags table after each edit, or even every day.

One tags table can virtually include another. Specify the included tags file name with the '`--include=`*file*' option when creating the file that is to include it. The latter file then acts as if it covered all the source files specified in the included file, as well as the files it directly contains.

If you specify the source files with relative file names when you run `etags`, the tags file will contain file names relative to the directory where the tags file was initially written. This way, you can move an entire directory tree containing both the tags file and the source files, and the tags file will still refer correctly to the source files.

If you specify absolute file names as arguments to `etags`, then the tags file will contain absolute file names. This way, the tags file will still refer to the same files even if you move it, as long as the source files remain in the same place. Absolute file names start with '`/`', or with '*device*`:/`' on MS-DOS and MS-Windows.

When you want to make a tags table from a great number of files, you may have problems listing them on the command line, because some systems have a limit on its length. The simplest way to circumvent this limit is to tell `etags` to read the file names from its standard input, by typing a dash in place of the file names, like this:

```
find . -name "*.[chCH]" -print | etags -
```

Use the option '`--language=`*name*' to specify the language explicitly. You can intermix these options with file names; each one applies to the file names that follow it. Specify '`--language=auto`' to tell `etags` to resume guessing the language from the file names and file contents. Specify '`--language=none`' to turn off language-specific processing entirely; then `etags` recognizes tags by regexp matching alone (see Section 22.16.3 [Etags Regexps], page 305).

‘`etags --help`’ prints the list of the languages `etags` knows, and the file name rules for guessing the language. It also prints a list of all the available `etags` options, together with a short explanation.

22.16.3 Etags Regexps

The ‘`--regex`’ option provides a general way of recognizing tags based on regexp matching. You can freely intermix it with file names. Each ‘`--regex`’ option adds to the preceding ones, and applies only to the following files. The syntax is:

```
--regex=/tagregexp[/nameregexp]/
```

where *tagregexp* is used to match the lines to tag. It is always anchored, that is, it behaves as if preceded by ‘`^`’. If you want to account for indentation, just match any initial number of blanks by beginning your regular expression with ‘`[ \t]*`’. In the regular expressions, ‘`\`’ quotes the next character, and ‘`\t`’ stands for the tab character. Note that `etags` does not handle the other C escape sequences for special characters.

The syntax of regular expressions in `etags` is the same as in Emacs, augmented with the *interval operator*, which works as in `grep` and `ed`. The syntax of an interval operator is ‘`\{`*m*`,`*n*`\}`’, and its meaning is to match the preceding expression at least *m* times and up to *n* times.

You should not match more characters with *tagregexp* than that needed to recognize what you want to tag. If the match is such that more characters than needed are unavoidably matched by *tagregexp* (as will usually be the case), you should add a *nameregexp*, to pick out just the tag. This will enable Emacs to find tags more accurately and to do completion on tag names more reliably. You can find some examples below.

The option ‘`--ignore-case-regex`’ (or ‘`-c`’) works like ‘`--regex`’, except that matching ignores case. This is appropriate for certain programming languages.

The ‘`-R`’ option deletes all the regexps defined with ‘`--regex`’ options. It applies to the file names following it, as you can see from the following example:

```
etags --regex=/reg1/ voo.doo --regex=/reg2/ \
    bar.ber -R --lang=lisp los.er
```

Here `etags` chooses the parsing language for ‘`voo.doo`’ and ‘`bar.ber`’ according to their contents. `etags` also uses *reg1* to recognize additional tags in ‘`voo.doo`’, and both *reg1* and *reg2* to recognize additional tags in ‘`bar.ber`’. `etags` uses the Lisp tags rules, and no regexp matching, to recognize tags in ‘`los.er`’.

You can specify a regular expression for a particular language, by writing ‘`{lang}`’ in front of it. Then `etags` will use the regular expression only for files of that language. (‘`etags --help`’ prints the list of languages recognised

by `etags`.) The following example tags the `DEFVAR` macros in the Emacs source files, for the C language only:

```
--regex='{c}/[ \t]*DEFVAR_[A-Z_ \t(]+"\([^"]+\)"/'
```

This feature is particularly useful when you store a list of regular expressions in a file. The following option syntax instructs `etags` to read two files of regular expressions. The regular expressions contained in the second file are matched without regard to case.

```
--regex=@first-file --ignore-case-regex=@second-file
```

A regex file contains one regular expressions per line. Empty lines, and lines beginning with space or tab are ignored. When the first character in a line is '`@`', `etags` assumes that the rest of the line is the name of a file of regular expressions; thus, one such file can include another file. All the other lines are taken to be regular expressions. If the first non-whitespace text on the line is '`--`', that line is a comment.

For example, one can create a file called '`emacs.tags`' with the following contents:

```
        -- This is for GNU Emacs C source files
{c}/[ \t]*DEFVAR_[A-Z_ \t(]+"\([^"]+\)"/\1/
```

and then use it like this:

```
etags --regex=@emacs.tags *.[ch] */*.[ch]
```

Here are some more examples. The regexps are quoted to protect them from shell interpretation.

- Tag Octave files:

  ```
  etags --language=none \
        --regex='/[ \t]*function.*=[ \t]*\([^ \t]*\)[ \t]*(/\1/' \
        --regex='/###key \(.*\)/\1/' \
        --regex='/[ \t]*global[ \t].*/' \
        *.m
  ```

  Note that tags are not generated for scripts, so that you have to add a line by yourself of the form '`###key` *scriptname*' if you want to jump to it.

- Tag Tcl files:

  ```
  etags --language=none --regex='/proc[ \t]+\([^ \t]+\)/\1/' *.tcl
  ```

- Tag VHDL files:

  ```
  etags --language=none \
    --regex='/[ \t]*\(ARCHITECTURE\|CONFIGURATION\) +[^ ]* +OF/' \
    --regex='/[ \t]*\(ATTRIBUTE\|ENTITY\|FUNCTION\|PACKAGE\
    \( BODY\)?\|PROCEDURE\|PROCESS\|TYPE\)[ \t]+\([^ \t(]+\)/\3/'
  ```

22.16.4 Selecting a Tags Table

Emacs has at any time one *selected* tags table, and all the commands for working with tags tables use the selected one. To select a tags table, type `M-x visit-tags-table`, which reads the tags table file name as an argument. The name '`TAGS`' in the default directory is used as the default file name.

All this command does is store the file name in the variable `tags-file-name`. Emacs does not actually read in the tags table contents until you try to use them. Setting this variable yourself is just as good as using `visit-tags-table`. The variable's initial value is `nil`; that value tells all the commands for working with tags tables that they must ask for a tags table file name to use.

Using `visit-tags-table` when a tags table is already loaded gives you a choice: you can add the new tags table to the current list of tags tables, or start a new list. The tags commands use all the tags tables in the current list. If you start a new list, the new tags table is used *instead* of others. If you add the new table to the current list, it is used *as well as* the others. When the tags commands scan the list of tags tables, they don't always start at the beginning of the list; they start with the first tags table (if any) that describes the current file, proceed from there to the end of the list, and then scan from the beginning of the list until they have covered all the tables in the list.

You can specify a precise list of tags tables by setting the variable `tags-table-list` to a list of strings, like this:

```
(setq tags-table-list
      '("~/emacs" "/usr/local/lib/emacs/src"))
```

This tells the tags commands to look at the '`TAGS`' files in your '`~/emacs`' 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, as explained above.

Do not set both `tags-file-name` and `tags-table-list`.

22.16.5 Finding a Tag

The most important thing that a tags table enables you to do is to find the definition of a specific tag.

`M-.` *tag* ⟨RET⟩
: Find first definition of *tag* (`find-tag`).

`C-u M-.`
: Find next alternate definition of last tag specified.

`C-u - M-.`
: Go back to previous tag found.

`C-M-.` *pattern* ⟨RET⟩
: Find a tag whose name matches *pattern* (`find-tag-regexp`).

`C-u C-M-.` Find the next tag whose name matches the last pattern used.

`C-x 4 .` *tag* ⟨RET⟩
: Find first definition of *tag*, but display it in another window (`find-tag-other-window`).

`C-x 5 .` *tag* ⟨RET⟩
: Find first definition of *tag*, and create a new frame to select the buffer (`find-tag-other-frame`).

`M-*` Pop back to where you previously invoked `M-.` and friends.

`M-.` (`find-tag`) is the command to find the definition of a specified tag. It searches through the tags table for that tag, as a string, and then uses the tags table info to determine the file that the definition is in and the approximate character position in the file of the definition. Then `find-tag` visits that file, moves point to the approximate character position, and searches ever-increasing distances away to find the tag definition.

If an empty argument is given (just type ⟨RET⟩), the sexp in the buffer before or around point is used as the *tag* argument. See Section 22.2 [Lists], page 274, for info on sexps.

You don't need to give `M-.` the full name of the tag; a part will do. This is because `M-.` finds tags in the table which contain *tag* as a substring. However, it prefers an exact match to a substring match. To find other tags that match the same substring, give `find-tag` a numeric argument, as in `C-u M-.`; this does not read a tag name, but continues searching the tags table's text for another tag containing the same substring last used. If you have a real ⟨META⟩ key, `M-0 M-.` is an easier alternative to `C-u M-.`.

Like most commands that can switch buffers, `find-tag` has a variant that displays the new buffer in another window, and one that makes a new frame for it. The former is `C-x 4 .`, which invokes the command `find-tag-other-window`. The latter is `C-x 5 .`, which invokes `find-tag-other-frame`.

To move back to places you've found tags recently, use `C-u - M-.`; more generally, `M-.` with a negative numeric argument. This command can take you to another buffer. `C-x 4 .` with a negative argument finds the previous tag location in another window.

As well as going back to places you've found tags recently, you can go back to places *from where* you found them. Use `M-*`, which invokes the command `pop-tag-mark`, for this. Typically you would find and study the definition of something with `M-.` and then return to where you were with `M-*`.

Both `C-u - M-.` and `M-*` allow you to retrace your steps to a depth determined by the variable `find-tag-marker-ring-length`.

The command `C-M-.` (`find-tag-regexp`) visits the tags that match a specified regular expression. It is just like `M-.` except that it does regexp matching instead of substring matching.

22.16.6 Searching and Replacing with Tags Tables

The commands in this section visit and search all the files listed in the selected tags table, one by one. For these commands, the tags table serves only to specify a sequence of files to search.

`M-x tags-search` ⟨RET⟩ *regexp* ⟨RET⟩
: Search for *regexp* through the files in the selected tags table.

`M-x tags-query-replace` ⟨RET⟩ *regexp* ⟨RET⟩ *replacement* ⟨RET⟩
: Perform a `query-replace-regexp` on each file in the selected tags table.

`M-,`
: Restart one of the commands above, from the current location of point (`tags-loop-continue`).

`M-x tags-search` reads a regexp using the minibuffer, then searches for matches in all the files in the selected tags table, one file at a time. It displays the name of the file being searched so you can follow its progress. As soon as it finds an occurrence, `tags-search` returns.

Having found one match, you probably want to find all the rest. To find one more match, type `M-,` (`tags-loop-continue`) to resume the `tags-search`. This searches the rest of the current buffer, followed by the remaining files of the tags table.

`M-x tags-query-replace` performs a single `query-replace-regexp` through all the files in the tags table. It reads a regexp to search for and a string to replace with, just like ordinary `M-x query-replace-regexp`. It searches much like `M-x tags-search`, but repeatedly, processing matches according to your input. See Section 12.7 [Replace], page 132, for more information on query replace.

You can control the case-sensitivity of tags search commands by customizing the value of the variable `tags-case-fold-search`. The default is to use the same setting as the value of `case-fold-search` (see Section 12.6 [Search Case], page 131).

It is possible to get through all the files in the tags table with a single invocation of `M-x tags-query-replace`. But often it is useful to exit temporarily, which you can do with any input event that has no special query replace meaning. You can resume the query replace subsequently by typing `M-,`; this command resumes the last tags search or replace command that you did.

The commands in this section carry out much broader searches than the `find-tag` family. The `find-tag` commands search only for definitions of tags that match your substring or regexp. The commands `tags-search` and `tags-query-replace` find every occurrence of the regexp, as ordinary search commands and replace commands do in the current buffer.

These commands create buffers only temporarily for the files that they have to search (those which are not already visited in Emacs buffers). Buffers in which no match is found are quickly killed; the others continue to exist.

It may have struck you that `tags-search` is a lot like `grep`. You can also run `grep` itself as an inferior of Emacs and have Emacs show you the matching lines one by one. This works much like running a compilation; finding the source locations of the `grep` matches works like finding the compilation errors. See Section 23.1 [Compilation], page 331.

22.16.7 Tags Table Inquiries

`M-x list-tags` ⟨RET⟩ *file* ⟨RET⟩
: Display a list of the tags defined in the program file *file*.

`M-x tags-apropos` ⟨RET⟩ *regexp* ⟨RET⟩
: Display a list of all tags matching *regexp*.

`M-x list-tags` reads the name of one of the files described by the selected tags table, and displays a list of all the tags defined in that file. The "file name" argument is really just a string to compare against the file names recorded in the tags table; it is read as a string rather than as a file name. Therefore, completion and defaulting are not available, and you must enter the file name the same way it appears in the tags table. Do not include a directory as part of the file name unless the file name recorded in the tags table includes a directory.

`M-x tags-apropos` is like `apropos` for tags (see Section 7.3 [Apropos], page 71). It finds all the tags in the selected tags table whose entries match *regexp*, and displays them. If the variable `tags-apropos-verbose` is non-`nil`, it displays the names of the tags files together with the tag names.

You can customize the appearance of the output with the face `tags-tag-face`. You can display additional output with `M-x tags-apropos` by customizing the variable `tags-apropos-additional-actions`—see its documentation for details.

You can also use the collection of tag names to complete a symbol name in the buffer. See Section 22.9 [Symbol Completion], page 295.

22.17 Imenu

The Imenu facility is another way to find definitions or sections in a file. It is similar in spirit to Tags, but operates on a single buffer only, and works entirely within Emacs with no need for a separate tags table.

If you type `M-x imenu`, it reads the name of a section or definition in the current buffer, then goes to that section or definition. You can use

completion to specify the name, and a complete list of possible names is always displayed.

Alternatively you can bind the command `imenu` to a mouse click. Then it displays mouse menus for you to select the section or definition you want. You can also add the buffer's index to the menu bar by calling `imenu-add-menu-bar-index`. If you want to have this menu bar item available for all buffers in a certain major mode, you can do this by adding `imenu-add-menu-bar-index` to its mode hook. But then you will have to wait for the buffer to be searched for sections and definitions, each time you visit a file which uses that mode.

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

You can customize the way the menus are sorted via the variable `imenu-sort-function`. By default names are ordered as they occur in the buffer; alphabetic sorting is provided as an alternative.

Imenu provides the information to guide Which Function mode (see Section 22.10 [Which Function], page 295). The Speedbar can also use it (see Section 17.9 [Speedbar], page 211).

22.18 Merging Files with Emerge

It's not unusual for programmers to get their signals crossed and modify the same program in two different directions. To recover from this confusion, you need to merge the two versions. Emerge makes this easier. See also Section 14.9 [Comparing Files], page 181, for commands to compare in a more manual fashion, and section "" in *The Ediff Manual*.

22.18.1 Overview of Emerge

To start Emerge, run one of these four commands:

`M-x emerge-files`
: Merge two specified files.

`M-x emerge-files-with-ancestor`
: Merge two specified files, with reference to a common ancestor.

`M-x emerge-buffers`
: Merge two buffers.

`M-x emerge-buffers-with-ancestor`
: Merge two buffers with reference to a common ancestor in a third buffer.

The Emerge commands compare two files or buffers, and display the comparison in three buffers: one for each input text (the *A buffer* and the *B buffer*), and one (the *merge buffer*) where merging takes place. The merge buffer shows the full merged text, not just the differences. Wherever the two input texts differ, you can choose which one of them to include in the merge buffer.

The Emerge commands that take input from existing buffers use only the accessible portions of those buffers, if they are narrowed (see Section 30.9 [Narrowing], page 443).

If a common ancestor version is available, from which the two texts to be merged were both derived, Emerge can use it to guess which alternative is right. Wherever one current version agrees with the ancestor, Emerge presumes that the other current version is a deliberate change which should be kept in the merged version. Use the '`with-ancestor`' commands if you want to specify a common ancestor text. These commands read three file or buffer names—variant A, variant B, and the common ancestor.

After the comparison is done and the buffers are prepared, the interactive merging starts. You control the merging by typing special *merge commands* in the merge buffer. The merge buffer shows you a full merged text, not just differences. For each run of differences between the input texts, you can choose which one of them to keep, or edit them both together.

The merge buffer uses a special major mode, Emerge mode, with commands for making these choices. But you can also edit the buffer with ordinary Emacs commands.

At any given time, the attention of Emerge is focused on one particular difference, called the *selected* difference. This difference is marked off in the three buffers like this:

```
vvvvvvvvvvvvvvvvvvv
text that differs
^^^^^^^^^^^^^^^^^^^
```

Emerge numbers all the differences sequentially and the mode line always shows the number of the selected difference.

Normally, the merge buffer starts out with the A version of the text. But when the A version of a difference agrees with the common ancestor, then the B version is initially preferred for that difference.

Emerge leaves the merged text in the merge buffer when you exit. At that point, you can save it in a file with `C-x C-w`. If you give a numeric argument to `emerge-files` or `emerge-files-with-ancestor`, it reads the name of the output file using the minibuffer. (This is the last file name those commands read.) Then exiting from Emerge saves the merged text in the output file.

Normally, Emerge commands save the output buffer in its file when you exit. If you abort Emerge with `C-]`, the Emerge command does not save the output buffer, but you can save it yourself if you wish.

22.18.2 Submodes of Emerge

You can choose between two modes for giving merge commands: Fast mode and Edit mode. In Fast mode, basic merge commands are single characters, but ordinary Emacs commands are disabled. This is convenient if you use only merge commands. In Edit mode, all merge commands start with the prefix key `C-c C-c`, and the normal Emacs commands are also available. This allows editing the merge buffer, but slows down Emerge operations.

Use `e` to switch to Edit mode, and `C-c C-c f` to switch to Fast mode. The mode line indicates Edit and Fast modes with '`E`' and '`F`'.

Emerge has two additional submodes that affect how particular merge commands work: Auto Advance mode and Skip Prefers mode.

If Auto Advance mode is in effect, the `a` and `b` commands advance to the next difference. This lets you go through the merge faster as long as you simply choose one of the alternatives from the input. The mode line indicates Auto Advance mode with '`A`'.

If Skip Prefers mode is in effect, the `n` and `p` commands skip over differences in states prefer-A and prefer-B (see Section 22.18.3 [State of Difference], page 313). Thus you see only differences for which neither version is presumed "correct." The mode line indicates Skip Prefers mode with '`S`'.

Use the command `s a` (`emerge-auto-advance-mode`) to set or clear Auto Advance mode. Use `s s` (`emerge-skip-prefers-mode`) to set or clear Skip Prefers mode. These commands turn on the mode with a positive argument, turns it off with a negative or zero argument, and toggle the mode with no argument.

22.18.3 State of a Difference

In the merge buffer, a difference is marked with lines of '`v`' and '`^`' characters. Each difference has one of these seven states:

A
: The difference is showing the A version. The `a` command always produces this state; the mode line indicates it with '`A`'.

B
: The difference is showing the B version. The `b` command always produces this state; the mode line indicates it with '`B`'.

default-A
default-B
: The difference is showing the A or the B state by default, because you haven't made a choice. All differences start in the default-

A state (and thus the merge buffer is a copy of the A buffer), except those for which one alternative is "preferred" (see below).

When you select a difference, its state changes from default-A or default-B to plain A or B. Thus, the selected difference never has state default-A or default-B, and these states are never displayed in the mode line.

The command `d a` chooses default-A as the default state, and `d b` chooses default-B. This chosen default applies to all differences which you haven't ever selected and for which no alternative is preferred. If you are moving through the merge sequentially, the differences you haven't selected are those following the selected one. Thus, while moving sequentially, you can effectively make the A version the default for some sections of the merge buffer and the B version the default for others by using `d a` and `d b` between sections.

prefer-A
prefer-B
: The difference is showing the A or B state because it is *preferred*. This means that you haven't made an explicit choice, but one alternative seems likely to be right because the other alternative agrees with the common ancestor. Thus, where the A buffer agrees with the common ancestor, the B version is preferred, because chances are it is the one that was actually changed.

  These two states are displayed in the mode line as '`A*`' and '`B*`'.

combined
: The difference is showing a combination of the A and B states, as a result of the `x c` or `x C` commands.

  Once a difference is in this state, the `a` and `b` commands don't do anything to it unless you give them a numeric argument.

  The mode line displays this state as '`comb`'.

22.18.4 Merge Commands

Here are the Merge commands for Fast mode; in Edit mode, precede them with `C-c C-c`:

`p`
: Select the previous difference.

`n`
: Select the next difference.

`a`
: Choose the A version of this difference.

`b`
: Choose the B version of this difference.

`C-u` *n* `j`
: Select difference number *n*.

`.`
: Select the difference containing point. You can use this command in the merge buffer or in the A or B buffer.

`q`	Quit—finish the merge.
`C-]`	Abort—exit merging and do not save the output.
`f`	Go into Fast mode. (In Edit mode, this is actually `C-c C-c f`.)
`e`	Go into Edit mode.
`l`	Recenter (like `C-l`) all three windows.
`-`	Specify part of a prefix numeric argument.
digit	Also specify part of a prefix numeric argument.
`d a`	Choose the A version as the default from here down in the merge buffer.
`d b`	Choose the B version as the default from here down in the merge buffer.
`c a`	Copy the A version of this difference into the kill ring.
`c b`	Copy the B version of this difference into the kill ring.
`i a`	Insert the A version of this difference at point.
`i b`	Insert the B version of this difference at point.
`m`	Put point and mark around the difference.
`^`	Scroll all three windows down (like `M-v`).
`v`	Scroll all three windows up (like `C-v`).
`<`	Scroll all three windows left (like `C-x <`).
`>`	Scroll all three windows right (like `C-x >`).
`|`	Reset horizontal scroll on all three windows.
`x 1`	Shrink the merge window to one line. (Use `C-u l` to restore it to full size.)
`x c`	Combine the two versions of this difference (see Section 22.18.6 [Combining in Emerge], page 316).
`x f`	Show the names of the files/buffers Emerge is operating on, in a Help window. (Use `C-u l` to restore windows.)
`x j`	Join this difference with the following one. (`C-u x j` joins this difference with the previous one.)
`x s`	Split this difference into two differences. Before you use this command, position point in each of the three buffers at the place where you want to split the difference.
`x t`	Trim identical lines off the top and bottom of the difference. Such lines occur when the A and B versions are identical but differ from the ancestor version.

22.18.5 Exiting Emerge

The `q` command (`emerge-quit`) finishes the merge, storing the results into the output file if you specified one. It restores the A and B buffers to their proper contents, or kills them if they were created by Emerge and you haven't changed them. It also disables the Emerge commands in the merge buffer, since executing them later could damage the contents of the various buffers.

`C-]` aborts the merge. This means exiting without writing the output file. If you didn't specify an output file, then there is no real difference between aborting and finishing the merge.

If the Emerge command was called from another Lisp program, then its return value is `t` for successful completion, or `nil` if you abort.

22.18.6 Combining the Two Versions

Sometimes you want to keep *both* alternatives for a particular difference. To do this, use `x c`, which edits the merge buffer like this:

```
#ifdef NEW
version from A buffer
#else /* not NEW */
version from B buffer
#endif /* not NEW */
```

While this example shows C preprocessor conditionals delimiting the two alternative versions, you can specify the strings to use by setting the variable `emerge-combine-versions-template` to a string of your choice. In the string, '`%a`' says where to put version A, and '`%b`' says where to put version B. The default setting, which produces the results shown above, looks like this:

```
"#ifdef NEW\n%a#else /* not NEW */\n%b#endif /* not NEW */\n"
```

22.18.7 Fine Points of Emerge

During the merge, you mustn't try to edit the A and B buffers yourself. Emerge modifies them temporarily, but ultimately puts them back the way they were.

You can have any number of merges going at once—just don't use any one buffer as input to more than one merge at once, since the temporary changes made in these buffers would get in each other's way.

Starting Emerge can take a long time because it needs to compare the files fully. Emacs can't do anything else until `diff` finishes. Perhaps in the

future someone will change Emerge to do the comparison in the background when the input files are large—then you could keep on doing other things with Emacs until Emerge is ready to accept commands.

After setting up the merge, Emerge runs the hook `emerge-startup-hook` (see Section 31.2.3 [Hooks], page 465).

22.19 C and Related Modes

This section gives a brief description of the special features available in C, C++, Objective-C, Java, CORBA IDL, and Pike modes. (These are called "C mode and related modes.") See section "ccmode" in *CC Mode*, for a more extensive description of these modes and their special features.

22.19.1 C Mode Motion Commands

This section describes commands for moving point, in C mode and related modes.

`C-c C-u` Move point back to the containing preprocessor conditional, leaving the mark behind. A prefix argument acts as a repeat count. With a negative argument, move point forward to the end of the containing preprocessor conditional. When going backwards, `#elif` is treated like `#else` followed by `#if`. When going forwards, `#elif` is ignored.

`C-c C-p` Move point back over a preprocessor conditional, leaving the mark behind. A prefix argument acts as a repeat count. With a negative argument, move forward.

`C-c C-n` Move point forward across a preprocessor conditional, leaving the mark behind. A prefix argument acts as a repeat count. With a negative argument, move backward.

`M-a` Move point to the beginning of the innermost C statement (`c-beginning-of-statement`). If point is already at the beginning of a statement, move to the beginning of the preceding statement. With prefix argument n, move back $n - 1$ statements.

If point is within a string or comment, or next to a comment (only whitespace between them), this command moves by sentences instead of statements.

When called from a program, this function takes three optional arguments: the numeric prefix argument, a buffer position limit (don't move back before that place), and a flag that controls whether to do sentence motion when inside of a comment.

`M-e` Move point to the end of the innermost C statement; like `M-a` except that it moves in the other direction (`c-end-of-statement`).

`M-x c-backward-into-nomenclature`
Move point backward to beginning of a C++ nomenclature section or word. With prefix argument *n*, move *n* times. If *n* is negative, move forward. C++ nomenclature means a symbol name in the style of NamingSymbolsWithMixedCaseAndNoUnderlines; each capital letter begins a section or word.

In the GNU project, we recommend using underscores to separate words within an identifier in C or C++, rather than using case distinctions.

`M-x c-forward-into-nomenclature`
Move point forward to end of a C++ nomenclature section or word. With prefix argument *n*, move *n* times.

22.19.2 Electric C Characters

In C mode and related modes, certain printing characters are "electric"—in addition to inserting themselves, they also reindent the current line and may insert newlines. This feature is controlled by the variable `c-auto-newline`. The "electric" characters are `{`, `}`, `:`, `#`, `;`, `,`, `<`, `>`, `/`, `*`, `(`, and `)`.

Electric characters insert newlines only when the *auto-newline* feature is enabled (indicated by '`/a`' in the mode line after the mode name). This feature is controlled by the variable `c-auto-newline`. You can turn this feature on or off with the command `C-c C-a`:

`C-c C-a` Toggle the auto-newline feature (`c-toggle-auto-state`). With a prefix argument, this command turns the auto-newline feature on if the argument is positive, and off if it is negative.

The colon character is electric because that is appropriate for a single colon. But when you want to insert a double colon in C++, the electric behavior of colon is inconvenient. You can insert a double colon with no reindentation or newlines by typing `C-c :`:

`C-c :` Insert a double colon scope operator at point, without reindenting the line or adding any newlines (`c-scope-operator`).

The electric `#` key reindents the line if it appears to be the beginning of a preprocessor directive. This happens when the value of `c-electric-pound-behavior` is `(alignleft)`. You can turn this feature off by setting `c-electric-pound-behavior` to `nil`.

The variable `c-hanging-braces-alist` controls the insertion of newlines before and after inserted braces. It is an association list with elements of the

following form: (*syntactic-symbol* . *nl-list*). Most of the syntactic symbols that appear in `c-offsets-alist` are meaningful here as well.

The list *nl-list* may contain either of the symbols `before` or `after`, or both; or it may be `nil`. When a brace is inserted, the syntactic context it defines is looked up in `c-hanging-braces-alist`; if it is found, the *nl-list* is used to determine where newlines are inserted: either before the brace, after, or both. If not found, the default is to insert a newline both before and after braces.

The variable `c-hanging-colons-alist` controls the insertion of newlines before and after inserted colons. It is an association list with elements of the following form: (*syntactic-symbol* . *nl-list*). The list *nl-list* may contain either of the symbols `before` or `after`, or both; or it may be `nil`.

When a colon is inserted, the syntactic symbol it defines is looked up in this list, and if found, the *nl-list* is used to determine where newlines are inserted: either before the brace, after, or both. If the syntactic symbol is not found in this list, no newlines are inserted.

Electric characters can also delete newlines automatically when the auto-newline feature is enabled. This feature makes auto-newline more acceptable, by deleting the newlines in the most common cases where you do not want them. Emacs can recognize several cases in which deleting a newline might be desirable; by setting the variable `c-cleanup-list`, you can specify *which* of these cases that should happen. The variable's value is a list of symbols, each describing one case for possible deletion of a newline. Here are the meaningful symbols, and their meanings:

`brace-catch-brace`
: Clean up '`}` `catch` `(`*condition*`)` `{`' constructs by placing the entire construct on a single line. The clean-up occurs when you type the '`{`', if there is nothing between the braces aside from `catch` and *condition*.

`brace-else-brace`
: Clean up '`}` `else` `{`' constructs by placing the entire construct on a single line. The clean-up occurs when you type the '`{`' after the `else`, but only if there is nothing but white space between the braces and the `else`.

`brace-elseif-brace`
: Clean up '`}` `else if (...)` `{`' constructs by placing the entire construct on a single line. The clean-up occurs when you type the '`{`', if there is nothing but white space between the '`}`' and '`{`' aside from the keywords and the `if`-condition.

`empty-defun-braces`
: Clean up empty defun braces by placing the braces on the same line. Clean-up occurs when you type the closing brace.

`defun-close-semi`
: Clean up the semicolon after a `struct` or similar type declaration, by placing the semicolon on the same line as the closing brace. Clean-up occurs when you type the semicolon.

`list-close-comma`
: Clean up commas following braces in array and aggregate initializers. Clean-up occurs when you type the comma.

`scope-operator`
: Clean up double colons which may designate a C++ scope operator, by placing the colons together. Clean-up occurs when you type the second colon, but only when the two colons are separated by nothing but whitespace.

22.19.3 Hungry Delete Feature in C

When the *hungry-delete* feature is enabled (indicated by '`/h`' or '`/ah`' in the mode line after the mode name), a single ⟨DEL⟩ command deletes all preceding whitespace, not just one space. To turn this feature on or off, use `C-c C-d`:

`C-c C-d` Toggle the hungry-delete feature (`c-toggle-hungry-state`). With a prefix argument, this command turns the hungry-delete feature on if the argument is positive, and off if it is negative.

`C-c C-t` Toggle the auto-newline and hungry-delete features, both at once (`c-toggle-auto-hungry-state`).

The variable `c-hungry-delete-key` controls whether the hungry-delete feature is enabled.

22.19.4 Other Commands for C Mode

`C-M-h` Put mark at the end of a function definition, and put point at the beginning (`c-mark-function`).

`M-q` Fill a paragraph, handling C and C++ comments (`c-fill-paragraph`). If any part of the current line is a comment or within a comment, this command fills the comment or the paragraph of it that point is in, preserving the comment indentation and comment delimiters.

`C-c C-e` Run the C preprocessor on the text in the region, and show the result, which includes the expansion of all the macro calls (`c-macro-expand`). The buffer text before the region is also

included in preprocessing, for the sake of macros defined there, but the output from this part isn't shown.

When you are debugging C code that uses macros, sometimes it is hard to figure out precisely how the macros expand. With this command, you don't have to figure it out; you can see the expansions.

`C-c C-\` Insert or align '\' characters at the ends of the lines of the region (`c-backslash-region`). This is useful after writing or editing a C macro definition.

If a line already ends in '\', this command adjusts the amount of whitespace before it. Otherwise, it inserts a new '\'. However, the last line in the region is treated specially; no '\' is inserted on that line, and any '\' there is deleted.

`M-x cpp-highlight-buffer`
Highlight parts of the text according to its preprocessor conditionals. This command displays another buffer named '`*CPP Edit*`', which serves as a graphic menu for selecting how to display particular kinds of conditionals and their contents. After changing various settings, click on '`[A]pply these settings`' (or go to that buffer and type `a`) to rehighlight the C mode buffer accordingly.

`C-c C-s` Display the syntactic information about the current source line (`c-show-syntactic-information`). This is the information that directs how the line is indented.

`M-x cwarn-mode`
`M-x global-cwarn-mode`
CWarn minor mode highlights certain suspicious C and C++ constructions:

- Assignments inside expressions.
- Semicolon following immediately after '`if`', '`for`', and '`while`' (except after a '`do ... while`' statement);
- C++ functions with reference parameters.

You can enable the mode for one buffer with the command `M-x cwarn-mode`, or for all suitable buffers with the command `M-x global-cwarn-mode` or by customizing the variable `global-cwarn-mode`. You must also enable Font Lock mode to make it work.

`M-x hide-ifdef-mode`
Hide-ifdef minor mode hides selected code within '`#if`' and '`#ifdef`' preprocessor blocks. See the documentation string of `hide-ifdef-mode` for more information.

`M-x ff-find-related-file`
: Find a file "related" in a special way to the file visited by the current buffer. Typically this will be the header file corresponding to a C/C++ source file, or vice versa. The variable `ff-related-file-alist` specifies how to compute related file names.

22.19.5 Comments in C Modes

C mode and related modes use a number of variables for controlling comment format.

`c-comment-only-line-offset`
: Extra offset for line which contains only the start of a comment. It can be either an integer or a cons cell of the form (*non-anchored-offset* . *anchored-offset*), where *non-anchored-offset* is the amount of offset given to non-column-zero anchored comment-only lines, and *anchored-offset* is the amount of offset to give column-zero anchored comment-only lines. Just an integer as value is equivalent to (*val* . 0).

`c-comment-start-regexp`
: This buffer-local variable specifies how to recognize the start of a comment.

`c-hanging-comment-ender-p`
: If this variable is `nil`, `c-fill-paragraph` leaves the comment terminator of a block comment on a line by itself. The default value is `t`, which puts the comment-end delimiter '`*/`' at the end of the last line of the comment text.

`c-hanging-comment-starter-p`
: If this variable is `nil`, `c-fill-paragraph` leaves the starting delimiter of a block comment on a line by itself. The default value is `t`, which puts the comment-start delimiter '`/*`' at the beginning of the first line of the comment text.

22.20 Fortran Mode

Fortran mode provides special motion commands for Fortran statements and subprograms, and indentation commands that understand Fortran conventions of nesting, line numbers and continuation statements. Fortran mode has its own Auto Fill mode that breaks long lines into proper Fortran continuation lines.

Special commands for comments are provided because Fortran comments are unlike those of other languages. Built-in abbrevs optionally save typing when you insert Fortran keywords.

Use `M-x fortran-mode` to switch to this major mode. This command runs the hook `fortran-mode-hook` (see Section 31.2.3 [Hooks], page 465).

Fortan mode is meant for editing Fortran77 "fixed format" source code. For editing the modern Fortran90 "free format" source code, use F90 mode (`f90-mode`). Emacs normally uses Fortran mode for files with extension '`.f`', '`.F`' or '`.for`', and F90 mode for the extension '`.f90`'. GNU Fortran supports both kinds of format.

22.20.1 Motion Commands

In addition to the normal commands for moving by and operating on "defuns" (Fortran subprograms—functions and subroutines), Fortran mode provides special commands to move by statements.

`C-c C-n` Move to beginning of current or next statement (`fortran-next-statement`).

`C-c C-p` Move to beginning of current or previous statement (`fortran-previous-statement`).

22.20.2 Fortran Indentation

Special commands and features are needed for indenting Fortran code in order to make sure various syntactic entities (line numbers, comment line indicators and continuation line flags) appear in the columns that are required for standard Fortran.

22.20.2.1 Fortran Indentation and Filling Commands

`C-M-j` Break the current line and set up a continuation line (`fortran-split-line`).

`M-^` Join this line to the previous line (`fortran-join-line`).

`C-M-q` Indent all the lines of the subprogram point is in (`fortran-indent-subprogram`).

`M-q` Fill a comment block or statement.

The key `C-M-q` runs `fortran-indent-subprogram`, a command to reindent all the lines of the Fortran subprogram (function or subroutine) containing point.

The key `C-M-j` runs `fortran-split-line`, which splits a line in the appropriate fashion for Fortran. In a non-comment line, the second half becomes a continuation line and is indented accordingly. In a comment line, both halves become separate comment lines.

`M-^` or `C-c C-d` runs the command `fortran-join-line`, which joins a continuation line back to the previous line, roughly as the inverse of `fortran-split-line`. The point must be on a continuation line when this command is invoked.

`M-q` in Fortran mode fills the comment block or statement that point is in. This removes any excess statement continuations.

22.20.2.2 Continuation Lines

Most modern Fortran compilers allow two ways of writing continuation lines. If the first non-space character on a line is in column 5, then that line is a continuation of the previous line. We call this *fixed format*. (In GNU Emacs we always count columns from 0.) The variable `fortran-continuation-string` specifies what character to put on column 5. A line that starts with a tab character followed by any digit except '`0`' is also a continuation line. We call this style of continuation *tab format*.

Fortran mode can make either style of continuation line, but you must specify which one you prefer. The value of the variable `indent-tabs-mode` controls the choice: `nil` for fixed format, and non-`nil` for tab format. You can tell which style is presently in effect by the presence or absence of the string '`Tab`' in the mode line.

If the text on a line starts with the conventional Fortran continuation marker '`$`', or if it begins with any non-whitespace character in column 5, Fortran mode treats it as a continuation line. When you indent a continuation line with ⟨TAB⟩, it converts the line to the current continuation style. When you split a Fortran statement with `C-M-j`, the continuation marker on the newline is created according to the continuation style.

The setting of continuation style affects several other aspects of editing in Fortran mode. In fixed format mode, the minimum column number for the body of a statement is 6. Lines inside of Fortran blocks that are indented to larger column numbers always use only the space character for whitespace. In tab format mode, the minimum column number for the statement body is 8, and the whitespace before column 8 must always consist of one tab character.

When you enter Fortran mode for an existing file, it tries to deduce the proper continuation style automatically from the file contents. The first line that begins with either a tab character or six spaces determines the choice. The variable `fortran-analyze-depth` specifies how many lines to consider (at the beginning of the file); if none of those lines indicates a style, then the variable `fortran-tab-mode-default` specifies the style. If it is `nil`, that specifies fixed format, and non-`nil` specifies tab format.

22.20.2.3 Line Numbers

If a number is the first non-whitespace in the line, Fortran indentation assumes it is a line number and moves it to columns 0 through 4. (Columns always count from 0 in GNU Emacs.)

Line numbers of four digits or less are normally indented one space. The variable `fortran-line-number-indent` controls this; it specifies the maximum indentation a line number can have. Line numbers are indented to right-justify them to end in column 4 unless that would require more than this maximum indentation. The default value of the variable is 1.

Simply inserting a line number is enough to indent it according to these rules. As each digit is inserted, the indentation is recomputed. To turn off this feature, set the variable `fortran-electric-line-number` to `nil`. Then inserting line numbers is like inserting anything else.

22.20.2.4 Syntactic Conventions

Fortran mode assumes that you follow certain conventions that simplify the task of understanding a Fortran program well enough to indent it properly:

- Two nested '`do`' loops never share a '`continue`' statement.
- Fortran keywords such as '`if`', '`else`', '`then`', '`do`' and others are written without embedded whitespace or line breaks.

  Fortran compilers generally ignore whitespace outside of string constants, but Fortran mode does not recognize these keywords if they are not contiguous. Constructs such as '`else if`' or '`end do`' are acceptable, but the second word should be on the same line as the first and not on a continuation line.

If you fail to follow these conventions, the indentation commands may indent some lines unaesthetically. However, a correct Fortran program retains its meaning when reindented even if the conventions are not followed.

22.20.2.5 Variables for Fortran Indentation

Several additional variables control how Fortran indentation works:

`fortran-do-indent`
: Extra indentation within each level of '`do`' statement (default 3).

`fortran-if-indent`
: Extra indentation within each level of '`if`' statement (default 3). This value is also used for extra indentation within each level of the Fortran 90 '`where`' statement.

`fortran-structure-indent`
: Extra indentation within each level of '`structure`', '`union`', or '`map`' statements (default 3).

`fortran-continuation-indent`
: Extra indentation for bodies of continuation lines (default 5).

`fortran-check-all-num-for-matching-do`
: If this is `nil`, indentation assumes that each '`do`' statement ends on a '`continue`' statement. Therefore, when computing indentation for a statement other than '`continue`', it can save time by not checking for a '`do`' statement ending there. If this is non-`nil`, indenting any numbered statement must check for a '`do`' that ends there. The default is `nil`.

`fortran-blink-matching-if`
: If this is `t`, indenting an '`endif`' statement moves the cursor momentarily to the matching '`if`' statement to show where it is. The default is `nil`.

`fortran-minimum-statement-indent-fixed`
: Minimum indentation for fortran statements when using fixed format continuation line style. Statement bodies are never indented less than this much. The default is 6.

`fortran-minimum-statement-indent-tab`
: Minimum indentation for fortran statements for tab format continuation line style. Statement bodies are never indented less than this much. The default is 8.

22.20.3 Fortran Comments

The usual Emacs comment commands assume that a comment can follow a line of code. In Fortran, the standard comment syntax requires an entire line to be just a comment. Therefore, Fortran mode replaces the standard Emacs comment commands and defines some new variables.

Fortran mode can also handle the Fortran90 comment syntax where comments start with '`!`' and can follow other text. Because only some Fortran77 compilers accept this syntax, Fortran mode will not insert such comments unless you have said in advance to do so. To do this, set the variable `comment-start` to '`"!"`' (see Section 31.2 [Variables], page 457).

`M-;`
: Align comment or insert new comment (`fortran-comment-indent`).

`C-x ;`
: Applies to nonstandard '`!`' comments only.

`C-c ;`
: Turn all lines of the region into comments, or (with argument) turn them back into real code (`fortran-comment-region`).

`M-;` in Fortran mode is redefined as the command `fortran-comment-indent`. Like the usual `M-;` command, this recognizes any kind of existing comment and aligns its text appropriately; if there is no existing comment, a comment is inserted and aligned. But inserting and aligning comments are not the same in Fortran mode as in other modes.

When a new comment must be inserted, if the current line is blank, a full-line comment is inserted. On a non-blank line, a nonstandard '`!`' comment is inserted if you have said you want to use them. Otherwise a full-line comment is inserted on a new line before the current line.

Nonstandard '`!`' comments are aligned like comments in other languages, but full-line comments are different. In a standard full-line comment, the comment delimiter itself must always appear in column zero. What can be aligned is the text within the comment. You can choose from three styles of alignment by setting the variable `fortran-comment-indent-style` to one of these values:

`fixed` Align the text at a fixed column, which is the sum of `fortran-comment-line-extra-indent` and the minimum statement indentation. This is the default.

The minimum statement indentation is `fortran-minimum-statement-indent-fixed` for fixed format continuation line style and `fortran-minimum-statement-indent-tab` for tab format style.

`relative` Align the text as if it were a line of code, but with an additional `fortran-comment-line-extra-indent` columns of indentation.

`nil` Don't move text in full-line comments automatically at all.

In addition, you can specify the character to be used to indent within full-line comments by setting the variable `fortran-comment-indent-char` to the single-character string you want to use.

Fortran mode introduces two variables `comment-line-start` and `comment-line-start-skip`, which play for full-line comments the same roles played by `comment-start` and `comment-start-skip` for ordinary text-following comments. Normally these are set properly by Fortran mode, so you do not need to change them.

The normal Emacs comment command `C-x ;` has not been redefined. If you use '`!`' comments, this command can be used with them. Otherwise it is useless in Fortran mode.

The command `C-c ;` (`fortran-comment-region`) turns all the lines of the region into comments by inserting the string '`C$$$`' at the front of each one. With a numeric argument, it turns the region back into live code by deleting '`C$$$`' from the front of each line in it. The string used for these comments can be controlled by setting the variable `fortran-comment-region`. Note that here we have an example of a command and a variable with the same

name; these two uses of the name never conflict because in Lisp and in Emacs it is always clear from the context which one is meant.

22.20.4 Fortran Auto Fill Mode

Fortran Auto Fill mode is a minor mode which automatically splits Fortran statements as you insert them when they become too wide. Splitting a statement involves making continuation lines using `fortran-continuation-string` (see Section 22.20.2.2 [ForIndent Cont], page 324). This splitting happens when you type ⟨SPC⟩, ⟨RET⟩, or ⟨TAB⟩, and also in the Fortran indentation commands.

`M-x fortran-auto-fill-mode` turns Fortran Auto Fill mode on if it was off, or off if it was on. This command works the same as `M-x auto-fill-mode` does for normal Auto Fill mode (see Section 21.5 [Filling], page 248). A positive numeric argument turns Fortran Auto Fill mode on, and a negative argument turns it off. You can see when Fortran Auto Fill mode is in effect by the presence of the word '`Fill`' in the mode line, inside the parentheses. Fortran Auto Fill mode is a minor mode, turned on or off for each buffer individually. See Section 31.1 [Minor Modes], page 455.

Fortran Auto Fill mode breaks lines at spaces or delimiters when the lines get longer than the desired width (the value of `fill-column`). The delimiters that Fortran Auto Fill mode may break at are '`,`', '`'`', '`+`', '`-`', '`/`', '`*`', '`=`', and '`)`'. The line break comes after the delimiter if the variable `fortran-break-before-delimiters` is `nil`. Otherwise (and by default), the break comes before the delimiter.

By default, Fortran Auto Fill mode is not enabled. If you want this feature turned on permanently, add a hook function to `fortran-mode-hook` to execute `(fortran-auto-fill-mode 1)`. See Section 31.2.3 [Hooks], page 465.

22.20.5 Checking Columns in Fortran

`C-c C-r` Display a "column ruler" momentarily above the current line (`fortran-column-ruler`).

`C-c C-w` Split the current window horizontally temporarily so that it is 72 columns wide (`fortran-window-create-momentarily`). This may help you avoid making lines longer than the 72-character limit that some Fortran compilers impose.

`C-u C-c C-w`
Split the current window horizontally so that it is 72 columns wide (`fortran-window-create`). You can then continue editing.

`M-x fortran-strip-sequence-nos`
: Delete all text in column 72 and beyond.

The command `C-c C-r` (`fortran-column-ruler`) shows a column ruler momentarily above the current line. The comment ruler is two lines of text that show you the locations of columns with special significance in Fortran programs. Square brackets show the limits of the columns for line numbers, and curly brackets show the limits of the columns for the statement body. Column numbers appear above them.

Note that the column numbers count from zero, as always in GNU Emacs. As a result, the numbers may be one less than those you are familiar with; but the positions they indicate in the line are standard for Fortran.

The text used to display the column ruler depends on the value of the variable `indent-tabs-mode`. If `indent-tabs-mode` is `nil`, then the value of the variable `fortran-column-ruler-fixed` is used as the column ruler. Otherwise, the variable `fortran-column-ruler-tab` is displayed. By changing these variables, you can change the column ruler display.

`C-c C-w` (`fortran-window-create-momentarily`) temporarily splits the current window horizontally, making a window 72 columns wide, so you can see which lines that is too long. Type a space to restore the normal width.

You can also split the window horizontally and continue editing with the split in place. To do this, use `C-u C-c C-w` (`M-x fortran-window-create`). By editing in this window you can immediately see when you make a line too wide to be correct Fortran.

The command `M-x fortran-strip-sequence-nos` deletes all text in column 72 and beyond, on all lines in the current buffer. This is the easiest way to get rid of old sequence numbers.

22.20.6 Fortran Keyword Abbrevs

Fortran mode provides many built-in abbrevs for common keywords and declarations. These are the same sort of abbrev that you can define yourself. To use them, you must turn on Abbrev mode. See Chapter 24 [Abbrevs], page 345.

The built-in abbrevs are unusual in one way: they all start with a semicolon. You cannot normally use semicolon in an abbrev, but Fortran mode makes this possible by changing the syntax of semicolon to "word constituent."

For example, one built-in Fortran abbrev is '`;c`' for '`continue`'. If you insert '`;c`' and then insert a punctuation character such as a space or a newline, the '`;c`' expands automatically to '`continue`', provided Abbrev mode is enabled.

Type '`;?`' or '`;C-h`' to display a list of all the built-in Fortran abbrevs and what they stand for.

22.21 Asm Mode

Asm mode is a major mode for editing files of assembler code. It defines these commands:

⟨TAB⟩ `tab-to-tab-stop`.

`C-j` Insert a newline and then indent using `tab-to-tab-stop`.

`:` Insert a colon and then remove the indentation from before the label preceding colon. Then do `tab-to-tab-stop`.

`;` Insert or align a comment.

The variable `asm-comment-char` specifies which character starts comments in assembler syntax.

23 Compiling and Testing Programs

The previous chapter discusses the Emacs commands that are useful for making changes in programs. This chapter deals with commands that assist in the larger process of developing and maintaining programs.

23.1 Running Compilations under Emacs

Emacs can run compilers for noninteractive languages such as C and Fortran as inferior processes, feeding the error log into an Emacs buffer. It can also parse the error messages and show you the source lines where compilation errors occurred.

`M-x compile`
: Run a compiler asynchronously under Emacs, with error messages to '`*compilation*`' buffer.

`M-x grep`
: Run `grep` asynchronously under Emacs, with matching lines listed in the buffer named '`*grep*`'.

`M-x grep-find`
: Run `grep` via `find`, with user-specified arguments, and collect output in the buffer named '`*grep*`'.

`M-x kill-compilation`
`M-x kill-grep`
: Kill the running compilation or `grep` subprocess.

To run `make` or another compilation command, do `M-x compile`. This command reads a shell command line using the minibuffer, and then executes the command in an inferior shell, putting output in the buffer named '`*compilation*`'. The current buffer's default directory is used as the working directory for the execution of the command; normally, therefore, the compilation happens in this directory.

When the shell command line is read, the minibuffer appears containing a default command line, which is the command you used the last time you did `M-x compile`. If you type just ⟨RET⟩, the same command line is used again. For the first `M-x compile`, the default is '`make -k`'. The default compilation command comes from the variable `compile-command`; if the appropriate compilation command for a file is something other than '`make -k`', it can be useful for the file to specify a local value for `compile-command` (see Section 31.2.5 [File Variables], page 468).

Starting a compilation displays the buffer '`*compilation*`' in another window but does not select it. The buffer's mode line tells you whether compilation is finished, with the word '`run`' or '`exit`' inside the parentheses.

You do not have to keep this buffer visible; compilation continues in any case. While a compilation is going on, the string '`Compiling`' appears in the mode lines of all windows. When this string disappears, the compilation is finished.

If you want to watch the compilation transcript as it appears, switch to the '`*compilation*`' buffer and move point to the end of the buffer. When point is at the end, new compilation output is inserted above point, which remains at the end. If point is not at the end of the buffer, it remains fixed while more compilation output is added at the end of the buffer.

If you set the variable `compilation-scroll-output` to a non-`nil` value, then the compilation buffer always scrolls to follow output as it comes in.

To kill the compilation process, do `M-x kill-compilation`. When the compiler process terminates, the mode line of the '`*compilation*`' buffer changes to say '`signal`' instead of '`run`'. Starting a new compilation also kills any running compilation, as only one can exist at any time. However, `M-x compile` asks for confirmation before actually killing a compilation that is running.

23.2 Searching with Grep under Emacs

Just as you can run a compiler from Emacs and then visit the lines where there were compilation errors, you can also run `grep` and then visit the lines on which matches were found. This works by treating the matches reported by `grep` as if they were "errors."

To do this, type `M-x grep`, then enter a command line that specifies how to run `grep`. Use the same arguments you would give `grep` when running it normally: a `grep`-style regexp (usually in single-quotes to quote the shell's special characters) followed by file names, which may use wildcards. The output from `grep` goes in the '`*grep*`' buffer. You can find the corresponding lines in the original files using `C-x ‘` and ⟨RET⟩, as with compilation errors.

If you specify a prefix argument for `M-x grep`, it figures out the tag (see Section 22.16 [Tags], page 301) around point, and puts that into the default `grep` command.

The command `M-x grep-find` is similar to `M-x grep`, but it supplies a different initial default for the command—one that runs both `find` and `grep`, so as to search every file in a directory tree. See also the `find-grep-dired` command, in Section 28.15 [Dired and Find], page 399.

23.3 Compilation Mode

The '*compilation*' buffer uses a special major mode, Compilation mode, whose main feature is to provide a convenient way to look at the source line where the error happened.

If you set the variable `compilation-scroll-output` to a non-`nil` value, then the compilation buffer always scrolls to follow output as it comes in.

`C-x ‘` Visit the locus of the next compiler error message or `grep` match.

⟨RET⟩ Visit the locus of the error message that point is on. This command is used in the compilation buffer.

`Mouse-2` Visit the locus of the error message that you click on.

You can visit the source for any particular error message by moving point in '*compilation*' to that error message and typing ⟨RET⟩ (`compile-goto-error`). Or click `Mouse-2` on the error message; you need not switch to the '*compilation*' buffer first.

To parse the compiler error messages sequentially, type `C-x ‘` (`next-error`). The character following the `C-x` is the backquote or "grave accent," not the single-quote. This command is available in all buffers, not just in '*compilation*'; it displays the next error message at the top of one window and source location of the error in another window.

The first time `C-x ‘` is used after the start of a compilation, it moves to the first error's location. Subsequent uses of `C-x ‘` advance down to subsequent errors. If you visit a specific error message with ⟨RET⟩ or `Mouse-2`, subsequent `C-x ‘` commands advance from there. When `C-x ‘` gets to the end of the buffer and finds no more error messages to visit, it fails and signals an Emacs error.

`C-u C-x ‘` starts scanning from the beginning of the compilation buffer. This is one way to process the same set of errors again.

Compilation mode also redefines the keys ⟨SPC⟩ and ⟨DEL⟩ to scroll by screenfuls, and `M-n` and `M-p` to move to the next or previous error message. You can also use `M-{` and `M-}` to move up or down to an error message for a different source file.

The features of Compilation mode are also available in a minor mode called Compilation Minor mode. This lets you parse error messages in any buffer, not just a normal compilation output buffer. Type `M-x compilation-minor-mode` to enable the minor mode. This defines the keys ⟨RET⟩ and `Mouse-2`, as in the Compilation major mode.

Compilation minor mode works in any buffer, as long as the contents are in a format that it understands. In an Rlogin buffer (see Section 30.2.10 [Remote Host], page 436), Compilation minor mode automatically accesses remote source files by FTP (see Section 14.1 [File Names], page 143).

23.4 Subshells for Compilation

Emacs uses a shell to run the compilation command, but specifies the option for a noninteractive shell. This means, in particular, that the shell should start with no prompt. If you find your usual shell prompt making an unsightly appearance in the '`*compilation*`' buffer, it means you have made a mistake in your shell's init file by setting the prompt unconditionally. (This init file's name may be '`.bashrc`', '`.profile`', '`.cshrc`', '`.shrc`', or various other things, depending on the shell you use.) The shell init file should set the prompt only if there already is a prompt. In csh, here is how to do it:

```
if ($?prompt) set prompt = ...
```

And here's how to do it in bash:

```
if [ "${PS1+set}" = set ]
then PS1=...
fi
```

There may well be other things that your shell's init file ought to do only for an interactive shell. You can use the same method to conditionalize them.

The MS-DOS "operating system" does not support asynchronous subprocesses; to work around this lack, `M-x compile` runs the compilation command synchronously on MS-DOS. As a consequence, you must wait until the command finishes before you can do anything else in Emacs. See Appendix E [MS-DOS], page 539.

23.5 Running Debuggers Under Emacs

The GUD (Grand Unified Debugger) library provides an interface to various symbolic debuggers from within Emacs. We recommend the debugger GDB, which is free software, but you can also run DBX, SDB or XDB if you have them. GUD can also serve as an interface to the Perl's debugging mode, the Python debugger PDB, and to JDB, the Java Debugger.

23.5.1 Starting GUD

There are several commands for starting a debugger, each corresponding to a particular debugger program.

`M-x gdb` ⟨RET⟩ *file* ⟨RET⟩

Run GDB as a subprocess of Emacs. This command creates a buffer for input and output to GDB, and switches to it. If a GDB buffer already exists, it just switches to that buffer.

`M-x dbx` ⟨RET⟩ *file* ⟨RET⟩
: Similar, but run DBX instead of GDB.

`M-x xdb` ⟨RET⟩ *file* ⟨RET⟩
: Similar, but run XDB instead of GDB. Use the variable `gud-xdb-directories` to specify directories to search for source files.

`M-x sdb` ⟨RET⟩ *file* ⟨RET⟩
: Similar, but run SDB instead of GDB.

  Some versions of SDB do not mention source file names in their messages. When you use them, you need to have a valid tags table (see Section 22.16 [Tags], page 301) in order for GUD to find functions in the source code. If you have not visited a tags table or the tags table doesn't list one of the functions, you get a message saying '`The sdb support requires a valid tags table to work`'. If this happens, generate a valid tags table in the working directory and try again.

`M-x perldb` ⟨RET⟩ *file* ⟨RET⟩
: Run the Perl interpreter in debug mode to debug *file*, a Perl program.

`M-x jdb` ⟨RET⟩ *file* ⟨RET⟩
: Run the Java debugger to debug *file*.

`M-x pdb` ⟨RET⟩ *file* ⟨RET⟩
: Run the Python debugger to debug *file*.

Each of these commands takes one argument: a command line to invoke the debugger. In the simplest case, specify just the name of the executable file you want to debug. You may also use options that the debugger supports. However, shell wildcards and variables are not allowed. GUD assumes that the first argument not starting with a '`-`' is the executable file name.

Emacs can only run one debugger process at a time.

23.5.2 Debugger Operation

When you run a debugger with GUD, the debugger uses an Emacs buffer for its ordinary input and output. This is called the GUD buffer. The debugger displays the source files of the program by visiting them in Emacs buffers. An arrow ('`=>`') in one of these buffers indicates the current execution line.[1] Moving point in this buffer does not move the arrow.

You can start editing these source files at any time in the buffers that were made to display them. The arrow is not part of the file's text; it appears

[1] Under a window system the arrow is displayed in the marginal area of the Emacs window.

only on the screen. If you do modify a source file, keep in mind that inserting or deleting lines will throw off the arrow's positioning; GUD has no way of figuring out which line corresponded before your changes to the line number in a debugger message. Also, you'll typically have to recompile and restart the program for your changes to be reflected in the debugger's tables.

If you wish, you can control your debugger process entirely through the debugger buffer, which uses a variant of Shell mode. All the usual commands for your debugger are available, and you can use the Shell mode history commands to repeat them. See Section 30.2.3 [Shell Mode], page 428.

23.5.3 Commands of GUD

The GUD interaction buffer uses a variant of Shell mode, so the commands of Shell mode are available (see Section 30.2.3 [Shell Mode], page 428). GUD mode also provides commands for setting and clearing breakpoints, for selecting stack frames, and for stepping through the program. These commands are available both in the GUD buffer and globally, but with different key bindings.

The breakpoint commands are usually used in source file buffers, because that is the way to specify where to set or clear the breakpoint. Here's the global command to set a breakpoint:

`C-x` ⟨SPC⟩ Set a breakpoint on the source line that point is on.

Here are the other special commands provided by GUD. The keys starting with `C-c` are available only in the GUD interaction buffer. The key bindings that start with `C-x C-a` are available in the GUD interaction buffer and also in source files.

`C-c C-l`
`C-x C-a C-l`
: Display in another window the last line referred to in the GUD buffer (that is, the line indicated in the last location message). This runs the command `gud-refresh`.

`C-c C-s`
`C-x C-a C-s`
: Execute a single line of code (`gud-step`). If the line contains a function call, execution stops after entering the called function.

`C-c C-n`
`C-x C-a C-n`
: Execute a single line of code, stepping across entire function calls at full speed (`gud-next`).

`C-c C-i`
`C-x C-a C-i`
: Execute a single machine instruction (`gud-stepi`).

`C-c C-r`
`C-x C-a C-r`
: Continue execution without specifying any stopping point. The program will run until it hits a breakpoint, terminates, or gets a signal that the debugger is checking for (`gud-cont`).

`C-c C-d`
`C-x C-a C-d`
: Delete the breakpoint(s) on the current source line, if any (`gud-remove`). If you use this command in the GUD interaction buffer, it applies to the line where the program last stopped.

`C-c C-t`
`C-x C-a C-t`
: Set a temporary breakpoint on the current source line, if any. If you use this command in the GUD interaction buffer, it applies to the line where the program last stopped.

The above commands are common to all supported debuggers. If you are using GDB or (some versions of) DBX, these additional commands are available:

`C-c <`
`C-x C-a <` Select the next enclosing stack frame (`gud-up`). This is equivalent to the '`up`' command.

`C-c >`
`C-x C-a >` Select the next inner stack frame (`gud-down`). This is equivalent to the '`down`' command.

If you are using GDB, these additional key bindings are available:

⟨TAB⟩
: With GDB, complete a symbol name (`gud-gdb-complete-command`). This key is available only in the GUD interaction buffer, and requires GDB versions 4.13 and later.

`C-c C-f`
`C-x C-a C-f`
: Run the program until the selected stack frame returns (or until it stops for some other reason).

These commands interpret a numeric argument as a repeat count, when that makes sense.

Because ⟨TAB⟩ serves as a completion command, you can't use it to enter a tab as input to the program you are debugging with GDB. Instead, type `C-q` ⟨TAB⟩ to enter a tab.

23.5.4 GUD Customization

On startup, GUD runs one of the following hooks: `gdb-mode-hook`, if you are using GDB; `dbx-mode-hook`, if you are using DBX; `sdb-mode-hook`, if you are using SDB; `xdb-mode-hook`, if you are using XDB; `perldb-mode-hook`, for Perl debugging mode; `jdb-mode-hook`, for PDB; `jdb-mode-hook`, for JDB. You can use these hooks to define custom key bindings for the debugger interaction buffer. See Section 31.2.3 [Hooks], page 465.

Here is a convenient way to define a command that sends a particular command string to the debugger, and set up a key binding for it in the debugger interaction buffer:

`(gud-def` *function cmdstring binding docstring*`)`

This defines a command named *function* which sends *cmdstring* to the debugger process, and gives it the documentation string *docstring*. You can use the command thus defined in any buffer. If *binding* is non-`nil`, `gud-def` also binds the command to `C-c` *binding* in the GUD buffer's mode and to `C-x C-a` *binding* generally.

The command string *cmdstring* may contain certain '`%`'-sequences that stand for data to be filled in at the time *function* is called:

'`%f`' The name of the current source file. If the current buffer is the GUD buffer, then the "current source file" is the file that the program stopped in.

'`%l`' The number of the current source line. If the current buffer is the GUD buffer, then the "current source line" is the line that the program stopped in.

'`%e`' The text of the C lvalue or function-call expression at or adjacent to point.

'`%a`' The text of the hexadecimal address at or adjacent to point.

'`%p`' The numeric argument of the called function, as a decimal number. If the command is used without a numeric argument, '`%p`' stands for the empty string.

If you don't use '`%p`' in the command string, the command you define ignores any numeric argument.

23.5.5 GUD Tooltips

The Tooltip facility (see Section 17.18 [Tooltips], page 217) provides support for GUD. If GUD support is activated by customizing the `tooltip` group, variable values can be displayed in tooltips by pointing at them with the mouse in the GUD buffer or in source buffers with major modes in the customizable list `tooltip-gud-modes`.

23.6 Executing Lisp Expressions

Emacs has several different major modes for Lisp and Scheme. They are the same in terms of editing commands, but differ in the commands for executing Lisp expressions. Each mode has its own purpose.

Emacs-Lisp mode
: The mode for editing source files of programs to run in Emacs Lisp. This mode defines `C-M-x` to evaluate the current defun. See Section 23.7 [Lisp Libraries], page 339.

Lisp Interaction mode
: The mode for an interactive session with Emacs Lisp. It defines `C-j` to evaluate the sexp before point and insert its value in the buffer. See Section 23.9 [Lisp Interaction], page 342.

Lisp mode
: The mode for editing source files of programs that run in Lisps other than Emacs Lisp. This mode defines `C-M-x` to send the current defun to an inferior Lisp process. See Section 23.10 [External Lisp], page 342.

Inferior Lisp mode
: The mode for an interactive session with an inferior Lisp process. This mode combines the special features of Lisp mode and Shell mode (see Section 30.2.3 [Shell Mode], page 428).

Scheme mode
: Like Lisp mode but for Scheme programs.

Inferior Scheme mode
: The mode for an interactive session with an inferior Scheme process.

Most editing commands for working with Lisp programs are in fact available globally. See Chapter 22 [Programs], page 273.

23.7 Libraries of Lisp Code for Emacs

Lisp code for Emacs editing commands is stored in files whose names conventionally end in '`.el`'. This ending tells Emacs to edit them in Emacs-Lisp mode (see Section 23.6 [Executing Lisp], page 339).

To execute a file of Emacs Lisp code, use `M-x load-file`. This command reads a file name using the minibuffer and then executes the contents of that file as Lisp code. It is not necessary to visit the file first; in any case, this command reads the file as found on disk, not text in an Emacs buffer.

Once a file of Lisp code is installed in the Emacs Lisp library directories, users can load it using `M-x load-library`. Programs can load it by calling

`load-library`, or with `load`, a more primitive function that is similar but accepts some additional arguments.

`M-x load-library` differs from `M-x load-file` in that it searches a sequence of directories and tries three file names in each directory. Suppose your argument is *lib*; the three names are '*lib*`.elc`', '*lib*`.el`', and lastly just '*lib*'. If '*lib*`.elc`' exists, it is by convention the result of compiling '*lib*`.el`'; it is better to load the compiled file, since it will load and run faster.

If `load-library` finds that '*lib*`.el`' is newer than '*lib*`.elc`' file, it prints a warning, because it's likely that somebody made changes to the '`.el`' file and forgot to recompile it.

Because the argument to `load-library` is usually not in itself a valid file name, file name completion is not available. Indeed, when using this command, you usually do not know exactly what file name will be used.

The sequence of directories searched by `M-x load-library` is specified by the variable `load-path`, a list of strings that are directory names. The default value of the list contains the directory where the Lisp code for Emacs itself is stored. If you have libraries of your own, put them in a single directory and add that directory to `load-path`. `nil` in this list stands for the current default directory, but it is probably not a good idea to put `nil` in the list. If you find yourself wishing that `nil` were in the list, most likely what you really want to do is use `M-x load-file` this once.

Often you do not have to give any command to load a library, because the commands defined in the library are set up to *autoload* that library. Trying to run any of those commands calls `load` to load the library; this replaces the autoload definitions with the real ones from the library.

Emacs Lisp code can be compiled into byte-code which loads faster, takes up less space when loaded, and executes faster. See section "Byte Compilation" in *the Emacs Lisp Reference Manual*. By convention, the compiled code for a library goes in a separate file whose name consists of the library source file with '`c`' appended. Thus, the compiled code for '`foo.el`' goes in '`foo.elc`'. That's why `load-library` searches for '`.elc`' files first.

By default, Emacs refuses to load compiled Lisp files which were compiled with XEmacs, a modified versions of Emacs—they can cause Emacs to crash. Set the variable `load-dangerous-libraries` to `t` if you want to try loading them.

23.8 Evaluating Emacs-Lisp Expressions

Lisp programs intended to be run in Emacs should be edited in Emacs-Lisp mode; this happens automatically for file names ending in '`.el`'. By contrast, Lisp mode itself is used for editing Lisp programs intended for other Lisp systems. To switch to Emacs-Lisp mode explicitly, use the command `M-x emacs-lisp-mode`.

For testing of Lisp programs to run in Emacs, it is often useful to evaluate part of the program as it is found in the Emacs buffer. For example, after changing the text of a Lisp function definition, evaluating the definition installs the change for future calls to the function. Evaluation of Lisp expressions is also useful in any kind of editing, for invoking noninteractive functions (functions that are not commands).

`M-:` Read a single Lisp expression in the minibuffer, evaluate it, and print the value in the echo area (`eval-expression`).

`C-x C-e` Evaluate the Lisp expression before point, and print the value in the echo area (`eval-last-sexp`).

`C-M-x` Evaluate the defun containing or after point, and print the value in the echo area (`eval-defun`).

`M-x eval-region`
Evaluate all the Lisp expressions in the region.

`M-x eval-current-buffer`
Evaluate all the Lisp expressions in the buffer.

`M-:` (`eval-expression`) is the most basic command for evaluating a Lisp expression interactively. It reads the expression using the minibuffer, so you can execute any expression on a buffer regardless of what the buffer contains. When the expression is evaluated, the current buffer is once again the buffer that was current when `M-:` was typed.

In Emacs-Lisp mode, the key `C-M-x` is bound to the command `eval-defun`, which parses the defun containing or following point as a Lisp expression and evaluates it. The value is printed in the echo area. This command is convenient for installing in the Lisp environment changes that you have just made in the text of a function definition.

`C-M-x` treats `defvar` expressions specially. Normally, evaluating a `defvar` expression does nothing if the variable it defines already has a value. But `C-M-x` unconditionally resets the variable to the initial value specified in the `defvar` expression. `defcustom` expressions are treated similarly. This special feature is convenient for debugging Lisp programs.

The command `C-x C-e` (`eval-last-sexp`) evaluates the Lisp expression preceding point in the buffer, and displays the value in the echo area. It is available in all major modes, not just Emacs-Lisp mode. It does not treat `defvar` specially.

If `C-M-x`, `C-x C-e`, or `M-:` is given a numeric argument, it inserts the value into the current buffer at point, rather than displaying it in the echo area. The argument's value does not matter.

The most general command for evaluating Lisp expressions from a buffer is `eval-region`. `M-x eval-region` parses the text of the region as one or more Lisp expressions, evaluating them one by one. `M-x eval-current-buffer` is similar but evaluates the entire buffer. This is

a reasonable way to install the contents of a file of Lisp code that you are just ready to test. Later, as you find bugs and change individual functions, use `C-M-x` on each function that you change. This keeps the Lisp world in step with the source file.

The customizable variables `eval-expression-print-level` and `eval-expression-print-length` control the maximum depth and length of lists to print in the result of the evaluation commands before abbreviating them. `eval-expression-debug-on-error` controls whether evaluation errors invoke the debugger when these commands are used.

23.9 Lisp Interaction Buffers

The buffer '`*scratch*`' which is selected when Emacs starts up is provided for evaluating Lisp expressions interactively inside Emacs.

The simplest way to use the '`*scratch*`' buffer is to insert Lisp expressions and type `C-j` after each expression. This command reads the Lisp expression before point, evaluates it, and inserts the value in printed representation before point. The result is a complete typescript of the expressions you have evaluated and their values.

The '`*scratch*`' buffer's major mode is Lisp Interaction mode, which is the same as Emacs-Lisp mode except for the binding of `C-j`.

The rationale for this feature is that Emacs must have a buffer when it starts up, but that buffer is not useful for editing files since a new buffer is made for every file that you visit. The Lisp interpreter typescript is the most useful thing I can think of for the initial buffer to do. Type `M-x lisp-interaction-mode` to put the current buffer in Lisp Interaction mode.

An alternative way of evaluating Emacs Lisp expressions interactively is to use Inferior Emacs-Lisp mode, which provides an interface rather like Shell mode (see Section 30.2.3 [Shell Mode], page 428) for evaluating Emacs Lisp expressions. Type `M-x ielm` to create an '`*ielm*`' buffer which uses this mode.

23.10 Running an External Lisp

Emacs has facilities for running programs in other Lisp systems. You can run a Lisp process as an inferior of Emacs, and pass expressions to it to be evaluated. You can also pass changed function definitions directly from the Emacs buffers in which you edit the Lisp programs to the inferior Lisp process.

To run an inferior Lisp process, type `M-x run-lisp`. This runs the program named `lisp`, the same program you would run by typing `lisp` as a shell command, with both input and output going through an Emacs buffer

named '`*lisp*`'. That is to say, any "terminal output" from Lisp will go into the buffer, advancing point, and any "terminal input" for Lisp comes from text in the buffer. (You can change the name of the Lisp executable file by setting the variable `inferior-lisp-program`.)

To give input to Lisp, go to the end of the buffer and type the input, terminated by ⟨RET⟩. The '`*lisp*`' buffer is in Inferior Lisp mode, which combines the special characteristics of Lisp mode with most of the features of Shell mode (see Section 30.2.3 [Shell Mode], page 428). The definition of ⟨RET⟩ to send a line to a subprocess is one of the features of Shell mode.

For the source files of programs to run in external Lisps, use Lisp mode. This mode can be selected with `M-x lisp-mode`, and is used automatically for files whose names end in '`.l`', '`.lsp`', or '`.lisp`', as most Lisp systems usually expect.

When you edit a function in a Lisp program you are running, the easiest way to send the changed definition to the inferior Lisp process is the key `C-M-x`. In Lisp mode, this runs the function `lisp-eval-defun`, which finds the defun around or following point and sends it as input to the Lisp process. (Emacs can send input to any inferior process regardless of what buffer is current.)

Contrast the meanings of `C-M-x` in Lisp mode (for editing programs to be run in another Lisp system) and Emacs-Lisp mode (for editing Lisp programs to be run in Emacs): in both modes it has the effect of installing the function definition that point is in, but the way of doing so is different according to where the relevant Lisp environment is found. See Section 23.6 [Executing Lisp], page 339.

24 Abbrevs

A defined *abbrev* is a word which *expands*, if you insert it, into some different text. Abbrevs are defined by the user to expand in specific ways. For example, you might define '`foo`' as an abbrev expanding to '`find outer otter`'. Then you would be able to insert '`find outer otter` ' into the buffer by typing `f o o` ⟨SPC⟩.

A second kind of abbreviation facility is called *dynamic abbrev expansion*. You use dynamic abbrev expansion with an explicit command to expand the letters in the buffer before point by looking for other words in the buffer that start with those letters. See Section 24.6 [Dynamic Abbrevs], page 349.

"Hippie" expansion generalizes abbreviation expansion. See section ""Hippie" Expansion" in *Features for Automatic Typing*.

24.1 Abbrev Concepts

An *abbrev* is a word which has been defined to *expand* into a specified *expansion*. When you insert a word-separator character following the abbrev, that expands the abbrev—replacing the abbrev with its expansion. For example, if '`foo`' is defined as an abbrev expanding to '`find outer otter`', then you can insert '`find outer otter.`' into the buffer by typing `f o o .`.

Abbrevs expand only when Abbrev mode (a minor mode) is enabled. Disabling Abbrev mode does not cause abbrev definitions to be forgotten, but they do not expand until Abbrev mode is enabled again. The command `M-x abbrev-mode` toggles Abbrev mode; with a numeric argument, it turns Abbrev mode on if the argument is positive, off otherwise. See Section 31.1 [Minor Modes], page 455. `abbrev-mode` is also a variable; Abbrev mode is on when the variable is non-`nil`. The variable `abbrev-mode` automatically becomes local to the current buffer when it is set.

Abbrev definitions can be *mode-specific*—active only in one major mode. Abbrevs can also have *global* definitions that are active in all major modes. The same abbrev can have a global definition and various mode-specific definitions for different major modes. A mode-specific definition for the current major mode overrides a global definition.

Abbrevs can be defined interactively during the editing session. Lists of abbrev definitions can also be saved in files and reloaded in later sessions. Some users keep extensive lists of abbrevs that they load in every session.

24.2 Defining Abbrevs

`C-x a g` Define an abbrev, using one or more words before point as its expansion (`add-global-abbrev`).

`C-x a l` Similar, but define an abbrev specific to the current major mode (`add-mode-abbrev`).

`C-x a i g` Define a word in the buffer as an abbrev (`inverse-add-global-abbrev`).

`C-x a i l` Define a word in the buffer as a mode-specific abbrev (`inverse-add-mode-abbrev`).

`M-x kill-all-abbrevs`
This command discards all abbrev definitions currently in effect, leaving a blank slate.

The usual way to define an abbrev is to enter the text you want the abbrev to expand to, position point after it, and type `C-x a g` (`add-global-abbrev`). This reads the abbrev itself using the minibuffer, and then defines it as an abbrev for one or more words before point. Use a numeric argument to say how many words before point should be taken as the expansion. For example, to define the abbrev '`foo`' as mentioned above, insert the text '`find outer otter`' and then type `C-u 3 C-x a g f o o` ⟨RET⟩.

An argument of zero to `C-x a g` means to use the contents of the region as the expansion of the abbrev being defined.

The command `C-x a l` (`add-mode-abbrev`) is similar, but defines a mode-specific abbrev. Mode-specific abbrevs are active only in a particular major mode. `C-x a l` defines an abbrev for the major mode in effect at the time `C-x a l` is typed. The arguments work the same as for `C-x a g`.

If the text already in the buffer is the abbrev, rather than its expansion, use command `C-x a i g` (`inverse-add-global-abbrev`) instead of `C-x a g`, or use `C-x a i l` (`inverse-add-mode-abbrev`) instead of `C-x a l`. These commands are called "inverse" because they invert the meaning of the two text strings they use (one from the buffer and one read with the minibuffer).

To change the definition of an abbrev, just define a new definition. When the abbrev has a prior definition, the abbrev definition commands ask for confirmation for replacing it.

To remove an abbrev definition, give a negative argument to the abbrev definition command: `C-u - C-x a g` or `C-u - C-x a l`. The former removes a global definition, while the latter removes a mode-specific definition.

`M-x kill-all-abbrevs` removes all the abbrev definitions there are, both global and local.

24.3 Controlling Abbrev Expansion

An abbrev expands whenever it is present in the buffer just before point and you type a self-inserting whitespace or punctuation character (⟨SPC⟩, comma, etc.). More precisely, any character that is not a word constituent expands an abbrev, and any word-constituent character can be part of an abbrev. The most common way to use an abbrev is to insert it and then insert a punctuation character to expand it.

Abbrev expansion preserves case; thus, '`foo`' expands into '`find outer otter`'; '`Foo`' into '`Find outer otter`', and '`FOO`' into '`FIND OUTER OTTER`' or '`Find Outer Otter`' according to the variable `abbrev-all-caps` (a non-`nil` value chooses the first of the two expansions).

These commands are used to control abbrev expansion:

`M-'` Separate a prefix from a following abbrev to be expanded (`abbrev-prefix-mark`).

`C-x a e` Expand the abbrev before point (`expand-abbrev`). This is effective even when Abbrev mode is not enabled.

`M-x expand-region-abbrevs`
Expand some or all abbrevs found in the region.

You may wish to expand an abbrev with a prefix attached; for example, if '`cnst`' expands into '`construction`', you might want to use it to enter '`reconstruction`'. It does not work to type `recnst`, because that is not necessarily a defined abbrev. What you can do is use the command `M-'` (`abbrev-prefix-mark`) in between the prefix '`re`' and the abbrev '`cnst`'. First, insert '`re`'. Then type `M-'`; this inserts a hyphen in the buffer to indicate that it has done its work. Then insert the abbrev '`cnst`'; the buffer now contains '`re-cnst`'. Now insert a non-word character to expand the abbrev '`cnst`' into '`construction`'. This expansion step also deletes the hyphen that indicated `M-'` had been used. The result is the desired '`reconstruction`'.

If you actually want the text of the abbrev in the buffer, rather than its expansion, you can accomplish this by inserting the following punctuation with `C-q`. Thus, `foo C-q ,` leaves '`foo,`' in the buffer.

If you expand an abbrev by mistake, you can undo the expansion and bring back the abbrev itself by typing `C-_` to undo (see Section 4.4 [Undo], page 45). This also undoes the insertion of the non-word character that expanded the abbrev. If the result you want is the terminating non-word character plus the unexpanded abbrev, you must reinsert the terminating character, quoting it with `C-q`. You can also use the command `M-x unexpand-abbrev` to cancel the last expansion without deleting the terminating character.

`M-x expand-region-abbrevs` searches through the region for defined abbrevs, and for each one found offers to replace it with its expansion. This command is useful if you have typed in text using abbrevs but forgot to turn on Abbrev mode first. It may also be useful together with a special set

of abbrev definitions for making several global replacements at once. This command is effective even if Abbrev mode is not enabled.

Expanding an abbrev runs the hook `pre-abbrev-expand-hook` (see Section 31.2.3 [Hooks], page 465).

24.4 Examining and Editing Abbrevs

`M-x list-abbrevs`
: Display a list of all abbrev definitions. With numeric argument, list only local abbrevs.

`M-x edit-abbrevs`
: Edit a list of abbrevs; you can add, alter or remove definitions.

The output from `M-x list-abbrevs` looks like this:

```
(lisp-mode-abbrev-table)
"dk"        0    "define-key"
(global-abbrev-table)
"dfn"        0    "definition"
```

(Some blank lines of no semantic significance, and some other abbrev tables, have been omitted.)

A line containing a name in parentheses is the header for abbrevs in a particular abbrev table; `global-abbrev-table` contains all the global abbrevs, and the other abbrev tables that are named after major modes contain the mode-specific abbrevs.

Within each abbrev table, each nonblank line defines one abbrev. The word at the beginning of the line is the abbrev. The number that follows is the number of times the abbrev has been expanded. Emacs keeps track of this to help you see which abbrevs you actually use, so that you can eliminate those that you don't use often. The string at the end of the line is the expansion.

`M-x edit-abbrevs` allows you to add, change or kill abbrev definitions by editing a list of them in an Emacs buffer. The list has the same format described above. The buffer of abbrevs is called '`*Abbrevs*`', and is in Edit-Abbrevs mode. Type `C-c C-c` in this buffer to install the abbrev definitions as specified in the buffer—and delete any abbrev definitions not listed.

The command `edit-abbrevs` is actually the same as `list-abbrevs` except that it selects the buffer '`*Abbrevs*`' whereas `list-abbrevs` merely displays it in another window.

24.5 Saving Abbrevs

These commands allow you to keep abbrev definitions between editing sessions.

`M-x write-abbrev-file` ⟨RET⟩ *file* ⟨RET⟩
: Write a file *file* describing all defined abbrevs.

`M-x read-abbrev-file` ⟨RET⟩ *file* ⟨RET⟩
: Read the file *file* and define abbrevs as specified therein.

`M-x quietly-read-abbrev-file` ⟨RET⟩ *file* ⟨RET⟩
: Similar but do not display a message about what is going on.

`M-x define-abbrevs`
: Define abbrevs from definitions in current buffer.

`M-x insert-abbrevs`
: Insert all abbrevs and their expansions into current buffer.

`M-x write-abbrev-file` reads a file name using the minibuffer and then writes a description of all current abbrev definitions into that file. This is used to save abbrev definitions for use in a later session. The text stored in the file is a series of Lisp expressions that, when executed, define the same abbrevs that you currently have.

`M-x read-abbrev-file` reads a file name using the minibuffer and then reads the file, defining abbrevs according to the contents of the file. `M-x quietly-read-abbrev-file` is the same except that it does not display a message in the echo area saying that it is doing its work; it is actually useful primarily in the '`.emacs`' file. If an empty argument is given to either of these functions, they use the file name specified in the variable `abbrev-file-name`, which is by default `"~/.abbrev_defs"`.

Emacs will offer to save abbrevs automatically if you have changed any of them, whenever it offers to save all files (for `C-x s` or `C-x C-c`). This feature can be inhibited by setting the variable `save-abbrevs` to `nil`.

The commands `M-x insert-abbrevs` and `M-x define-abbrevs` are similar to the previous commands but work on text in an Emacs buffer. `M-x insert-abbrevs` inserts text into the current buffer before point, describing all current abbrev definitions; `M-x define-abbrevs` parses the entire current buffer and defines abbrevs accordingly.

24.6 Dynamic Abbrev Expansion

The abbrev facility described above operates automatically as you insert text, but all abbrevs must be defined explicitly. By contrast, *dynamic abbrevs* allow the meanings of abbrevs to be determined automatically from the contents of the buffer, but dynamic abbrev expansion happens only when you request it explicitly.

`M-/` Expand the word in the buffer before point as a *dynamic abbrev*, by searching in the buffer for words starting with that abbreviation (`dabbrev-expand`).

`C-M-/` Complete the word before point as a dynamic abbrev (`dabbrev-completion`).

For example, if the buffer contains '`does this follow` ' and you type `f o M-/`, the effect is to insert '`follow`' because that is the last word in the buffer that starts with '`fo`'. A numeric argument to `M-/` says to take the second, third, etc. distinct expansion found looking backward from point. Repeating `M-/` searches for an alternative expansion by looking farther back. After scanning all the text before point, it searches the text after point. The variable `dabbrev-limit`, if non-`nil`, specifies how far in the buffer to search for an expansion.

After scanning the current buffer, `M-/` normally searches other buffers, unless you have set `dabbrev-check-all-buffers` to `nil`.

For finer control over which buffers to scan, customize the variable `dabbrev-ignored-buffer-regexps`. Its value is a list of regular expressions. If a buffer's name matches any of these regular expressions, dynamic abbrev expansion skips that buffer.

A negative argument to `M-/`, as in `C-u - M-/`, says to search first for expansions after point, and second for expansions before point. If you repeat the `M-/` to look for another expansion, do not specify an argument. This tries all the expansions after point and then the expansions before point.

After you have expanded a dynamic abbrev, you can copy additional words that follow the expansion in its original context. Simply type ⟨SPC⟩ `M-/` for each word you want to copy. The spacing and punctuation between words is copied along with the words.

The command `C-M-/` (`dabbrev-completion`) performs completion of a dynamic abbreviation. Instead of trying the possible expansions one by one, it finds all of them, then inserts the text that they have in common. If they have nothing in common, `C-M-/` displays a list of completions, from which you can select a choice in the usual manner. See Section 5.3 [Completion], page 57.

Dynamic abbrev expansion is completely independent of Abbrev mode; the expansion of a word with `M-/` is completely independent of whether it has a definition as an ordinary abbrev.

24.7 Customizing Dynamic Abbreviation

Normally, dynamic abbrev expansion ignores case when searching for expansions. That is, the expansion need not agree in case with the word you are expanding.

This feature is controlled by the variable `dabbrev-case-fold-search`. If it is `t`, case is ignored in this search; if `nil`, the word and the expansion must match in case. If the value of `dabbrev-case-fold-search` is `case-fold-search`, which is true by default, then the variable `case-fold-search` controls whether to ignore case while searching for expansions.

Normally, dynamic abbrev expansion preserves the case pattern *of the abbrev you have typed*, by converting the expansion to that case pattern.

The variable `dabbrev-case-replace` controls whether to preserve the case pattern of the abbrev. If it is `t`, the abbrev's case pattern is preserved in most cases; if `nil`, the expansion is always copied verbatim. If the value of `dabbrev-case-replace` is `case-replace`, which is true by default, then the variable `case-replace` controls whether to copy the expansion verbatim.

However, if the expansion contains a complex mixed case pattern, and the abbrev matches this pattern as far as it goes, then the expansion is always copied verbatim, regardless of those variables. Thus, for example, if the buffer contains `variableWithSillyCasePattern`, and you type `v a M-/`, it copies the expansion verbatim including its case pattern.

The variable `dabbrev-abbrev-char-regexp`, if non-`nil`, controls which characters are considered part of a word, for dynamic expansion purposes. The regular expression must match just one character, never two or more. The same regular expression also determines which characters are part of an expansion. The value `nil` has a special meaning: abbreviations are made of word characters, but expansions are made of word and symbol characters.

In shell scripts and makefiles, a variable name is sometimes prefixed with '`$`' and sometimes not. Major modes for this kind of text can customize dynamic abbreviation to handle optional prefixes by setting the variable `dabbrev-abbrev-skip-leading-regexp`. Its value should be a regular expression that matches the optional prefix that dynamic abbreviation should ignore.

25 Editing Pictures

To edit a picture made out of text characters (for example, a picture of the division of a register into fields, as a comment in a program), use the command `M-x edit-picture` to enter Picture mode.

In Picture mode, editing is based on the *quarter-plane* model of text, according to which the text characters lie studded on an area that stretches infinitely far to the right and downward. The concept of the end of a line does not exist in this model; the most you can say is where the last nonblank character on the line is found.

Of course, Emacs really always considers text as a sequence of characters, and lines really do have ends. But Picture mode replaces the most frequently-used commands with variants that simulate the quarter-plane model of text. They do this by inserting spaces or by converting tabs to spaces.

Most of the basic editing commands of Emacs are redefined by Picture mode to do essentially the same thing but in a quarter-plane way. In addition, Picture mode defines various keys starting with the `C-c` prefix to run special picture editing commands.

One of these keys, `C-c C-c`, is pretty important. Often a picture is part of a larger file that is usually edited in some other major mode. `M-x edit-picture` records the name of the previous major mode so you can use the `C-c C-c` command (`picture-mode-exit`) later to go back to that mode. `C-c C-c` also deletes spaces from the ends of lines, unless given a numeric argument.

The special commands of Picture mode all work in other modes (provided the '`picture`' library is loaded), but are not bound to keys except in Picture mode. The descriptions below talk of moving "one column" and so on, but all the picture mode commands handle numeric arguments as their normal equivalents do.

Turning on Picture mode runs the hook `picture-mode-hook` (see Section 31.2.3 [Hooks], page 465).

25.1 Basic Editing in Picture Mode

Most keys do the same thing in Picture mode that they usually do, but do it in a quarter-plane style. For example, `C-f` is rebound to run `picture-forward-column`, a command which moves point one column to the right, inserting a space if necessary so that the actual end of the line makes no difference. `C-b` is rebound to run `picture-backward-column`, which always moves point left one column, converting a tab to multiple spaces if necessary. `C-n` and `C-p` are rebound to run `picture-move-down` and `picture-move-up`, which can either insert spaces or convert tabs as necessary to make sure that

point stays in exactly the same column. `C-e` runs `picture-end-of-line`, which moves to after the last nonblank character on the line. There is no need to change `C-a`, as the choice of screen model does not affect beginnings of lines.

Insertion of text is adapted to the quarter-plane screen model through the use of Overwrite mode (see Section 31.1 [Minor Modes], page 455). Self-inserting characters replace existing text, column by column, rather than pushing existing text to the right. ⟨RET⟩ runs `picture-newline`, which just moves to the beginning of the following line so that new text will replace that line.

Picture mode provides erasure instead of deletion and killing of text. ⟨DEL⟩ (`picture-backward-clear-column`) replaces the preceding character with a space rather than removing it; this moves point backwards. `C-d` (`picture-clear-column`) replaces the next character or characters with spaces, but does not move point. (If you want to clear characters to spaces and move forward over them, use ⟨SPC⟩.) `C-k` (`picture-clear-line`) really kills the contents of lines, but does not delete the newlines from the buffer.

To do actual insertion, you must use special commands. `C-o` (`picture-open-line`) creates a blank line after the current line; it never splits a line. `C-M-o` (`split-line`) makes sense in Picture mode, so it is not changed. `C-j` (`picture-duplicate-line`) inserts below the current line another line with the same contents.

To do actual deletion in Picture mode, use `C-w`, `C-c C-d` (which is defined as `delete-char`, as `C-d` is in other modes), or one of the picture rectangle commands (see Section 25.4 [Rectangles in Picture], page 355).

25.2 Controlling Motion after Insert

Since "self-inserting" characters in Picture mode overwrite and move point, there is no essential restriction on how point should be moved. Normally point moves right, but you can specify any of the eight orthogonal or diagonal directions for motion after a "self-inserting" character. This is useful for drawing lines in the buffer.

`C-c <` Move left after insertion (`picture-movement-left`).

`C-c >` Move right after insertion (`picture-movement-right`).

`C-c ^` Move up after insertion (`picture-movement-up`).

`C-c .` Move down after insertion (`picture-movement-down`).

`C-c ‘` Move up and left ("northwest") after insertion (`picture-movement-nw`).

`C-c ’` Move up and right ("northeast") after insertion (`picture-movement-ne`).

`C-c /` Move down and left ("southwest") after insertion (`picture-movement-sw`).

`C-c \` Move down and right ("southeast") after insertion (`picture-movement-se`).

Two motion commands move based on the current Picture insertion direction. The command `C-c C-f` (`picture-motion`) moves in the same direction as motion after "insertion" currently does, while `C-c C-b` (`picture-motion-reverse`) moves in the opposite direction.

25.3 Picture Mode Tabs

Two kinds of tab-like action are provided in Picture mode. Use `M-`⟨TAB⟩ (`picture-tab-search`) for context-based tabbing. With no argument, it moves to a point underneath the next "interesting" character that follows whitespace in the previous nonblank line. "Next" here means "appearing at a horizontal position greater than the one point starts out at." With an argument, as in `C-u M-`⟨TAB⟩, this command moves to the next such interesting character in the current line. `M-`⟨TAB⟩ does not change the text; it only moves point. "Interesting" characters are defined by the variable `picture-tab-chars`, which should define a set of characters. The syntax for this variable is like the syntax used inside of '`[...]`' in a regular expression—but without the '`[`' and the '`]`'. Its default value is `"!-~"`.

⟨TAB⟩ itself runs `picture-tab`, which operates based on the current tab stop settings; it is the Picture mode equivalent of `tab-to-tab-stop`. Normally it just moves point, but with a numeric argument it clears the text that it moves over.

The context-based and tab-stop-based forms of tabbing are brought together by the command `C-c` ⟨TAB⟩ (`picture-set-tab-stops`). This command sets the tab stops to the positions which `M-`⟨TAB⟩ would consider significant in the current line. The use of this command, together with ⟨TAB⟩, can get the effect of context-based tabbing. But `M-`⟨TAB⟩ is more convenient in the cases where it is sufficient.

It may be convenient to prevent use of actual tab characters in pictures. For example, this prevents `C-x` ⟨TAB⟩ from messing up the picture. You can do this by setting the variable `indent-tabs-mode` to `nil`. See Section 20.3 [Just Spaces], page 242.

25.4 Picture Mode Rectangle Commands

Picture mode defines commands for working on rectangular pieces of the text in ways that fit with the quarter-plane model. The standard rectangle commands may also be useful (see Section 9.4 [Rectangles], page 93).

`C-c C-k` Clear out the region-rectangle with spaces (`picture-clear-rectangle`). With argument, delete the text.

`C-c C-w` *r* Similar but save rectangle contents in register *r* first (`picture-clear-rectangle-to-register`).

`C-c C-y` Copy last killed rectangle into the buffer by overwriting, with upper left corner at point (`picture-yank-rectangle`). With argument, insert instead.

`C-c C-x` *r* Similar, but use the rectangle in register *r* (`picture-yank-rectangle-from-register`).

The picture rectangle commands `C-c C-k` (`picture-clear-rectangle`) and `C-c C-w` (`picture-clear-rectangle-to-register`) differ from the standard rectangle commands in that they normally clear the rectangle instead of deleting it; this is analogous with the way `C-d` is changed in Picture mode.

However, deletion of rectangles can be useful in Picture mode, so these commands delete the rectangle if given a numeric argument. `C-c C-k` either with or without a numeric argument saves the rectangle for `C-c C-y`.

The Picture mode commands for yanking rectangles differ from the standard ones in overwriting instead of inserting. This is the same way that Picture mode insertion of other text differs from other modes. `C-c C-y` (`picture-yank-rectangle`) inserts (by overwriting) the rectangle that was most recently killed, while `C-c C-x` (`picture-yank-rectangle-from-register`) does likewise for the rectangle found in a specified register.

26 Sending Mail

To send a message in Emacs, you start by typing a command (`C-x m`) to select and initialize the '`*mail*`' buffer. Then you edit the text and headers of the message in this buffer, and type another command (`C-c C-s` or `C-c C-c`) to send the message.

`C-x m` Begin composing a message to send (`compose-mail`).

`C-x 4 m` Likewise, but display the message in another window (`compose-mail-other-window`).

`C-x 5 m` Likewise, but make a new frame (`compose-mail-other-frame`).

`C-c C-s` In Mail mode, send the message (`mail-send`).

`C-c C-c` Send the message and bury the mail buffer (`mail-send-and-exit`).

The command `C-x m` (`compose-mail`) selects a buffer named '`*mail*`' and initializes it with the skeleton of an outgoing message. `C-x 4 m` (`compose-mail-other-window`) selects the '`*mail*`' buffer in a different window, leaving the previous current buffer visible. `C-x 5 m` (`compose-mail-other-frame`) creates a new frame to select the '`*mail*`' buffer.

Because the mail-composition buffer is an ordinary Emacs buffer, you can switch to other buffers while in the middle of composing mail, and switch back later (or never). If you use the `C-x m` command again when you have been composing another message but have not sent it, you are asked to confirm before the old message is erased. If you answer `n`, the '`*mail*`' buffer is left selected with its old contents, so you can finish the old message and send it. `C-u C-x m` is another way to do this. Sending the message marks the '`*mail*`' buffer "unmodified," which avoids the need for confirmation when `C-x m` is next used.

If you are composing a message in the '`*mail*`' buffer and want to send another message before finishing the first, rename the '`*mail*`' buffer using `M-x rename-uniquely` (see Section 15.3 [Misc Buffer], page 187). Then you can use `C-x m` or its variants described above to make a new '`*mail*`' buffer. Once you've done that, you can work with each mail buffer independently.

26.1 The Format of the Mail Buffer

In addition to the *text* or *body*, a message has *header fields* which say who sent it, when, to whom, why, and so on. Some header fields, such as '`Date`' and '`Sender`', are created automatically when you send the message. Others, such as the recipient names, must be specified by you in order to send the message properly.

Mail mode provides a few commands to help you edit some header fields, and some are preinitialized in the buffer automatically at times. You can insert and edit header fields using ordinary editing commands.

The line in the buffer that says

```
--text follows this line--
```

is a special delimiter that separates the headers you have specified from the text. Whatever follows this line is the text of the message; the headers precede it. The delimiter line itself does not appear in the message actually sent. The text used for the delimiter line is controlled by the variable `mail-header-separator`.

Here is an example of what the headers and text in the mail buffer might look like.

```
To: gnu@gnu.org
CC: lungfish@spam.org, byob@spam.org
Subject: The Emacs Manual
--Text follows this line--
Please ignore this message.
```

26.2 Mail Header Fields

A header field in the mail buffer starts with a field name at the beginning of a line, terminated by a colon. Upper and lower case are equivalent in field names (and in mailing addresses also). After the colon and optional whitespace comes the contents of the field.

You can use any name you like for a header field, but normally people use only standard field names with accepted meanings. Here is a table of fields commonly used in outgoing messages.

‘`To`’ This field contains the mailing addresses to which the message is addressed. If you list more than one address, use commas, not spaces, to separate them.

‘`Subject`’ The contents of the ‘`Subject`’ field should be a piece of text that says what the message is about. The reason ‘`Subject`’ fields are useful is that most mail-reading programs can provide a summary of messages, listing the subject of each message but not its text.

‘`CC`’ This field contains additional mailing addresses to send the message to, like ‘`To`’ except that these readers should not regard the message as directed at them.

‘`BCC`’ This field contains additional mailing addresses to send the message to, which should not appear in the header of the message actually sent. Copies sent this way are called *blind carbon copies*.

To send a blind carbon copy of every outgoing message to yourself, set the variable `mail-self-blind` to `t`. To send a blind carbon copy of every message to some other *address*, set the variable `mail-default-headers` to `"Bcc: address\n"`.

'`FCC`' This field contains the name of one file and directs Emacs to append a copy of the message to that file when you send the message. If the file is in Rmail format, Emacs writes the message in Rmail format; otherwise, Emacs writes the message in system mail file format.

To put a fixed file name in the '`FCC`' field each time you start editing an outgoing message, set the variable `mail-archive-file-name` to that file name. Unless you remove the '`FCC`' field before sending, the message will be written into that file when it is sent.

'`From`' Use the '`From`' field to say who you are, when the account you are using to send the mail is not your own. The contents of the '`From`' field should be a valid mailing address, since replies will normally go there. If you don't specify the '`From`' field yourself, Emacs uses the value of `user-mail-address` as the default.

'`Reply-to`'
Use this field to direct replies to a different address. Most mail-reading programs (including Rmail) automatically send replies to the '`Reply-to`' address in preference to the '`From`' address. By adding a '`Reply-to`' field to your header, you can work around any problems your '`From`' address may cause for replies.

To put a fixed '`Reply-to`' address into every outgoing message, set the variable `mail-default-reply-to` to that address (as a string). Then `mail` initializes the message with a '`Reply-to`' field as specified. You can delete or alter that header field before you send the message, if you wish. When Emacs starts up, if the environment variable `REPLYTO` is set, `mail-default-reply-to` is initialized from that environment variable.

'`In-reply-to`'
This field contains a piece of text describing a message you are replying to. Some mail systems can use this information to correlate related pieces of mail. Normally this field is filled in by Rmail when you reply to a message in Rmail, and you never need to think about it (see Chapter 27 [Rmail], page 367).

'`References`'
This field lists the message IDs of related previous messages. Rmail sets up this field automatically when you reply to a message.

The ‘To’, ‘CC’, ‘BCC’ and ‘FCC’ header fields can appear any number of times, and each such header field can contain multiple addresses, separated by commas. This way, you can specify any number of places to send the message. A ‘To’, ‘CC’, or ‘BCC’ field can also have continuation lines: one or more lines starting with whitespace, following the starting line of the field, are considered part of the field. Here’s an example of a ‘To’ field with a continuation line:

```
To: foo@here.net, this@there.net,
  me@gnu.cambridge.mass.usa.earth.spiral3281
```

When you send the message, if you didn’t write a ‘From’ field yourself, Emacs puts in one for you. The variable `mail-from-style` controls the format:

`nil` Use just the email address, as in ‘king@grassland.com’.

`parens` Use both email address and full name, as in ‘king@grassland.com (Elvis Parsley)’.

`angles` Use both email address and full name, as in ‘Elvis Parsley <king@grassland.com>’.

`system-default`
Allow the system to insert the ‘From’ field.

You can direct Emacs to insert certain default headers into the outgoing message by setting the variable `mail-default-headers` to a string. Then `C-x m` inserts this string into the message headers. If the default header fields are not appropriate for a particular message, edit them as appropriate before sending the message.

26.3 Mail Aliases

You can define *mail aliases* in a file named ‘~/.mailrc’. These are short mnemonic names which stand for mail addresses or groups of mail addresses. Like many other mail programs, Emacs expands aliases when they occur in the ‘To’, ‘From’, ‘CC’, ‘BCC’, and ‘Reply-to’ fields, plus their ‘Resent-’ variants.

To define an alias in ‘~/.mailrc’, write a line in the following format:

`alias` *shortaddress fulladdresses*

Here *fulladdresses* stands for one or more mail addresses for *shortaddress* to expand into. Separate multiple addresses with spaces; if an address contains a space, quote the whole address with a pair of double-quotes.

For instance, to make `maingnu` stand for `gnu@gnu.org` plus a local address of your own, put in this line:

```
alias maingnu gnu@gnu.org local-gnu
```

Emacs also recognizes include commands in '`.mailrc`' files. They look like this:

```
source filename
```

The file '`~/.mailrc`' is used primarily by other mail-reading programs; it can contain various other commands. Emacs ignores everything in it except for alias definitions and include commands.

Another way to define a mail alias, within Emacs alone, is with the `define-mail-alias` command. It prompts for the alias and then the full address. You can use it to define aliases in your '`.emacs`' file, like this:

```
(define-mail-alias "maingnu" "gnu@gnu.org")
```

`define-mail-alias` records aliases by adding them to a variable named `mail-aliases`. If you are comfortable with manipulating Lisp lists, you can set `mail-aliases` directly. The initial value of `mail-aliases` is `t`, which means that Emacs should read '`.mailrc`' to get the proper value.

You can specify a different file name to use instead of '`~/.mailrc`' by setting the variable `mail-personal-alias-file`.

Normally, Emacs expands aliases when you send the message. You do not need to expand mail aliases before sending the message, but you can expand them if you want to see where the mail will actually go. To do this, use the command `M-x expand-mail-aliases`; it expands all mail aliases currently present in the mail headers that hold addresses.

If you like, you can have mail aliases expand as abbrevs, as soon as you type them in (see Chapter 24 [Abbrevs], page 345). To enable this feature, execute the following:

```
(add-hook 'mail-mode-hook 'mail-abbrevs-setup)
```

This can go in your '`.emacs`' file. See Section 31.2.3 [Hooks], page 465. If you use this feature, you must use `define-mail-abbrev` instead of `define-mail-alias`; the latter does not work with this package. Note that the mail abbreviation package uses the variable `mail-abbrevs` instead of `mail-aliases`, and that all alias names are converted to lower case.

The mail abbreviation package also provides the `C-c C-a` (`mail-interactive-insert-alias`) command, which reads an alias name (with completion) and inserts its definition at point. This is useful when editing the message text itself or a header field such as '`Subject`' in which Emacs does not normally expand aliases.

Note that abbrevs expand only if you insert a word-separator character afterward. However, you can rebind `C-n` and `M->` to cause expansion as well. Here's how to do that:

```
(add-hook 'mail-mode-hook
          (lambda ()
            (substitute-key-definition
              'next-line 'mail-abbrev-next-line
              mail-mode-map global-map)
```

```
(substitute-key-definition
  'end-of-buffer 'mail-abbrev-end-of-buffer
  mail-mode-map global-map)))
```

26.4 Mail Mode

The major mode used in the mail buffer is Mail mode, which is much like Text mode except that various special commands are provided on the `C-c` prefix. These commands all have to do specifically with editing or sending the message. In addition, Mail mode defines the character '`%`' as a word separator; this is helpful for using the word commands to edit mail addresses.

Mail mode is normally used in buffers set up automatically by the `mail` command and related commands. However, you can also switch to Mail mode in a file-visiting buffer. That is a useful thing to do if you have saved draft message text in a file.

26.4.1 Mail Sending

Mail mode has two commands for sending the message you have been editing:

`C-c C-s` Send the message, and leave the mail buffer selected (`mail-send`).

`C-c C-c` Send the message, and select some other buffer (`mail-send-and-exit`).

`C-c C-s` (`mail-send`) sends the message and marks the mail buffer unmodified, but leaves that buffer selected so that you can modify the message (perhaps with new recipients) and send it again. `C-c C-c` (`mail-send-and-exit`) sends and then deletes the window or switches to another buffer. It puts the mail buffer at the lowest priority for reselection by default, since you are finished with using it. This is the usual way to send the message.

In a file-visiting buffer, sending the message does not clear the modified flag, because only saving the file should do that. As a result, you don't get a warning if you try to send the same message twice.

When you send a message that contains non-ASCII characters, they need to be encoded with a coding system (see Section 18.6 [Coding Systems], page 223). Usually the coding system is specified automatically by your chosen language environment (see Section 18.3 [Language Environments], page 220). You can explicitly specify the coding system for outgoing mail by setting the variable `sendmail-coding-system` (see Section 18.7 [Recognize Coding], page 225).

If the coding system thus determined does not handle the characters in a particular message, Emacs asks you to select the coding system to use, showing a list of possible coding systems.

26.4.2 Mail Header Editing

Mail mode provides special commands to move to particular header fields and to complete addresses in headers.

`C-c C-f C-t`
: Move to the '`To`' header field, creating one if there is none (`mail-to`).

`C-c C-f C-s`
: Move to the '`Subject`' header field, creating one if there is none (`mail-subject`).

`C-c C-f C-c`
: Move to the '`CC`' header field, creating one if there is none (`mail-cc`).

`C-c C-f C-b`
: Move to the '`BCC`' header field, creating one if there is none (`mail-bcc`).

`C-c C-f C-f`
: Move to the '`FCC`' header field, creating one if there is none (`mail-fcc`).

`M-`⟨TAB⟩ Complete a mailing address (`mail-complete`).

There are five commands to move point to particular header fields, all based on the prefix `C-c C-f` ('`C-f`' is for "field"). They are listed in the table above. If the field in question does not exist, these commands create one. We provide special motion commands for these particular fields because they are the fields users most often want to edit.

While editing a header field that contains mailing addresses, such as '`To:`', '`CC:`' and '`BCC:`', you can complete a mailing address by typing `M-`⟨TAB⟩ (`mail-complete`). It inserts the full name corresponding to the address, if it can determine the full name. The variable `mail-complete-style` controls whether to insert the full name, and what style to use, as in `mail-from-style` (see Section 26.2 [Mail Headers], page 358).

For completion purposes, the valid mailing addresses are taken to be the local users' names plus your personal mail aliases. You can specify additional sources of valid addresses; use the customization buffer to see the options for this.

If you type `M-`⟨TAB⟩ in the body of the message, it invokes `ispell-complete-word`, as in Text mode.

26.4.3 Citing Mail

Mail mode also has commands for yanking or *citing* all or part of a message that you are replying to. These commands are active only when you started sending a message using an Rmail command.

`C-c C-y` Yank the selected message from Rmail (`mail-yank-original`).

`C-c C-r` Yank the region from the Rmail buffer (`mail-yank-region`).

`C-c C-q` Fill each paragraph cited from another message (`mail-fill-yanked-message`).

When mail sending is invoked from the Rmail mail reader using an Rmail command, `C-c C-y` can be used inside the mail buffer to insert the text of the message you are replying to. Normally it indents each line of that message three spaces and eliminates most header fields. A numeric argument specifies the number of spaces to indent. An argument of just `C-u` says not to indent at all and not to eliminate anything. `C-c C-y` always uses the current message from the Rmail buffer, so you can insert several old messages by selecting one in Rmail, switching to '`*mail*`' and yanking it, then switching back to Rmail to select another.

You can specify the text for `C-c C-y` to insert at the beginning of each line: set `mail-yank-prefix` to the desired string. (A value of `nil` means to use indentation; this is the default.) However, `C-u C-c C-y` never adds anything at the beginning of the inserted lines, regardless of the value of `mail-yank-prefix`.

To yank just a part of an incoming message, set the region in Rmail to the part you want; then go to the '`*Mail*`' message and type `C-c C-r` (`mail-yank-region`). Each line that is copied is indented or prefixed according to `mail-yank-prefix`.

After using `C-c C-y` or `C-c C-r`, you can type `C-c C-q` (`mail-fill-yanked-message`) to fill the paragraphs of the yanked old message or messages. One use of `C-c C-q` fills all such paragraphs, each one individually. To fill a single paragraph of the quoted message, use `M-q`. If filling does not automatically handle the type of citation prefix you use, try setting the fill prefix explicitly. See Section 21.5 [Filling], page 248.

26.4.4 Mail Mode Miscellany

`C-c C-t` Move to the beginning of the message body text (`mail-text`).

`C-c C-w` Insert the file '`~/.signature`' at the end of the message text (`mail-signature`).

`C-c C-i` *file* ⟨RET⟩
: Insert the contents of *file* at the end of the outgoing message (`mail-attach-file`).

`M-x ispell-message`
: Do spelling correction on the message text, but not on citations from other messages.

`C-c C-t` (`mail-text`) moves point to just after the header separator line—that is, to the beginning of the message body text.

`C-c C-w` (`mail-signature`) adds a standard piece of text at the end of the message to say more about who you are. The text comes from the file '`~/.signature`' in your home directory. To insert your signature automatically, set the variable `mail-signature` to `t`; then starting a mail message automatically inserts the contents of your '`~/.signature`' file. If you want to omit your signature from a particular message, delete it from the buffer before you send the message.

You can also set `mail-signature` to a string; then that string is inserted automatically as your signature when you start editing a message to send. If you set it to some other Lisp expression, the expression is evaluated each time, and its value (which should be a string) specifies the signature.

You can do spelling correction on the message text you have written with the command `M-x ispell-message`. If you have yanked an incoming message into the outgoing draft, this command skips what was yanked, but it checks the text that you yourself inserted. (It looks for indentation or `mail-yank-prefix` to distinguish the cited lines from your input.) See Section 13.4 [Spelling], page 139.

To include a file in the outgoing message, you can use `C-x i`, the usual command to insert a file in the current buffer. But it is often more convenient to use a special command, `C-c C-i` (`mail-attach-file`). This command inserts the file contents at the end of the buffer, after your signature if any, with a delimiter line that includes the file name.

Turning on Mail mode (which `C-x m` does automatically) runs the normal hooks `text-mode-hook` and `mail-mode-hook`. Initializing a new outgoing message runs the normal hook `mail-setup-hook`; if you want to add special fields to your mail header or make other changes to the appearance of the mail buffer, use that hook. See Section 31.2.3 [Hooks], page 465.

The main difference between these hooks is just when they are invoked. Whenever you type `M-x mail`, `mail-mode-hook` runs as soon as the '`*mail*`' buffer is created. Then the `mail-setup` function puts in the default contents of the buffer. After these default contents are inserted, `mail-setup-hook` runs.

26.5 Mail Amusements

`M-x spook` adds a line of randomly chosen keywords to an outgoing mail message. The keywords are chosen from a list of words that suggest you are discussing something subversive.

The idea behind this feature is the suspicion that the NSA[1] snoops on all electronic mail messages that contain keywords suggesting they might find them interesting. (The NSA says they don't, but that's what they *would* say.) The idea is that if lots of people add suspicious words to their messages, the NSA will get so busy with spurious input that they will have to give up reading it all.

Here's how to insert spook keywords automatically whenever you start entering an outgoing message:

```
(add-hook 'mail-setup-hook 'spook)
```

Whether or not this confuses the NSA, it at least amuses people.

You can use the `fortune` program to put a "fortune cookie" message into outpoing mail. To do this, add `fortune-to-signature` to `mail-setup-hook`:

```
(add-hook 'mail-setup-hook 'fortune-to-signature)
```

26.6 Mail-Composition Methods

In this chapter we have described the usual Emacs mode for editing and sending mail—Mail mode. Emacs has alternative facilities for editing and sending mail, including MH-E and Message mode, not documented in this manual. See section "" in *The Emacs Interface to MH*. See section "" in *Message Manual*. You can choose any of them as your preferred method. The commands `C-x m`, `C-x 4 m` and `C-x 5 m` use whichever agent you have specified. So do various other Emacs commands and facilities that send mail.

To specify your mail-composition method, customize the variable `mail-user-agent`. Currently legitimate values include `sendmail-user-agent` (Mail mode), `mh-e-user-agent`, `message-user-agent` and `gnus-user-agent`.

If you select a different mail-composition method, the information in this chapter about the '`*mail*`' buffer and Mail mode does not apply; the other methods use a different format of text in a different buffer, and their commands are different as well.

[1] The US National Security Agency.

27 Reading Mail with Rmail

Rmail is an Emacs subsystem for reading and disposing of mail that you receive. Rmail stores mail messages in files called Rmail files. Reading the message in an Rmail file is done in a special major mode, Rmail mode, which redefines most letters to run commands for managing mail. The command `rmail-mode` is used to switch into Rmail mode, and it runs the hook `rmail-mode-hook` as usual, but don't run this command by hand; it can't do a reasonable job unless the buffer is visiting a proper Rmail file.

27.1 Basic Concepts of Rmail

Using Rmail in the simplest fashion, you have one Rmail file '`~/RMAIL`' in which all of your mail is saved. It is called your *primary Rmail file*. The command `M-x rmail` reads your primary Rmail file, merges new mail in from your inboxes, displays the first message you haven't read yet, and lets you begin reading. The variable `rmail-file-name` specifies the name of the primary Rmail file.

Rmail uses narrowing to hide all but one message in the Rmail file. The message that is shown is called the *current message*. Rmail mode's special commands can do such things as delete the current message, copy it into another file, send a reply, or move to another message. You can also create multiple Rmail files and use Rmail to move messages between them.

Within the Rmail file, messages are normally arranged sequentially in order of receipt; you can specify other ways to sort them. Messages are assigned consecutive integers as their *message numbers*. The number of the current message is displayed in Rmail's mode line, followed by the total number of messages in the file. You can move to a message by specifying its message number with the `j` key (see Section 27.3 [Rmail Motion], page 368).

Following the usual conventions of Emacs, changes in an Rmail file become permanent only when the file is saved. You can save it with `s` (`rmail-save`), which also expunges deleted messages from the file first (see Section 27.4 [Rmail Deletion], page 369). To save the file without expunging, use `C-x C-s`. Rmail also saves the Rmail file after merging new mail from an inbox file (see Section 27.5 [Rmail Inbox], page 370).

You can exit Rmail with `q` (`rmail-quit`); this expunges and saves the Rmail file and then switches to another buffer. But there is no need to "exit" formally. If you switch from Rmail to editing in other buffers, and never happen to switch back, you have exited. (The Rmail command `b`, `rmail-bury`, does this for you.) Just make sure to save the Rmail file eventually (like any other file you have changed). `C-x s` is a good enough way to do this (see Section 14.3 [Saving], page 148).

27.2 Scrolling Within a Message

When Rmail displays a message that does not fit on the screen, you must scroll through it to read the rest. You could do this with `C-v`, `M-v` and `M-<`, but in Rmail scrolling is so frequent that it deserves to be easier to type.

⟨SPC⟩ Scroll forward (`scroll-up`).

⟨DEL⟩ Scroll backward (`scroll-down`).

`.` Scroll to start of message (`rmail-beginning-of-message`).

Since the most common thing to do while reading a message is to scroll through it by screenfuls, Rmail makes ⟨SPC⟩ and ⟨DEL⟩ synonyms of `C-v` (`scroll-up`) and `M-v` (`scroll-down`)

The command `.` (`rmail-beginning-of-message`) scrolls back to the beginning of the selected message. This is not quite the same as `M-<`: for one thing, it does not set the mark; for another, it resets the buffer boundaries to the current message if you have changed them.

27.3 Moving Among Messages

The most basic thing to do with a message is to read it. The way to do this in Rmail is to make the message current. The usual practice is to move sequentially through the file, since this is the order of receipt of messages. When you enter Rmail, you are positioned at the first message that you have not yet made current (that is, the first one that has the '`unseen`' attribute; see Section 27.9 [Rmail Attributes], page 375). Move forward to see the other new messages; move backward to reexamine old messages.

`n` Move to the next nondeleted message, skipping any intervening deleted messages (`rmail-next-undeleted-message`).

`p` Move to the previous nondeleted message (`rmail-previous-undeleted-message`).

`M-n` Move to the next message, including deleted messages (`rmail-next-message`).

`M-p` Move to the previous message, including deleted messages (`rmail-previous-message`).

`j` Move to the first message. With argument *n*, move to message number *n* (`rmail-show-message`).

`>` Move to the last message (`rmail-last-message`).

`<` Move to the first message (`rmail-first-message`).

`M-s` *regexp* ⟨RET⟩
: Move to the next message containing a match for *regexp* (`rmail-search`).

`- M-s` *regexp* ⟨RET⟩
: Move to the previous message containing a match for *regexp*.

`n` and `p` are the usual way of moving among messages in Rmail. They move through the messages sequentially, but skip over deleted messages, which is usually what you want to do. Their command definitions are named `rmail-next-undeleted-message` and `rmail-previous-undeleted-message`. If you do not want to skip deleted messages—for example, if you want to move to a message to undelete it—use the variants `M-n` and `M-p` (`rmail-next-message` and `rmail-previous-message`). A numeric argument to any of these commands serves as a repeat count.

In Rmail, you can specify a numeric argument by typing just the digits. You don't need to type `C-u` first.

The `M-s` (`rmail-search`) command is Rmail's version of search. The usual incremental search command `C-s` works in Rmail, but it searches only within the current message. The purpose of `M-s` is to search for another message. It reads a regular expression (see Section 12.5 [Regexps], page 125) nonincrementally, then searches starting at the beginning of the following message for a match. It then selects that message. If *regexp* is empty, `M-s` reuses the regexp used the previous time.

To search backward in the file for another message, give `M-s` a negative argument. In Rmail you can do this with `- M-s`.

It is also possible to search for a message based on labels. See Section 27.8 [Rmail Labels], page 374.

To move to a message specified by absolute message number, use `j` (`rmail-show-message`) with the message number as argument. With no argument, `j` selects the first message. `<` (`rmail-first-message`) also selects the first message. `>` (`rmail-last-message`) selects the last message.

27.4 Deleting Messages

When you no longer need to keep a message, you can *delete* it. This flags it as ignorable, and some Rmail commands pretend it is no longer present; but it still has its place in the Rmail file, and still has its message number.

Expunging the Rmail file actually removes the deleted messages. The remaining messages are renumbered consecutively. Expunging is the only action that changes the message number of any message, except for undigestifying (see Section 27.16 [Rmail Digest], page 383).

`d`
: Delete the current message, and move to the next nondeleted message (`rmail-delete-forward`).

`C-d` Delete the current message, and move to the previous nondeleted message (`rmail-delete-backward`).

`u` Undelete the current message, or move back to a deleted message and undelete it (`rmail-undelete-previous-message`).

`x` Expunge the Rmail file (`rmail-expunge`).

There are two Rmail commands for deleting messages. Both delete the current message and select another message. `d` (`rmail-delete-forward`) moves to the following message, skipping messages already deleted, while `C-d` (`rmail-delete-backward`) moves to the previous nondeleted message. If there is no nondeleted message to move to in the specified direction, the message that was just deleted remains current. A numeric argument to either command reverses the direction of motion after deletion.

Whenever Rmail deletes a message, it invokes the function(s) listed in `rmail-delete-message-hook`. When the hook functions are invoked, the message has been marked deleted, but it is still the current message in the Rmail buffer.

To make all the deleted messages finally vanish from the Rmail file, type `x` (`rmail-expunge`). Until you do this, you can still *undelete* the deleted messages. The undeletion command, `u` (`rmail-undelete-previous-message`), is designed to cancel the effect of a `d` command in most cases. It undeletes the current message if the current message is deleted. Otherwise it moves backward to previous messages until a deleted message is found, and undeletes that message.

You can usually undo a `d` with a `u` because the `u` moves back to and undeletes the message that the `d` deleted. But this does not work when the `d` skips a few already-deleted messages that follow the message being deleted; then the `u` command undeletes the last of the messages that were skipped. There is no clean way to avoid this problem. However, by repeating the `u` command, you can eventually get back to the message that you intend to undelete. You can also select a particular deleted message with the `M-p` command, then type `u` to undelete it.

A deleted message has the '`deleted`' attribute, and as a result '`deleted`' appears in the mode line when the current message is deleted. In fact, deleting or undeleting a message is nothing more than adding or removing this attribute. See Section 27.9 [Rmail Attributes], page 375.

27.5 Rmail Files and Inboxes

The operating system places incoming mail for you in a file that we call your *inbox*. When you start up Rmail, it runs a C program called `movemail` to copy the new messages from your inbox into your primary Rmail file, which also contains other messages saved from previous Rmail sessions. It

is in this file that you actually read the mail with Rmail. This operation is called *getting new mail*. You can get new mail at any time in Rmail by typing `g`.

The variable `rmail-primary-inbox-list` contains a list of the files which are inboxes for your primary Rmail file. If you don't set this variable explicitly, it is initialized from the `MAIL` environment variable, or, as a last resort, set to `nil`, which means to use the default inbox. The default inbox is '`/var/mail/`*username*', '`/usr/spool/mail/`*username*', or '`/usr/mail/`*username*', depending on your operating system.

To see what the default is on your system, use `C-h v rmail-primary-inbox` ⟨RET⟩. You can specify the inbox file(s) for any Rmail file with the command `set-rmail-inbox-list`; see Section 27.6 [Rmail Files], page 371.

There are two reasons for having separate Rmail files and inboxes.

1. The inbox file format varies between operating systems and according to the other mail software in use. Only one part of Rmail needs to know about the alternatives, and it need only understand how to convert all of them to Rmail's own format.
2. It is very cumbersome to access an inbox file without danger of losing mail, because it is necessary to interlock with mail delivery. Moreover, different operating systems use different interlocking techniques. The strategy of moving mail out of the inbox once and for all into a separate Rmail file avoids the need for interlocking in all the rest of Rmail, since only Rmail operates on the Rmail file.

Rmail was written to use Babyl format as its internal format. Since then, we have recognized that the usual inbox format on Unix and GNU systems is adequate for the job, and we plan to change Rmail to use that as its internal format. However, the Rmail file will still be separate from the inbox file, even on systems where their format is the same.

27.6 Multiple Rmail Files

Rmail operates by default on your *primary Rmail file*, which is named '`~/RMAIL`' and receives your incoming mail from your system inbox file. But you can also have other Rmail files and edit them with Rmail. These files can receive mail through their own inboxes, or you can move messages into them with explicit Rmail commands (see Section 27.7 [Rmail Output], page 373).

`i` *file* ⟨RET⟩
: Read *file* into Emacs and run Rmail on it (`rmail-input`).

`M-x set-rmail-inbox-list` ⟨RET⟩ *files* ⟨RET⟩
: Specify inbox file names for current Rmail file to get mail from.

`g`
: Merge new mail from current Rmail file's inboxes (`rmail-get-new-mail`).

`C-u g` *file* ⟨RET⟩
: Merge new mail from inbox file *file*.

To run Rmail on a file other than your primary Rmail file, you may use the `i` (`rmail-input`) command in Rmail. This visits the file in Rmail mode. You can use `M-x rmail-input` even when not in Rmail.

The file you read with `i` should normally be a valid Rmail file. If it is not, Rmail tries to decompose it into a stream of messages in various known formats. If it succeeds, it converts the whole file to an Rmail file. If you specify a file name that doesn't exist, `i` initializes a new buffer for creating a new Rmail file.

You can also select an Rmail file from a menu. Choose first the menu bar Classify item, then from the Classify menu choose the Input Rmail File item; then choose the Rmail file you want. The variables `rmail-secondary-file-directory` and `rmail-secondary-file-regexp` specify which files to offer in the menu: the first variable says which directory to find them in; the second says which files in that directory to offer (all those that match the regular expression). These variables also apply to choosing a file for output (see Section 27.7 [Rmail Output], page 373).

Each Rmail file can contain a list of inbox file names; you can specify this list with `M-x set-rmail-inbox-list` ⟨RET⟩ *files* ⟨RET⟩. The argument can contain any number of file names, separated by commas. It can also be empty, which specifies that this file should have no inboxes. Once a list of inboxes is specified, the Rmail file remembers it permanently until you specify a different list.

As a special exception, if your primary Rmail file does not specify any inbox files, it uses your standard system inbox.

The `g` command (`rmail-get-new-mail`) merges mail into the current Rmail file from its specified inboxes. If the Rmail file has no inboxes, `g` does nothing. The command `M-x rmail` also merges new mail into your primary Rmail file.

To merge mail from a file that is not the usual inbox, give the `g` key a numeric argument, as in `C-u g`. Then it reads a file name and merges mail from that file. The inbox file is not deleted or changed in any way when `g` with an argument is used. This is, therefore, a general way of merging one file of messages into another.

27.7 Copying Messages Out to Files

These commands copy messages from an Rmail file into another file.

`o` *file* ⟨RET⟩
: Append a copy of the current message to the file *file*, using Rmail file format by default (`rmail-output-to-rmail-file`).

`C-o` *file* ⟨RET⟩
: Append a copy of the current message to the file *file*, using system inbox file format by default (`rmail-output`).

`w` *file* ⟨RET⟩
: Output just the message body to the file *file*, taking the default file name from the message '`Subject`' header.

The commands `o` and `C-o` copy the current message into a specified file. This file may be an Rmail file or it may be in system inbox format; the output commands ascertain the file's format and write the copied message in that format.

When copying a message to a file in Unix mail file format, these commands include whichever header fields are currently visible. Use the `t` command first, if you wish, to specify which headers to show (and copy).

The `o` and `C-o` commands differ in two ways: each has its own separate default file name, and each specifies a choice of format to use when the file does not already exist. The `o` command uses Rmail format when it creates a new file, while `C-o` uses system inbox format for a new file. The default file name for `o` is the file name used last with `o`, and the default file name for `C-o` is the file name used last with `C-o`.

If the output file is an Rmail file currently visited in an Emacs buffer, the output commands copy the message into that buffer. It is up to you to save the buffer eventually in its file.

Sometimes you may receive a message whose body holds the contents of a file. You can save the body to a file (excluding the message header) with the `w` command (`rmail-output-body-to-file`). Often these messages contain the intended file name in the '`Subject`' field, so the `w` command uses the '`Subject`' field as the default for the output file name. However, the file name is read using the minibuffer, so you can specify a different name if you wish.

You can also output a message to an Rmail file chosen with a menu. Choose first the menu bar Classify item, then from the Classify menu choose the Output Rmail File menu item; then choose the Rmail file you want. This outputs the current message to that file, like the `o` command. The variables `rmail-secondary-file-directory` and `rmail-secondary-file-regexp` specify which files to offer in the menu: the first variable says which directory to find them in; the second says which files in that directory to offer (all those that match the regular expression).

Copying a message gives the original copy of the message the '`filed`' attribute, so that '`filed`' appears in the mode line when such a message is current. If you like to keep just a single copy of every mail message, set the variable `rmail-delete-after-output` to `t`; then the `o` and `C-o` commands delete the original message after copying it. (You can undelete the original afterward if you wish.)

Copying messages into files in system inbox format uses the header fields that are displayed in Rmail at the time. Thus, if you use the `t` command to view the entire header and then copy the message, the entire header is copied. See Section 27.13 [Rmail Display], page 381.

The variable `rmail-output-file-alist` lets you specify intelligent defaults for the output file, based on the contents of the current message. The value should be a list whose elements have this form:

(*regexp* . *name-exp*)

If there's a match for *regexp* in the current message, then the default file name for output is *name-exp*. If multiple elements match the message, the first matching element decides the default file name. The subexpression *name-exp* may be a string constant giving the file name to use, or more generally it may be any Lisp expression that returns a file name as a string. `rmail-output-file-alist` applies to both `o` and `C-o`.

27.8 Labels

Each message can have various *labels* assigned to it as a means of classification. Each label has a name; different names are different labels. Any given label is either present or absent on a particular message. A few label names have standard meanings and are given to messages automatically by Rmail when appropriate; these special labels are called *attributes*. All other labels are assigned only by users.

`a` *label* ⟨RET⟩
: Assign the label *label* to the current message (`rmail-add-label`).

`k` *label* ⟨RET⟩
: Remove the label *label* from the current message (`rmail-kill-label`).

`C-M-n` *labels* ⟨RET⟩
: Move to the next message that has one of the labels *labels* (`rmail-next-labeled-message`).

`C-M-p` *labels* ⟨RET⟩
: Move to the previous message that has one of the labels *labels* (`rmail-previous-labeled-message`).

`C-M-l` *labels* ⟨RET⟩
: Make a summary of all messages containing any of the labels *labels* (`rmail-summary-by-labels`).

The `a` (`rmail-add-label`) and `k` (`rmail-kill-label`) commands allow you to assign or remove any label on the current message. If the *label*

argument is empty, it means to assign or remove the same label most recently assigned or removed.

Once you have given messages labels to classify them as you wish, there are two ways to use the labels: in moving and in summaries.

The command `C-M-n` *labels* ⟨RET⟩ (`rmail-next-labeled-message`) moves to the next message that has one of the labels *labels*. The argument *labels* specifies one or more label names, separated by commas. `C-M-p` (`rmail-previous-labeled-message`) is similar, but moves backwards to previous messages. A numeric argument to either command serves as a repeat count.

The command `C-M-l` *labels* ⟨RET⟩ (`rmail-summary-by-labels`) displays a summary containing only the messages that have at least one of a specified set of labels. The argument *labels* is one or more label names, separated by commas. See Section 27.11 [Rmail Summary], page 378, for information on summaries.

If the *labels* argument to `C-M-n`, `C-M-p` or `C-M-l` is empty, it means to use the last set of labels specified for any of these commands.

27.9 Rmail Attributes

Some labels such as '`deleted`' and '`filed`' have built-in meanings and are assigned to or removed from messages automatically at appropriate times; these labels are called *attributes*. Here is a list of Rmail attributes:

'`unseen`' Means the message has never been current. Assigned to messages when they come from an inbox file, and removed when a message is made current. When you start Rmail, it initially shows the first message that has this attribute.

'`deleted`' Means the message is deleted. Assigned by deletion commands and removed by undeletion commands (see Section 27.4 [Rmail Deletion], page 369).

'`filed`' Means the message has been copied to some other file. Assigned by the file output commands (see Section 27.6 [Rmail Files], page 371).

'`answered`'
Means you have mailed an answer to the message. Assigned by the `r` command (`rmail-reply`). See Section 27.10 [Rmail Reply], page 376.

'`forwarded`'
Means you have forwarded the message. Assigned by the `f` command (`rmail-forward`). See Section 27.10 [Rmail Reply], page 376.

‘edited’ Means you have edited the text of the message within Rmail. See Section 27.15 [Rmail Editing], page 383.

‘resent’ Means you have resent the message. Assigned by the command M-x rmail-resend. See Section 27.10 [Rmail Reply], page 376.

All other labels are assigned or removed only by the user, and have no standard meaning.

27.10 Sending Replies

Rmail has several commands that use Mail mode to send outgoing mail. See Chapter 26 [Sending Mail], page 357, for information on using Mail mode, including certain features meant to work with Rmail. What this section documents are the special commands of Rmail for entering Mail mode. Note that the usual keys for sending mail—C-x m, C-x 4 m, and C-x 5 m—are available in Rmail mode and work just as they usually do.

m Send a message (rmail-mail).

c Continue editing the already started outgoing message (rmail-continue).

r Send a reply to the current Rmail message (rmail-reply).

f Forward the current message to other users (rmail-forward).

C-u f Resend the current message to other users (rmail-resend).

M-m Try sending a bounced message a second time (rmail-retry-failure).

The most common reason to send a message while in Rmail is to reply to the message you are reading. To do this, type r (rmail-reply). This displays the ‘*mail*’ buffer in another window, much like C-x 4 m, but preinitializes the ‘Subject’, ‘To’, ‘CC’ and ‘In-reply-to’ header fields based on the message you are replying to. The ‘To’ field starts out as the address of the person who sent the message you received, and the ‘CC’ field starts out with all the other recipients of that message.

You can exclude certain recipients from being placed automatically in the ‘CC’, using the variable rmail-dont-reply-to-names. Its value should be a regular expression (as a string); any recipient that the regular expression matches, is excluded from the ‘CC’ field. The default value matches your own name, and any name starting with ‘info-’. (Those names are excluded because there is a convention of using them for large mailing lists to broadcast announcements.)

To omit the ‘CC’ field completely for a particular reply, enter the reply command with a numeric argument: C-u r or 1 r.

Once the '*mail*' buffer has been initialized, editing and sending the mail goes as usual (see Chapter 26 [Sending Mail], page 357). You can edit the presupplied header fields if they are not right for you. You can also use the commands of Mail mode (see Section 26.4 [Mail Mode], page 362), including C-c C-y which yanks in the message that you are replying to. You can switch to the Rmail buffer, select a different message there, switch back, and yank the new current message.

Sometimes a message does not reach its destination. Mailers usually send the failed message back to you, enclosed in a *failure message*. The Rmail command M-m (rmail-retry-failure) prepares to send the same message a second time: it sets up a '*mail*' buffer with the same text and header fields as before. If you type C-c C-c right away, you send the message again exactly the same as the first time. Alternatively, you can edit the text or headers and then send it. The variable rmail-retry-ignored-headers, in the same format as rmail-ignored-headers (see Section 27.13 [Rmail Display], page 381), controls which headers are stripped from the failed message when retrying it; it defaults to nil.

Another frequent reason to send mail in Rmail is to *forward* the current message to other users. f (rmail-forward) makes this easy by preinitializing the '*mail*' buffer with the current message as the text, and a subject designating a forwarded message. All you have to do is fill in the recipients and send. When you forward a message, recipients get a message which is "from" you, and which has the original message in its contents.

Forwarding a message encloses it between two delimiter lines. It also modifies every line that starts with a dash, by inserting '- ' at the start of the line. When you receive a forwarded message, if it contains something besides ordinary text—for example, program source code—you might find it useful to undo that transformation. You can do this by selecting the forwarded message and typing M-x unforward-rmail-message. This command extracts the original forwarded message, deleting the inserted '- ' strings, and inserts it into the Rmail file as a separate message immediately following the current one.

Resending is an alternative similar to forwarding; the difference is that resending sends a message that is "from" the original sender, just as it reached you—with a few added header fields 'Resent-from' and 'Resent-to' to indicate that it came via you. To resend a message in Rmail, use C-u f. (f runs rmail-forward, which is programmed to invoke rmail-resend if you provide a numeric argument.)

The m (rmail-mail) command is used to start editing an outgoing message that is not a reply. It leaves the header fields empty. Its only difference from C-x 4 m is that it makes the Rmail buffer accessible for C-c C-y, just as r does. Thus, m can be used to reply to or forward a message; it can do anything r or f can do.

The `c` (`rmail-continue`) command resumes editing the '`*mail*`' buffer, to finish editing an outgoing message you were already composing, or to alter a message you have sent.

If you set the variable `rmail-mail-new-frame` to a non-`nil` value, then all the Rmail commands to start sending a message create a new frame to edit it in. This frame is deleted when you send the message, or when you use the '`Don't Send`' item in the '`Mail`' menu.

All the Rmail commands to send a message use the mail-composition method that you have chosen (see Section 26.6 [Mail Methods], page 366).

27.11 Summaries

A *summary* is a buffer containing one line per message to give you an overview of the mail in an Rmail file. Each line shows the message number, the sender, the labels, and the subject. Almost all Rmail commands are valid in the summary buffer also; these apply to the message described by the current line of the summary. Moving point in the summary buffer selects messages as you move to their summary lines.

A summary buffer applies to a single Rmail file only; if you are editing multiple Rmail files, each one can have its own summary buffer. The summary buffer name is made by appending '`-summary`' to the Rmail buffer's name. Normally only one summary buffer is displayed at a time.

27.11.1 Making Summaries

Here are the commands to create a summary for the current Rmail file. Once the Rmail file has a summary buffer, changes in the Rmail file (such as deleting or expunging messages, and getting new mail) automatically update the summary.

`h`
`C-M-h` Summarize all messages (`rmail-summary`).

`l` *labels* ⟨RET⟩
`C-M-l` *labels* ⟨RET⟩
Summarize messages that have one or more of the specified labels (`rmail-summary-by-labels`).

`C-M-r` *rcpts* ⟨RET⟩
Summarize messages that have one or more of the specified recipients (`rmail-summary-by-recipients`).

`C-M-t` *topic* ⟨RET⟩
Summarize messages that have a match for the specified regexp *topic* in their subjects (`rmail-summary-by-topic`).

The `h` or `C-M-h` (`rmail-summary`) command fills the summary buffer for the current Rmail file with a summary of all the messages in the file. It then displays and selects the summary buffer in another window.

`C-M-l` *labels* ⟨RET⟩ (`rmail-summary-by-labels`) makes a partial summary mentioning only the messages that have one or more of the labels *labels*. *labels* should contain label names separated by commas.

`C-M-r` *rcpts* ⟨RET⟩ (`rmail-summary-by-recipients`) makes a partial summary mentioning only the messages that have one or more of the recipients *rcpts*. *rcpts* should contain mailing addresses separated by commas.

`C-M-t` *topic* ⟨RET⟩ (`rmail-summary-by-topic`) makes a partial summary mentioning only the messages whose subjects have a match for the regular expression *topic*.

Note that there is only one summary buffer for any Rmail file; making one kind of summary discards any previously made summary.

The variable `rmail-summary-window-size` says how many lines to use for the summary window. The variable `rmail-summary-line-count-flag` controls whether the summary line for a message should include the line count of the message.

27.11.2 Editing in Summaries

You can use the Rmail summary buffer to do almost anything you can do in the Rmail buffer itself. In fact, once you have a summary buffer, there's no need to switch back to the Rmail buffer.

You can select and display various messages in the Rmail buffer, from the summary buffer, just by moving point in the summary buffer to different lines. It doesn't matter what Emacs command you use to move point; whichever line point is on at the end of the command, that message is selected in the Rmail buffer.

Almost all Rmail commands work in the summary buffer as well as in the Rmail buffer. Thus, `d` in the summary buffer deletes the current message, `u` undeletes, and `x` expunges. `o` and `C-o` output the current message to a file; `r` starts a reply to it. You can scroll the current message while remaining in the summary buffer using ⟨SPC⟩ and ⟨DEL⟩.

The Rmail commands to move between messages also work in the summary buffer, but with a twist: they move through the set of messages included in the summary. They also ensure the Rmail buffer appears on the screen (unlike cursor motion commands, which update the contents of the Rmail buffer but don't display it in a window unless it already appears). Here is a list of these commands:

`n` Move to next line, skipping lines saying 'deleted', and select its message.

`p` Move to previous line, skipping lines saying 'deleted', and select its message.

`M-n` Move to next line and select its message.

`M-p` Move to previous line and select its message.

`>` Move to the last line, and select its message.

`<` Move to the first line, and select its message.

`M-s` *pattern* ⟨RET⟩
Search through messages for *pattern* starting with the current message; select the message found, and move point in the summary buffer to that message's line.

Deletion, undeletion, and getting new mail, and even selection of a different message all update the summary buffer when you do them in the Rmail buffer. If the variable `rmail-redisplay-summary` is non-`nil`, these actions also bring the summary buffer back onto the screen.

When you are finished using the summary, type `Q` (`rmail-summary-wipe`) to delete the summary buffer's window. You can also exit Rmail while in the summary: `q` (`rmail-summary-quit`) deletes the summary window, then exits from Rmail by saving the Rmail file and switching to another buffer.

27.12 Sorting the Rmail File

`M-x rmail-sort-by-date`
Sort messages of current Rmail file by date.

`M-x rmail-sort-by-subject`
Sort messages of current Rmail file by subject.

`M-x rmail-sort-by-author`
Sort messages of current Rmail file by author's name.

`M-x rmail-sort-by-recipient`
Sort messages of current Rmail file by recipient's names.

`M-x rmail-sort-by-correspondent`
Sort messages of current Rmail file by the name of the other correspondent.

`M-x rmail-sort-by-lines`
Sort messages of current Rmail file by size (number of lines).

`M-x rmail-sort-by-keywords` ⟨RET⟩ *labels* ⟨RET⟩
Sort messages of current Rmail file by labels. The argument *labels* should be a comma-separated list of labels. The order of these labels specifies the order of messages; messages with

the first label come first, messages with the second label come second, and so on. Messages which have none of these labels come last.

The Rmail sort commands perform a *stable sort*: if there is no reason to prefer either one of two messages, their order remains unchanged. You can use this to sort by more than one criterion. For example, if you use `rmail-sort-by-date` and then `rmail-sort-by-author`, messages from the same author appear in order by date.

With a numeric argument, all these commands reverse the order of comparison. This means they sort messages from newest to oldest, from biggest to smallest, or in reverse alphabetical order.

27.13 Display of Messages

Rmail reformats the header of each message before displaying it for the first time. Reformatting hides uninteresting header fields to reduce clutter. You can use the `t` command to show the entire header or to repeat the header reformatting operation.

`t` Toggle display of complete header (`rmail-toggle-header`).

Reformatting the header involves deleting most header fields, on the grounds that they are not interesting. The variable `rmail-ignored-headers` holds a regular expression that specifies which header fields to hide in this way—if it matches the beginning of a header field, that whole field is hidden.

Rmail saves the complete original header before reformatting; to see it, use the `t` command (`rmail-toggle-header`). This discards the reformatted headers of the current message and displays it with the original header. Repeating `t` reformats the message again. Selecting the message again also reformats.

One consequence of this is that if you edit the reformatted header (using `e`; see Section 27.15 [Rmail Editing], page 383), subsequent use of `t` will discard your edits. On the other hand, if you use `e` after `t`, to edit the original (unreformatted) header, those changes are permanent.

When the `t` command has a prefix argument, a positive argument means to show the reformatted header, and a zero or negative argument means to show the full header.

When the terminal supports multiple fonts or colors, Rmail highlights certain header fields that are especially interesting—by default, the '`From`' and '`Subject`' fields. The variable `rmail-highlighted-headers` holds a regular expression that specifies the header fields to highlight; if it matches the beginning of a header field, that whole field is highlighted.

If you specify unusual colors for your text foreground and background, the colors used for highlighting may not go well with them. If so, specify different

colors for the `highlight` face. That is worth doing because the `highlight` face is used for other kinds of highlighting as well. See Section 11.1 [Faces], page 103, for how to do this.

To turn off highlighting entirely in Rmail, set `rmail-highlighted-headers` to `nil`.

You can highlight and activate URLs in incoming messages by adding the function `goto-address` to the hook `rmail-show-message-hook`. Then you can browse these URLs by clicking on them with `Mouse-2` or by moving to one and typing `C-c` ⟨RET⟩. See Section 30.15.2 [Goto-address], page 451.

27.14 Rmail and Coding Systems

Rmail automatically decodes messages which contain non-ASCII characters, just as it does with files you visit and with and subprocess output. Rmail uses the standard '`charset=`*charset*' header in the message to determine how the was message encoded by the sender. It maps *charset* into the corresponding Emacs coding system (see Section 18.6 [Coding Systems], page 223), and uses that coding system to decode message text. If the message header doesn't have the charset specification, or if the *charset* it specifies is not recognized, Rmail chooses the coding system with the usual Emacs heuristics and defaults (see Section 18.7 [Recognize Coding], page 225).

Occasionally, a message is decoded incorrectly, either because Emacs guessed the wrong coding system in the absence of the '`charset`' specification, or because the specification was inaccurate. For example, a misconfigured mailer could send a message with a '`charset=iso-8859-1`' header when the message is actually encoded in `koi8-r`. When you see the message text garbled, or some of its characters displayed as empty boxes, this may have happened.

You can correct the problem by decoding the message again using the right coding system, if you can figure out or guess which one is right. To do this, invoke the `M-x rmail-redecode-body` command. It reads the name of a coding system, encodes the message body using whichever coding system was used to decode it before, then redecodes it using the coding system you specified. If you specified the right coding system, the result should be readable.

Decoding and encoding using the wrong coding system is lossless for most encodings, in particular with 8-bit encodings such as iso-8859 or koi8. So, if the initial attempt to redecode the message didn't result in a legible text, you can try other coding systems until you succeed.

With some coding systems, notably those from the iso-2022 family, information can be lost in decoding, so that encoding the message again won't bring back the original incoming text. In such a case, `rmail-redecode-body` cannot work. However, the problems that call for use of `rmail-redecode-`

`body` rarely occur with those coding systems. So in practice the command works when you need it.

27.15 Editing Within a Message

Most of the usual Emacs commands are available in Rmail mode, though a few, such as `C-M-n` and `C-M-h`, are redefined by Rmail for other purposes. However, the Rmail buffer is normally read only, and most of the letters are redefined as Rmail commands. If you want to edit the text of a message, you must use the Rmail command `e`.

`e` Edit the current message as ordinary text.

The `e` command (`rmail-edit-current-message`) switches from Rmail mode into Rmail Edit mode, another major mode which is nearly the same as Text mode. The mode line indicates this change.

In Rmail Edit mode, letters insert themselves as usual and the Rmail commands are not available. When you are finished editing the message and are ready to go back to Rmail, type `C-c C-c`, which switches back to Rmail mode. Alternatively, you can return to Rmail mode but cancel all the editing that you have done, by typing `C-c C-]`.

Entering Rmail Edit mode runs the hook `text-mode-hook`; then it runs the hook `rmail-edit-mode-hook` (see Section 31.2.3 [Hooks], page 465). It adds the attribute '`edited`' to the message. It also displays the full headers of the message, so that you can edit the headers as well as the body of the message, and your changes in the headers will be permanent.

27.16 Digest Messages

A *digest message* is a message which exists to contain and carry several other messages. Digests are used on some moderated mailing lists; all the messages that arrive for the list during a period of time such as one day are put inside a single digest which is then sent to the subscribers. Transmitting the single digest uses much less computer time than transmitting the individual messages even though the total size is the same, because the per-message overhead in network mail transmission is considerable.

When you receive a digest message, the most convenient way to read it is to *undigestify* it: to turn it back into many individual messages. Then you can read and delete the individual messages as it suits you.

To do this, select the digest message and type the command `M-x undigestify-rmail-message`. This extracts the submessages as separate Rmail messages, and inserts them following the digest. The digest message itself is flagged as deleted.

27.17 Converting an Rmail File to Inbox Format

The command `M-x unrmail` converts a file in Rmail format to inbox format (also known as the system mailbox format), so that you can use it with other mail-editing tools. You must specify two arguments, the name of the Rmail file and the name to use for the converted file. `M-x unrmail` does not alter the Rmail file itself.

27.18 Reading Rot13 Messages

Mailing list messages that might offend some readers are sometimes encoded in a simple code called *rot13*—so named because it rotates the alphabet by 13 letters. This code is not for secrecy, as it provides none; rather, it enables those who might be offended to avoid ever seeing the real text of the message.

To view a buffer using the rot13 code, use the command `M-x rot13-other-window`. This displays the current buffer in another window which applies the code when displaying the text.

27.19 `movemail` and POP

When getting new mail, Rmail first copies the new mail from the inbox file to the Rmail file; then it saves the Rmail file; then it truncates the inbox file. This way, a system crash may cause duplication of mail between the inbox and the Rmail file, but cannot lose mail. If `rmail-preserve-inbox` is non-`nil`, then Rmail will copy new mail from the inbox file to the Rmail file without truncating the inbox file. You may wish to set this, for example, on a portable computer you use to check your mail via POP while traveling, so that your mail will remain on the server and you can save it later on your workstation.

In some cases, Rmail copies the new mail from the inbox file indirectly. First it runs the `movemail` program to move the mail from the inbox to an intermediate file called '`~/.newmail-`*inboxname*'. Then Rmail merges the new mail from that file, saves the Rmail file, and only then deletes the intermediate file. If there is a crash at the wrong time, this file continues to exist, and Rmail will use it again the next time it gets new mail from that inbox.

If Rmail is unable to convert the data in '`~/.newmail-`*inboxname*' into Babyl format, it renames the file to '`~/RMAILOSE.`*n*' (*n* is an integer chosen to make the name unique) so that Rmail will not have trouble with the data again. You should look at the file, find whatever message confuses

Rmail (probably one that includes the control-underscore character, octal code 037), and delete it. Then you can use `1 g` to get new mail from the corrected file.

Some sites use a method called POP for accessing users' inbox data instead of storing the data in inbox files. `movemail` can work with POP if you compile it with the macro `MAIL_USE_POP` defined. (You can achieve that by specifying '`--with-pop`' when you run `configure` during the installation of Emacs.) `movemail` only works with POP3, not with older versions of POP.

Assuming you have compiled and installed `movemail` appropriately, you can specify a POP inbox by using a "file name" of the form '`po:`*username*', in the inbox list of an Rmail file. `movemail` handles such a name by opening a connection to the POP server. The `MAILHOST` environment variable specifies the machine to look for the server on; alternatively, you can specify the POP server host name as part of the mailbox name using the syntax '`po:`*username*`:`*hostname*'.

Accessing mail via POP may require a password. If the variable `rmail-pop-password` is non-`nil`, it specifies the password to use for POP. Alternatively, if `rmail-pop-password-required` is non-`nil`, then Rmail asks you for the password to use.

If you need to pass additional command-line flags to `movemail`, set the variable `rmail-movemail-flags` a list of the flags you wish to use. Do not use this variable to pass the '`-p`' flag to preserve your inbox contents; use `rmail-preserve-inbox` instead.

The `movemail` program installed at your site may support Kerberos authentication. If it is supported, it is used by default whenever you attempt to retrieve POP mail when `rmail-pop-password` and `rmail-pop-password-required` are unset.

Some POP servers store messages in reverse order. If your server does this, and you would rather read your mail in the order in which it was received, you can tell `movemail` to reverse the order of downloaded messages by adding the '`-r`' flag to `rmail-movemail-flags`.

28 Dired, the Directory Editor

Dired makes an Emacs buffer containing a listing of a directory, and optionally some of its subdirectories as well. You can use the normal Emacs commands to move around in this buffer, and special Dired commands to operate on the files listed.

The Dired-X package provides various extra features for Dired mode. See section “” in *Dired Extra Version 2 User's Manual*.

28.1 Entering Dired

To invoke Dired, do `C-x d` or `M-x dired`. The command reads a directory name or wildcard file name pattern as a minibuffer argument to specify which files to list. Where `dired` differs from `list-directory` is in putting the buffer into Dired mode so that the special commands of Dired are available.

The variable `dired-listing-switches` specifies the options to give to `ls` for listing directory; this string *must* contain ‘`-l`’. If you use a numeric prefix argument with the `dired` command, you can specify the `ls` switches with the minibuffer before you enter the directory specification.

To display the Dired buffer in another window rather than in the selected window, use `C-x 4 d` (`dired-other-window`) instead of `C-x d`. `C-x 5 d` (`dired-other-frame`) uses a separate frame to display the Dired buffer.

28.2 Commands in the Dired Buffer

The Dired buffer is “read-only,” and inserting text in it is not useful, so ordinary printing characters such as `d` and `x` are used for special Dired commands. Some Dired commands *mark* or *flag* the *current file* (that is, the file on the current line); other commands operate on the marked files or on the flagged files.

All the usual Emacs cursor motion commands are available in Dired buffers. Some special-purpose cursor motion commands are also provided. The keys `C-n` and `C-p` are redefined to put the cursor at the beginning of the file name on the line, rather than at the beginning of the line.

For extra convenience, ⟨SPC⟩ and `n` in Dired are equivalent to `C-n`. `p` is equivalent to `C-p`. (Moving by lines is so common in Dired that it deserves to be easy to type.) ⟨DEL⟩ (move up and unflag) is often useful simply for moving up.

28.3 Deleting Files with Dired

The primary use of Dired is to *flag* files for deletion and then delete the files previously flagged.

`d` Flag this file for deletion.

`u` Remove deletion flag on this line.

⟨DEL⟩ Move point to previous line and remove the deletion flag on that line.

`x` Delete the files that are flagged for deletion.

You can flag a file for deletion by moving to the line describing the file and typing `d` (`dired-flag-file-deletion`). The deletion flag is visible as a '`D`' at the beginning of the line. This command moves point to the next line, so that repeated `d` commands flag successive files. A numeric argument serves as a repeat count.

The variable `dired-recursive-deletes` controls whether the delete command will delete non-empty directories (including their contents). The default is to delete only empty directories.

The files are flagged for deletion rather than deleted immediately to reduce the danger of deleting a file accidentally. Until you direct Dired to expunge the flagged files, you can remove deletion flags using the commands `u` and ⟨DEL⟩. `u` (`dired-unmark`) works just like `d`, but removes flags rather than making flags. ⟨DEL⟩ (`dired-unmark-backward`) moves upward, removing flags; it is like `u` with argument −1.

To delete the flagged files, type `x` (`dired-expunge`). This command first displays a list of all the file names flagged for deletion, and requests confirmation with `yes`. If you confirm, Dired deletes the flagged files, then deletes their lines from the text of the Dired buffer. The shortened Dired buffer remains selected.

If you answer `no` or quit with `C-g` when asked to confirm, you return immediately to Dired, with the deletion flags still present in the buffer, and no files actually deleted.

28.4 Flagging Many Files at Once

`#` Flag all auto-save files (files whose names start and end with '`#`') for deletion (see Section 14.5 [Auto Save], page 158).

`~` Flag all backup files (files whose names end with '`~`') for deletion (see Section 14.3.1 [Backup], page 150).

`&` Flag for deletion all files with certain kinds of names, names that suggest you could easily create the files again.

`.` (Period) Flag excess numeric backup files for deletion. The oldest and newest few backup files of any one file are exempt; the middle ones are flagged.

`% d` *regexp* ⟨RET⟩
Flag for deletion all files whose names match the regular expression *regexp*.

The `#`, `~`, `&`, and `.` commands flag many files for deletion, based on their file names. These commands are useful precisely because they do not themselves delete any files; you can remove the deletion flags from any flagged files that you really wish to keep.

`&` (`dired-flag-garbage-files`) flags files whose names match the regular expression specified by the variable `dired-garbage-files-regexp`. By default, this matches certain files produced by TEX, and the '`.orig`' and '`.rej`' files produced by `patch`.

`#` (`dired-flag-auto-save-files`) flags for deletion all files whose names look like auto-save files (see Section 14.5 [Auto Save], page 158)—that is, files whose names begin and end with '`#`'. `~` (`dired-flag-backup-files`) flags for deletion all files whose names say they are backup files (see Section 14.3.1 [Backup], page 150)—that is, whose names end in '`~`'.

`.` (period, `dired-clean-directory`) flags just some of the backup files for deletion: all but the oldest few and newest few backups of any one file. Normally `dired-kept-versions` (**not** `kept-new-versions`; that applies only when saving) specifies the number of newest versions of each file to keep, and `kept-old-versions` specifies the number of oldest versions to keep.

Period with a positive numeric argument, as in `C-u 3 .`, specifies the number of newest versions to keep, overriding `dired-kept-versions`. A negative numeric argument overrides `kept-old-versions`, using minus the value of the argument to specify the number of oldest versions of each file to keep.

The `% d` command flags all files whose names match a specified regular expression (`dired-flag-files-regexp`). Only the non-directory part of the file name is used in matching. You can use '`^`' and '`$`' to anchor matches. You can exclude subdirectories by hiding them (see Section 28.13 [Hiding Subdirectories], page 398).

28.5 Visiting Files in Dired

There are several Dired commands for visiting or examining the files listed in the Dired buffer. All of them apply to the current line's file; if that

file is really a directory, these commands invoke Dired on that subdirectory (making a separate Dired buffer).

`f` Visit the file described on the current line, like typing `C-x C-f` and supplying that file name (`dired-find-file`). See Section 14.2 [Visiting], page 145.

⟨RET⟩ Equivalent to `f`.

`a` Like `f`, but replaces the contents of the Dired buffer with that of an alternate file or directory (`dired-find-alternate-file`).

`o` Like `f`, but uses another window to display the file's buffer (`dired-find-file-other-window`). The Dired buffer remains visible in the first window. This is like using `C-x 4 C-f` to visit the file. See Chapter 16 [Windows], page 195.

`C-o` Visit the file described on the current line, and display the buffer in another window, but do not select that window (`dired-display-file`).

`Mouse-2` Visit the file named by the line you click on (`dired-mouse-find-file-other-window`). This uses another window to display the file, like the `o` command.

`v` View the file described on the current line, using `M-x view-file` (`dired-view-file`).

Viewing a file is like visiting it, but is slanted toward moving around in the file conveniently and does not allow changing the file. See Section 14.10 [Misc File Ops], page 181.

28.6 Dired Marks vs. Flags

Instead of flagging a file with '`D`', you can *mark* the file with some other character (usually '`*`'). Most Dired commands to operate on files, aside from "expunge" (`x`), look for files marked with '`*`'.

Here are some commands for marking with '`*`', or for unmarking or operating on marks. (See Section 28.3 [Dired Deletion], page 388, for commands to flag and unflag files.)

`m`
`* m` Mark the current file with '`*`' (`dired-mark`). With a numeric argument *n*, mark the next *n* files starting with the current file. (If *n* is negative, mark the previous $-n$ files.)

`* *` Mark all executable files with '`*`' (`dired-mark-executables`). With a numeric argument, unmark all those files.

`* @` Mark all symbolic links with '`*`' (`dired-mark-symlinks`). With a numeric argument, unmark all those files.

`* /` Mark with '`*`' all files which are actually directories, except for '`.`' and '`..`' (`dired-mark-directories`). With a numeric argument, unmark all those files.

`* s` Mark all the files in the current subdirectory, aside from '`.`' and '`..`' (`dired-mark-subdir-files`).

`u`
`* u` Remove any mark on this line (`dired-unmark`).

⟨DEL⟩
`*` ⟨DEL⟩ Move point to previous line and remove any mark on that line (`dired-unmark-backward`).

`* !` Remove all marks from all the files in this Dired buffer (`dired-unmark-all-files-no-query`).

`* ?` *markchar*
Remove all marks that use the character *markchar* (`dired-unmark-all-files`). The argument is a single character—do not use ⟨RET⟩ to terminate it. See the description of the `* c` command below, which lets you replace one mark character with another.

With a numeric argument, this command queries about each marked file, asking whether to remove its mark. You can answer `y` meaning yes, `n` meaning no, or `!` to remove the marks from the remaining files without asking about them.

`* C-n` Move down to the next marked file (`dired-next-marked-file`) A file is "marked" if it has any kind of mark.

`* C-p` Move up to the previous marked file (`dired-prev-marked-file`)

`* t` Toggle all marks (`dired-do-toggle`): files marked with '`*`' become unmarked, and unmarked files are marked with '`*`'. Files marked in any other way are not affected.

`* c` *old-markchar new-markchar*
Replace all marks that use the character *old-markchar* with marks that use the character *new-markchar* (`dired-change-marks`). This command is the primary way to create or use marks other than '`*`' or '`D`'. The arguments are single characters—do not use ⟨RET⟩ to terminate them.

You can use almost any character as a mark character by means of this command, to distinguish various classes of files. If *old-markchar* is a space (' '), then the command operates on all unmarked files; if *new-markchar* is a space, then the command unmarks the files it acts on.

To illustrate the power of this command, here is how to put 'D' flags on all the files that have no marks, while unflagging all those that already have 'D' flags:

```
* c D t  * c SPC D  * c t SPC
```

This assumes that no files were already marked with 't'.

`% m` *regexp* ⟨RET⟩
`* %` *regexp* ⟨RET⟩
Mark (with '*') all files whose names match the regular expression *regexp* (`dired-mark-files-regexp`). This command is like `% d`, except that it marks files with '*' instead of flagging with 'D'. See Section 28.4 [Flagging Many Files], page 388.

Only the non-directory part of the file name is used in matching. Use '^' and '$' to anchor matches. Exclude subdirectories by hiding them (see Section 28.13 [Hiding Subdirectories], page 398).

`% g` *regexp* ⟨RET⟩
Mark (with '*') all files whose *contents* contain a match for the regular expression *regexp* (`dired-mark-files-containing-regexp`). This command is like `% m`, except that it searches the file contents instead of the file name.

`C-_` Undo changes in the Dired buffer, such as adding or removing marks (`dired-undo`).

28.7 Operating on Files

This section describes the basic Dired commands to operate on one file or several files. All of these commands are capital letters; all of them use the minibuffer, either to read an argument or to ask for confirmation, before they act. All of them give you several ways to specify which files to manipulate:

- If you give the command a numeric prefix argument n, it operates on the next n files, starting with the current file. (If n is negative, the command operates on the $-n$ files preceding the current line.)
- Otherwise, if some files are marked with '*', the command operates on all those files.
- Otherwise, the command operates on the current file only.

Here are the file-manipulating commands that operate on files in this way. (Some other Dired commands, such as `!` and the '%' commands, also use these conventions to decide which files to work on.)

`C` *new* ⟨RET⟩
: Copy the specified files (`dired-do-copy`). The argument *new* is the directory to copy into, or (if copying a single file) the new name.

  If `dired-copy-preserve-time` is non-`nil`, then copying with this command sets the modification time of the new file to be the same as that of the old file.

  The variable `dired-recursive-copies` controls whether directories are copied recursively. The default is to not copy recursively, which means that directories cannot be copied.

`D`
: Delete the specified files (`dired-do-delete`). Like the other commands in this section, this command operates on the *marked* files, or the next *n* files. By contrast, `x` (`dired-expunge`) deletes all *flagged* files.

`R` *new* ⟨RET⟩
: Rename the specified files (`dired-do-rename`). The argument *new* is the directory to rename into, or (if renaming a single file) the new name.

  Dired automatically changes the visited file name of buffers associated with renamed files so that they refer to the new names.

`H` *new* ⟨RET⟩
: Make hard links to the specified files (`dired-do-hardlink`). The argument *new* is the directory to make the links in, or (if making just one link) the name to give the link.

`S` *new* ⟨RET⟩
: Make symbolic links to the specified files (`dired-do-symlink`). The argument *new* is the directory to make the links in, or (if making just one link) the name to give the link.

`M` *modespec* ⟨RET⟩
: Change the mode (also called "permission bits") of the specified files (`dired-do-chmod`). This uses the `chmod` program, so *modespec* can be any argument that `chmod` can handle.

`G` *newgroup* ⟨RET⟩
: Change the group of the specified files to *newgroup* (`dired-do-chgrp`).

`O` *newowner* ⟨RET⟩
: Change the owner of the specified files to *newowner* (`dired-do-chown`). (On most systems, only the superuser can do this.)

  The variable `dired-chown-program` specifies the name of the program to use to do the work (different systems put `chown` in different places).

`P` *command* ⟨RET⟩
: Print the specified files (`dired-do-print`). You must specify the command to print them with, but the minibuffer starts out with a suitable guess made using the variables `lpr-command` and `lpr-switches` (the same variables that `lpr-buffer` uses; see Section 30.5 [Hardcopy], page 438).

`Z`
: Compress the specified files (`dired-do-compress`). If the file appears to be a compressed file already, it is uncompressed instead.

`L`
: Load the specified Emacs Lisp files (`dired-do-load`). See Section 23.7 [Lisp Libraries], page 339.

`B`
: Byte compile the specified Emacs Lisp files (`dired-do-byte-compile`). See section "Byte Compilation" in *The Emacs Lisp Reference Manual*.

`A` *regexp* ⟨RET⟩
: Search all the specified files for the regular expression *regexp* (`dired-do-search`).

  This command is a variant of `tags-search`. The search stops at the first match it finds; use `M-,` to resume the search and find the next match. See Section 22.16.6 [Tags Search], page 309.

`Q` *from* ⟨RET⟩ *to* ⟨RET⟩
: Perform `query-replace-regexp` on each of the specified files, replacing matches for *from* (a regular expression) with the string *to* (`dired-do-query-replace`).

  This command is a variant of `tags-query-replace`. If you exit the query replace loop, you can use `M-,` to resume the scan and replace more matches. See Section 22.16.6 [Tags Search], page 309.

One special file-operation command is `+` (`dired-create-directory`). This command reads a directory name and creates the directory if it does not already exist.

28.8 Shell Commands in Dired

The Dired command `!` (`dired-do-shell-command`) reads a shell command string in the minibuffer and runs that shell command on all the specified files. `X` is a synonym for `!`. You can specify the files to operate on in the usual ways for Dired commands (see Section 28.7 [Operating on Files], page 392). There are two ways of applying a shell command to multiple files:

- If you use '`*`' in the shell command, then it runs just once, with the list of file names substituted for the '`*`'. The order of file names is the order of appearance in the Dired buffer.

Thus, `! tar cf foo.tar *` ⟨RET⟩ runs `tar` on the entire list of file names, putting them into one tar file '`foo.tar`'.

- If the command string doesn't contain '`*`', then it runs once *for each file*, with the file name added at the end.

  For example, `! uudecode` ⟨RET⟩ runs `uudecode` on each file.

What if you want to run the shell command once for each file, with the file name inserted in the middle? You can use '`?`' in the command instead of '`*`'. The current file name is substituted for '`?`'. You can use '`?`' more than once. For instance, here is how to uuencode each file, making the output file name by appending '`.uu`' to the input file name:

```
uuencode ? ? > ?.uu
```

To use the file names in a more complicated fashion, you can use a shell loop. For example, this shell command is another way to uuencode each file:

```
for file in *; do uuencode $file $file >$file.uu; done
```

The working directory for the shell command is the top-level directory of the Dired buffer.

The `!` command does not attempt to update the Dired buffer to show new or modified files, because it doesn't really understand shell commands, and does not know what files the shell command changed. Use the `g` command to update the Dired buffer (see Section 28.14 [Dired Updating], page 398).

28.9 Transforming File Names in Dired

Here are commands that alter file names in a systematic way:

`% u` Rename each of the selected files to an upper-case name (`dired-upcase`). If the old file names are '`Foo`' and '`bar`', the new names are '`FOO`' and '`BAR`'.

`% l` Rename each of the selected files to a lower-case name (`dired-downcase`). If the old file names are '`Foo`' and '`bar`', the new names are '`foo`' and '`bar`'.

`% R` *from* ⟨RET⟩ *to* ⟨RET⟩
`% C` *from* ⟨RET⟩ *to* ⟨RET⟩
`% H` *from* ⟨RET⟩ *to* ⟨RET⟩
`% S` *from* ⟨RET⟩ *to* ⟨RET⟩

These four commands rename, copy, make hard links and make soft links, in each case computing the new name by regular-expression substitution from the name of the old file.

The four regular-expression substitution commands effectively perform a search-and-replace on the selected file names in the Dired buffer. They read two arguments: a regular expression *from*, and a substitution pattern *to*.

The commands match each "old" file name against the regular expression *from*, and then replace the matching part with *to*. You can use '`\&`' and '`\`*digit*' in *to* to refer to all or part of what the pattern matched in the old file name, as in `replace-regexp` (see Section 12.7.2 [Regexp Replace], page 133). If the regular expression matches more than once in a file name, only the first match is replaced.

For example, `% R ^.*$ RET x-\& RET` renames each selected file by prepending '`x-`' to its name. The inverse of this, removing '`x-`' from the front of each file name, is also possible: one method is `% R ^x-\(.*\)$ RET \1 RET`; another is `% R ^x- RET RET`. (Use '`^`' and '`$`' to anchor matches that should span the whole filename.)

Normally, the replacement process does not consider the files' directory names; it operates on the file name within the directory. If you specify a numeric argument of zero, then replacement affects the entire absolute file name including directory name.

Often you will want to select the set of files to operate on using the same *regexp* that you will use to operate on them. To do this, mark those files with `% m` *regexp* `RET`, then use the same regular expression in the command to operate on the files. To make this easier, the `%` commands to operate on files use the last regular expression specified in any `%` command as a default.

28.10 File Comparison with Dired

Here are two Dired commands that compare specified files using `diff`.

`=` Compare the current file (the file at point) with another file (the file at the mark) using the `diff` program (`dired-diff`). The file at the mark is the first argument of `diff`, and the file at point is the second argument.

`M-=` Compare the current file with its latest backup file (`dired-backup-diff`). If the current file is itself a backup, compare it with the file it is a backup of; this way, you can compare a file with any backup version of your choice.

The backup file is the first file given to `diff`.

28.11 Subdirectories in Dired

A Dired buffer displays just one directory in the normal case; but you can optionally include its subdirectories as well.

The simplest way to include multiple directories in one Dired buffer is to specify the options '`-lR`' for running `ls`. (If you give a numeric argument when you run Dired, then you can specify these options in the minibuffer.)

That produces a recursive directory listing showing all subdirectories at all levels.

But usually all the subdirectories are too many; usually you will prefer to include specific subdirectories only. You can do this with the `i` command:

`i` Insert the contents of a subdirectory later in the buffer.

Use the `i` (`dired-maybe-insert-subdir`) command on a line that describes a file which is a directory. It inserts the contents of that directory into the same Dired buffer, and moves there. Inserted subdirectory contents follow the top-level directory of the Dired buffer, just as they do in '`ls -lR`' output.

If the subdirectory's contents are already present in the buffer, the `i` command just moves to it.

In either case, `i` sets the Emacs mark before moving, so `C-u C-`⟨SPC⟩ takes you back to the old position in the buffer (the line describing that subdirectory).

Use the `l` command (`dired-do-redisplay`) to update the subdirectory's contents. Use `C-u k` on the subdirectory header line to delete the subdirectory. See Section 28.14 [Dired Updating], page 398.

28.12 Moving Over Subdirectories

When a Dired buffer lists subdirectories, you can use the page motion commands `C-x [` and `C-x ]` to move by entire directories.

The following commands move across, up and down in the tree of directories within one Dired buffer. They move to *directory header lines*, which are the lines that give a directory's name, at the beginning of the directory's contents.

`C-M-n` Go to next subdirectory header line, regardless of level (`dired-next-subdir`).

`C-M-p` Go to previous subdirectory header line, regardless of level (`dired-prev-subdir`).

`C-M-u` Go up to the parent directory's header line (`dired-tree-up`).

`C-M-d` Go down in the directory tree, to the first subdirectory's header line (`dired-tree-down`).

`<` Move up to the previous directory-file line (`dired-prev-dirline`). These lines are the ones that describe a directory as a file in its parent directory.

`>` Move down to the next directory-file line (`dired-prev-dirline`).

28.13 Hiding Subdirectories

Hiding a subdirectory means to make it invisible, except for its header line, via selective display (see Section 11.9 [Selective Display], page 113).

`$` Hide or reveal the subdirectory that point is in, and move point to the next subdirectory (`dired-hide-subdir`). A numeric argument serves as a repeat count.

`M-$` Hide all subdirectories in this Dired buffer, leaving only their header lines (`dired-hide-all`). Or, if any subdirectory is currently hidden, make all subdirectories visible again. You can use this command to get an overview in very deep directory trees or to move quickly to subdirectories far away.

Ordinary Dired commands never consider files inside a hidden subdirectory. For example, the commands to operate on marked files ignore files in hidden directories even if they are marked. Thus you can use hiding to temporarily exclude subdirectories from operations without having to remove the markers.

The subdirectory hiding commands toggle; that is, they hide what was visible, and show what was hidden.

28.14 Updating the Dired Buffer

This section describes commands to update the Dired buffer to reflect outside (non-Dired) changes in the directories and files, and to delete part of the Dired buffer.

`g` Update the entire contents of the Dired buffer (`revert-buffer`).

`l` Update the specified files (`dired-do-redisplay`).

`k` Delete the specified *file lines*—not the files, just the lines (`dired-do-kill-lines`).

`s` Toggle between alphabetical order and date/time order (`dired-sort-toggle-or-edit`).

`C-u s` *switches* ⟨RET⟩
Refresh the Dired buffer using *switches* as `dired-listing-switches`.

Type `g` (`revert-buffer`) to update the contents of the Dired buffer, based on changes in the files and directories listed. This preserves all marks except for those on files that have vanished. Hidden subdirectories are updated but remain hidden.

To update only some of the files, type `l` (`dired-do-redisplay`). This command applies to the next *n* files, or to the marked files if any, or to the current file. Updating them means reading their current status from the file system and changing the buffer to reflect it properly.

If you use `l` on a subdirectory header line, it updates the contents of the corresponding subdirectory.

To delete the specified *file lines*—not the files, just the lines—type `k` (`dired-do-kill-lines`). With a numeric argument *n*, this command applies to the next *n* files; otherwise, it applies to the marked files.

If you kill the line for a file that is a directory, the directory's contents are also deleted from the buffer. Typing `C-u k` on the header line for a subdirectory is another way to delete a subdirectory from the Dired buffer.

The `g` command brings back any individual lines that you have killed in this way, but not subdirectories—you must use `i` to reinsert each subdirectory.

The files in a Dired buffers are normally listed in alphabetical order by file names. Alternatively Dired can sort them by date/time. The Dired command `s` (`dired-sort-toggle-or-edit`) switches between these two sorting modes. The mode line in a Dired buffer indicates which way it is currently sorted—by name, or by date.

`C-u s` *switches* ⟨RET⟩ lets you specify a new value for `dired-listing-switches`.

28.15 Dired and `find`

You can select a set of files for display in a Dired buffer more flexibly by using the `find` utility to choose the files.

To search for files with names matching a wildcard pattern use `M-x find-name-dired`. It reads arguments *directory* and *pattern*, and chooses all the files in *directory* or its subdirectories whose individual names match *pattern*.

The files thus chosen are displayed in a Dired buffer in which the ordinary Dired commands are available.

If you want to test the contents of files, rather than their names, use `M-x find-grep-dired`. This command reads two minibuffer arguments, *directory* and *regexp*; it chooses all the files in *directory* or its subdirectories that contain a match for *regexp*. It works by running the programs `find` and `grep`. See also `M-x grep-find`, in Section 23.1 [Compilation], page 331. Remember to write the regular expression for `grep`, not for Emacs. (An alternative method of showing files whose contents match a given regexp is the `% g` *regexp* command, see Section 28.6 [Marks vs Flags], page 390.)

The most general command in this series is `M-x find-dired`, which lets you specify any condition that `find` can test. It takes two minibuffer argu-

ments, *directory* and *find-args*; it runs `find` in *directory*, passing *find-args* to tell `find` what condition to test. To use this command, you need to know how to use `find`.

`M-x locate` provides a similar interface to the `locate`. `M-x locate-with-filter` is similar, but keeps only lines matching a given regular expression.

The format of listing produced by these commands is controlled by the variable `find-ls-option`, whose default value specifies using options '`-ld`' for `ls`. If your listings are corrupted, you may need to change the value of this variable.

29 The Calendar and the Diary

Emacs provides the functions of a desk calendar, with a diary of planned or past events. It also has facilities for managing your appointments, and keeping track of how much time you spend working on certain projects.

To enter the calendar, type `M-x calendar`; this displays a three-month calendar centered on the current month, with point on the current date. With a numeric argument, as in `C-u M-x calendar`, it prompts you for the month and year to be the center of the three-month calendar. The calendar uses its own buffer, whose major mode is Calendar mode.

`Mouse-2` in the calendar brings up a menu of operations on a particular date; `C-Mouse-3` brings up a menu of commonly used calendar features that are independent of any particular date. To exit the calendar, type `q`. See section "Calendar" in *The Emacs Lisp Reference Manual*, for customization information about the calendar and diary.

29.1 Movement in the Calendar

Calendar mode lets you move through the calendar in logical units of time such as days, weeks, months, and years. If you move outside the three months originally displayed, the calendar display "scrolls" automatically through time to make the selected date visible. Moving to a date lets you view its holidays or diary entries, or convert it to other calendars; moving longer time periods is also useful simply to scroll the calendar.

29.1.1 Motion by Standard Lengths of Time

The commands for movement in the calendar buffer parallel the commands for movement in text. You can move forward and backward by days, weeks, months, and years.

`C-f` Move point one day forward (`calendar-forward-day`).

`C-b` Move point one day backward (`calendar-backward-day`).

`C-n` Move point one week forward (`calendar-forward-week`).

`C-p` Move point one week backward (`calendar-backward-week`).

`M-}` Move point one month forward (`calendar-forward-month`).

`M-{` Move point one month backward (`calendar-backward-month`).

`C-x ]` Move point one year forward (`calendar-forward-year`).

`C-x [` Move point one year backward (`calendar-backward-year`).

The day and week commands are natural analogues of the usual Emacs commands for moving by characters and by lines. Just as `C-n` usually moves to the same column in the following line, in Calendar mode it moves to the same day in the following week. And `C-p` moves to the same day in the previous week.

The arrow keys are equivalent to `C-f`, `C-b`, `C-n` and `C-p`, just as they normally are in other modes.

The commands for motion by months and years work like those for weeks, but move a larger distance. The month commands `M-}` and `M-{` move forward or backward by an entire month's time. The year commands `C-x ]` and `C-x [` move forward or backward a whole year.

The easiest way to remember these commands is to consider months and years analogous to paragraphs and pages of text, respectively. But the commands themselves are not quite analogous. The ordinary Emacs paragraph commands move to the beginning or end of a paragraph, whereas these month and year commands move by an entire month or an entire year, which usually involves skipping across the end of a month or year.

All these commands accept a numeric argument as a repeat count. For convenience, the digit keys and the minus sign specify numeric arguments in Calendar mode even without the Meta modifier. For example, `100 C-f` moves point 100 days forward from its present location.

29.1.2 Beginning or End of Week, Month or Year

A week (or month, or year) is not just a quantity of days; we think of weeks (months, years) as starting on particular dates. So Calendar mode provides commands to move to the beginning or end of a week, month or year:

`C-a`	Move point to start of week (`calendar-beginning-of-week`).
`C-e`	Move point to end of week (`calendar-end-of-week`).
`M-a`	Move point to start of month (`calendar-beginning-of-month`).
`M-e`	Move point to end of month (`calendar-end-of-month`).
`M-<`	Move point to start of year (`calendar-beginning-of-year`).
`M->`	Move point to end of year (`calendar-end-of-year`).

These commands also take numeric arguments as repeat counts, with the repeat count indicating how many weeks, months, or years to move backward or forward.

By default, weeks begin on Sunday. To make them begin on Monday instead, set the variable `calendar-week-start-day` to 1.

29.1.3 Specified Dates

Calendar mode provides commands for moving to a particular date specified in various ways.

`g d` Move point to specified date (`calendar-goto-date`).

`o` Center calendar around specified month (`calendar-other-month`).

`.` Move point to today's date (`calendar-goto-today`).

`g d` (`calendar-goto-date`) prompts for a year, a month, and a day of the month, and then moves to that date. Because the calendar includes all dates from the beginning of the current era, you must type the year in its entirety; that is, type '`1990`', not '`90`'.

`o` (`calendar-other-month`) prompts for a month and year, then centers the three-month calendar around that month.

You can return to today's date with `.` (`calendar-goto-today`).

29.2 Scrolling in the Calendar

The calendar display scrolls automatically through time when you move out of the visible portion. You can also scroll it manually. Imagine that the calendar window contains a long strip of paper with the months on it. Scrolling it means moving the strip so that new months become visible in the window.

`C-x <` Scroll calendar one month forward (`scroll-calendar-left`).

`C-x >` Scroll calendar one month backward (`scroll-calendar-right`).

`C-v`
⟨NEXT⟩ Scroll calendar three months forward (`scroll-calendar-left-three-months`).

`M-v`
⟨PRIOR⟩ Scroll calendar three months backward (`scroll-calendar-right-three-months`).

The most basic calendar scroll commands scroll by one month at a time. This means that there are two months of overlap between the display before the command and the display after. `C-x <` scrolls the calendar contents one month to the left; that is, it moves the display forward in time. `C-x >` scrolls the contents to the right, which moves backwards in time.

The commands C-v and M-v scroll the calendar by an entire "screenful"—three months—in analogy with the usual meaning of these commands. C-v makes later dates visible and M-v makes earlier dates visible. These commands take a numeric argument as a repeat count; in particular, since C-u multiplies the next command by four, typing C-u C-v scrolls the calendar forward by a year and typing C-u M-v scrolls the calendar backward by a year.

The function keys ⟨NEXT⟩ and ⟨PRIOR⟩ are equivalent to C-v and M-v, just as they are in other modes.

29.3 Counting Days

M-= Display the number of days in the current region (calendar-count-days-region).

To determine the number of days in the region, type M-= (calendar-count-days-region). The numbers of days printed is *inclusive*; that is, it includes the days specified by mark and point.

29.4 Miscellaneous Calendar Commands

p d Display day-in-year (calendar-print-day-of-year).

C-c C-l Regenerate the calendar window (redraw-calendar).

SPC Scroll the next window (scroll-other-window).

q Exit from calendar (exit-calendar).

To print the number of days elapsed since the start of the year, or the number of days remaining in the year, type the p d command (calendar-print-day-of-year). This displays both of those numbers in the echo area. The number of days elapsed includes the selected date. The number of days remaining does not include that date.

If the calendar window text gets corrupted, type C-c C-l (redraw-calendar) to redraw it. (This can only happen if you use non-Calendar-mode editing commands.)

In Calendar mode, you can use SPC (scroll-other-window) to scroll the other window. This is handy when you display a list of holidays or diary entries in another window.

To exit from the calendar, type q (exit-calendar). This buries all buffers related to the calendar, selecting other buffers. (If a frame contains a dedicated calendar window, exiting from the calendar iconifies that frame.)

29.5 LaTeX Calendar

The Calendar LaTeX commands produce a buffer of LaTeX code that prints as a calendar. Depending on the command you use, the printed calendar covers the day, week, month or year that point is in.

`t m` Generate a one-month calendar (`cal-tex-cursor-month`).

`t M` Generate a sideways-printing one-month calendar (`cal-tex-cursor-month-landscape`).

`t d` Generate a one-day calendar (`cal-tex-cursor-day`).

`t w 1` Generate a one-page calendar for one week (`cal-tex-cursor-week`).

`t w 2` Generate a two-page calendar for one week (`cal-tex-cursor-week2`).

`t w 3` Generate an ISO-style calendar for one week (`cal-tex-cursor-week-iso`).

`t w 4` Generate a calendar for one Monday-starting week (`cal-tex-cursor-week-monday`).

`t f w` Generate a Filofax-style two-weeks-at-a-glance calendar (`cal-tex-cursor-filofax-2week`).

`t f W` Generate a Filofax-style one-week-at-a-glance calendar (`cal-tex-cursor-filofax-week`).

`t y` Generate a calendar for one year (`cal-tex-cursor-year`).

`t Y` Generate a sideways-printing calendar for one year (`cal-tex-cursor-year-landscape`).

`t f y` Generate a Filofax-style calendar for one year (`cal-tex-cursor-filofax-year`).

Some of these commands print the calendar sideways (in "landscape mode"), so it can be wider than it is long. Some of them use Filofax paper size (3.75in x 6.75in). All of these commands accept a prefix argument which specifies how many days, weeks, months or years to print (starting always with the selected one).

If the variable `cal-tex-holidays` is non-`nil` (the default), then the printed calendars show the holidays in `calendar-holidays`. If the variable `cal-tex-diary` is non-`nil` (the default is `nil`), diary entries are included also (in weekly and monthly calendars only). If the variable `cal-tex-rules` is non-`nil` (the default is `nil`), the calendar displays ruled pages in styles that have sufficient room.

29.6 Holidays

The Emacs calendar knows about all major and many minor holidays, and can display them.

`h` Display holidays for the selected date (`calendar-cursor-holidays`).

`Mouse-2 Holidays`
Display any holidays for the date you click on.

`x` Mark holidays in the calendar window (`mark-calendar-holidays`).

`u` Unmark calendar window (`calendar-unmark`).

`a` List all holidays for the displayed three months in another window (`list-calendar-holidays`).

`M-x holidays`
List all holidays for three months around today's date in another window.

`M-x list-holidays`
List holidays in another window for a specified range of years.

To see if any holidays fall on a given date, position point on that date in the calendar window and use the `h` command. Alternatively, click on that date with `Mouse-2` and then choose `Holidays` from the menu that appears. Either way, this displays the holidays for that date, in the echo area if they fit there, otherwise in a separate window.

To view the distribution of holidays for all the dates shown in the calendar, use the `x` command. This displays the dates that are holidays in a different face (or places a '`*`' after these dates, if display with multiple faces is not available). The command applies both to the currently visible months and to other months that subsequently become visible by scrolling. To turn marking off and erase the current marks, type `u`, which also erases any diary marks (see Section 29.10 [Diary], page 413).

To get even more detailed information, use the `a` command, which displays a separate buffer containing a list of all holidays in the current three-month range. You can use ⟨SPC⟩ in the calendar window to scroll that list.

The command `M-x holidays` displays the list of holidays for the current month and the preceding and succeeding months; this works even if you don't have a calendar window. If you want the list of holidays centered around a different month, use `C-u M-x holidays`, which prompts for the month and year.

The holidays known to Emacs include United States holidays and the major Christian, Jewish, and Islamic holidays; also the solstices and equinoxes.

The command `M-x list-holidays` displays the list of holidays for a range of years. This function asks you for the starting and stopping years, and allows you to choose all the holidays or one of several categories of holidays. You can use this command even if you don't have a calendar window.

The dates used by Emacs for holidays are based on *current practice*, not historical fact. Historically, for instance, the start of daylight savings time and even its existence have varied from year to year, but present United States law mandates that daylight savings time begins on the first Sunday in April. When the daylight savings rules are set up for the United States, Emacs always uses the present definition, even though it is wrong for some prior years.

29.7 Times of Sunrise and Sunset

Special calendar commands can tell you, to within a minute or two, the times of sunrise and sunset for any date.

`S` Display times of sunrise and sunset for the selected date (`calendar-sunrise-sunset`).

`Mouse-2 Sunrise/Sunset`
Display times of sunrise and sunset for the date you click on.

`M-x sunrise-sunset`
Display times of sunrise and sunset for today's date.

`C-u M-x sunrise-sunset`
Display times of sunrise and sunset for a specified date.

Within the calendar, to display the *local times* of sunrise and sunset in the echo area, move point to the date you want, and type `S`. Alternatively, click `Mouse-2` on the date, then choose `Sunrise/Sunset` from the menu that appears. The command `M-x sunrise-sunset` is available outside the calendar to display this information for today's date or a specified date. To specify a date other than today, use `C-u M-x sunrise-sunset`, which prompts for the year, month, and day.

You can display the times of sunrise and sunset for any location and any date with `C-u C-u M-x sunrise-sunset`. This asks you for a longitude, latitude, number of minutes difference from Coordinated Universal Time, and date, and then tells you the times of sunrise and sunset for that location on that date.

Because the times of sunrise and sunset depend on the location on earth, you need to tell Emacs your latitude, longitude, and location name before using these commands. Here is an example of what to set:

```
(setq calendar-latitude 40.1)
(setq calendar-longitude -88.2)
```

```
(setq calendar-location-name "Urbana, IL")
```

Use one decimal place in the values of `calendar-latitude` and `calendar-longitude`.

Your time zone also affects the local time of sunrise and sunset. Emacs usually gets time zone information from the operating system, but if these values are not what you want (or if the operating system does not supply them), you must set them yourself. Here is an example:

```
(setq calendar-time-zone -360)
(setq calendar-standard-time-zone-name "CST")
(setq calendar-daylight-time-zone-name "CDT")
```

The value of `calendar-time-zone` is the number of minutes difference between your local standard time and Coordinated Universal Time (Greenwich time). The values of `calendar-standard-time-zone-name` and `calendar-daylight-time-zone-name` are the abbreviations used in your time zone. Emacs displays the times of sunrise and sunset *corrected for daylight savings time*. See Section 29.12 [Daylight Savings], page 421, for how daylight savings time is determined.

As a user, you might find it convenient to set the calendar location variables for your usual physical location in your '`.emacs`' file. And when you install Emacs on a machine, you can create a '`default.el`' file which sets them properly for the typical location of most users of that machine. See Section 31.7 [Init File], page 486.

29.8 Phases of the Moon

These calendar commands display the dates and times of the phases of the moon (new moon, first quarter, full moon, last quarter). This feature is useful for debugging problems that "depend on the phase of the moon."

`M` Display the dates and times for all the quarters of the moon for the three-month period shown (`calendar-phases-of-moon`).

`M-x phases-of-moon`
Display dates and times of the quarters of the moon for three months around today's date.

Within the calendar, use the `M` command to display a separate buffer of the phases of the moon for the current three-month range. The dates and times listed are accurate to within a few minutes.

Outside the calendar, use the command `M-x phases-of-moon` to display the list of the phases of the moon for the current month and the preceding and succeeding months. For information about a different month, use `C-u M-x phases-of-moon`, which prompts for the month and year.

The dates and times given for the phases of the moon are given in local time (corrected for daylight savings, when appropriate); but if the variable

`calendar-time-zone` is void, Coordinated Universal Time (the Greenwich time zone) is used. See Section 29.12 [Daylight Savings], page 421.

29.9 Conversion To and From Other Calendars

The Emacs calendar displayed is *always* the Gregorian calendar, sometimes called the "new style" calendar, which is used in most of the world today. However, this calendar did not exist before the sixteenth century and was not widely used before the eighteenth century; it did not fully displace the Julian calendar and gain universal acceptance until the early twentieth century. The Emacs calendar can display any month since January, year 1 of the current era, but the calendar displayed is the Gregorian, even for a date at which the Gregorian calendar did not exist.

While Emacs cannot display other calendars, it can convert dates to and from several other calendars.

29.9.1 Supported Calendar Systems

The ISO commercial calendar is used largely in Europe.

The Julian calendar, named after Julius Caesar, was the one used in Europe throughout medieval times, and in many countries up until the nineteenth century.

Astronomers use a simple counting of days elapsed since noon, Monday, January 1, 4713 B.C. on the Julian calendar. The number of days elapsed is called the *Julian day number* or the *Astronomical day number*.

The Hebrew calendar is used by tradition in the Jewish religion. The Emacs calendar program uses the Hebrew calendar to determine the dates of Jewish holidays. Hebrew calendar dates begin and end at sunset.

The Islamic calendar is used in many predominantly Islamic countries. Emacs uses it to determine the dates of Islamic holidays. There is no universal agreement in the Islamic world about the calendar; Emacs uses a widely accepted version, but the precise dates of Islamic holidays often depend on proclamation by religious authorities, not on calculations. As a consequence, the actual dates of observance can vary slightly from the dates computed by Emacs. Islamic calendar dates begin and end at sunset.

The French Revolutionary calendar was created by the Jacobins after the 1789 revolution, to represent a more secular and nature-based view of the annual cycle, and to install a 10-day week in a rationalization measure similar to the metric system. The French government officially abandoned this calendar at the end of 1805.

The Maya of Central America used three separate, overlapping calendar systems, the *long count*, the *tzolkin*, and the *haab*. Emacs knows about

all three of these calendars. Experts dispute the exact correlation between the Mayan calendar and our calendar; Emacs uses the Goodman-Martinez-Thompson correlation in its calculations.

The Copts use a calendar based on the ancient Egyptian solar calendar. Their calendar consists of twelve 30-day months followed by an extra five-day period. Once every fourth year they add a leap day to this extra period to make it six days. The Ethiopic calendar is identical in structure, but has different year numbers and month names.

The Persians use a solar calendar based on a design of Omar Khayyam. Their calendar consists of twelve months of which the first six have 31 days, the next five have 30 days, and the last has 29 in ordinary years and 30 in leap years. Leap years occur in a complicated pattern every four or five years.

The Chinese calendar is a complicated system of lunar months arranged into solar years. The years go in cycles of sixty, each year containing either twelve months in an ordinary year or thirteen months in a leap year; each month has either 29 or 30 days. Years, ordinary months, and days are named by combining one of ten "celestial stems" with one of twelve "terrestrial branches" for a total of sixty names that are repeated in a cycle of sixty.

29.9.2 Converting To Other Calendars

The following commands describe the selected date (the date at point) in various other calendar systems:

`Mouse-2 Other Calendars`
: Display the date that you click on, expressed in various other calendars.

`p c`
: Display ISO commercial calendar equivalent for selected day (`calendar-print-iso-date`).

`p j`
: Display Julian date for selected day (`calendar-print-julian-date`).

`p a`
: Display astronomical (Julian) day number for selected day (`calendar-print-astro-day-number`).

`p h`
: Display Hebrew date for selected day (`calendar-print-hebrew-date`).

`p i`
: Display Islamic date for selected day (`calendar-print-islamic-date`).

`p f`
: Display French Revolutionary date for selected day (`calendar-print-french-date`).

`p C`
: Display Chinese date for selected day (`calendar-print-chinese-date`).

`p k` Display Coptic date for selected day (`calendar-print-coptic-date`).

`p e` Display Ethiopic date for selected day (`calendar-print-ethiopic-date`).

`p p` Display Persian date for selected day (`calendar-print-persian-date`).

`p m` Display Mayan date for selected day (`calendar-print-mayan-date`).

If you are using X, the easiest way to translate a date into other calendars is to click on it with `Mouse-2`, then choose `Other Calendars` from the menu that appears. This displays the equivalent forms of the date in all the calendars Emacs understands, in the form of a menu. (Choosing an alternative from this menu doesn't actually do anything—the menu is used only for display.)

Put point on the desired date of the Gregorian calendar, then type the appropriate keys. The `p` is a mnemonic for "print" since Emacs "prints" the equivalent date in the echo area.

29.9.3 Converting From Other Calendars

You can use the other supported calendars to specify a date to move to. This section describes the commands for doing this using calendars other than Mayan; for the Mayan calendar, see the following section.

`g c` Move to a date specified in the ISO commercial calendar (`calendar-goto-iso-date`).

`g j` Move to a date specified in the Julian calendar (`calendar-goto-julian-date`).

`g a` Move to a date specified in astronomical (Julian) day number (`calendar-goto-astro-day-number`).

`g h` Move to a date specified in the Hebrew calendar (`calendar-goto-hebrew-date`).

`g i` Move to a date specified in the Islamic calendar (`calendar-goto-islamic-date`).

`g f` Move to a date specified in the French Revolutionary calendar (`calendar-goto-french-date`).

`g C` Move to a date specified in the Chinese calendar (`calendar-goto-chinese-date`).

`g p` Move to a date specified in the Persian calendar (`calendar-goto-persian-date`).

`g k` Move to a date specified in the Coptic calendar (`calendar-goto-coptic-date`).

`g e` Move to a date specified in the Ethiopic calendar (`calendar-goto-ethiopic-date`).

These commands ask you for a date on the other calendar, move point to the Gregorian calendar date equivalent to that date, and display the other calendar's date in the echo area. Emacs uses strict completion (see Section 5.3 [Completion], page 57) whenever it asks you to type a month name, so you don't have to worry about the spelling of Hebrew, Islamic, or French names.

One common question concerning the Hebrew calendar is the computation of the anniversary of a date of death, called a "yahrzeit." The Emacs calendar includes a facility for such calculations. If you are in the calendar, the command `M-x list-yahrzeit-dates` asks you for a range of years and then displays a list of the yahrzeit dates for those years for the date given by point. If you are not in the calendar, this command first asks you for the date of death and the range of years, and then displays the list of yahrzeit dates.

29.9.4 Converting from the Mayan Calendar

Here are the commands to select dates based on the Mayan calendar:

`g m l` Move to a date specified by the long count calendar (`calendar-goto-mayan-long-count-date`).

`g m n t` Move to the next occurrence of a place in the tzolkin calendar (`calendar-next-tzolkin-date`).

`g m p t` Move to the previous occurrence of a place in the tzolkin calendar (`calendar-previous-tzolkin-date`).

`g m n h` Move to the next occurrence of a place in the haab calendar (`calendar-next-haab-date`).

`g m p h` Move to the previous occurrence of a place in the haab calendar (`calendar-previous-haab-date`).

`g m n c` Move to the next occurrence of a place in the calendar round (`calendar-next-calendar-round-date`).

`g m p c` Move to the previous occurrence of a place in the calendar round (`calendar-previous-calendar-round-date`).

To understand these commands, you need to understand the Mayan calendars. The *long count* is a counting of days with these units:

1 kin = 1 day 1 uinal = 20 kin 1 tun = 18 uinal
1 katun = 20 tun 1 baktun = 20 katun

Thus, the long count date 12.16.11.16.6 means 12 baktun, 16 katun, 11 tun, 16 uinal, and 6 kin. The Emacs calendar can handle Mayan long count dates as early as 7.17.18.13.1, but no earlier. When you use the `g m l` command, type the Mayan long count date with the baktun, katun, tun, uinal, and kin separated by periods.

The Mayan tzolkin calendar is a cycle of 260 days formed by a pair of independent cycles of 13 and 20 days. Since this cycle repeats endlessly, Emacs provides commands to move backward and forward to the previous or next point in the cycle. Type `g m p t` to go to the previous tzolkin date; Emacs asks you for a tzolkin date and moves point to the previous occurrence of that date. Similarly, type `g m n t` to go to the next occurrence of a tzolkin date.

The Mayan haab calendar is a cycle of 365 days arranged as 18 months of 20 days each, followed a 5-day monthless period. Like the tzolkin cycle, this cycle repeats endlessly, and there are commands to move backward and forward to the previous or next point in the cycle. Type `g m p h` to go to the previous haab date; Emacs asks you for a haab date and moves point to the previous occurrence of that date. Similarly, type `g m n h` to go to the next occurrence of a haab date.

The Maya also used the combination of the tzolkin date and the haab date. This combination is a cycle of about 52 years called a *calendar round*. If you type `g m p c`, Emacs asks you for both a haab and a tzolkin date and then moves point to the previous occurrence of that combination. Use `g m n c` to move point to the next occurrence of a combination. These commands signal an error if the haab/tzolkin date combination you have typed is impossible.

Emacs uses strict completion (see Section 5.3.3 [Strict Completion], page 59) whenever it asks you to type a Mayan name, so you don't have to worry about spelling.

29.10 The Diary

The Emacs diary keeps track of appointments or other events on a daily basis, in conjunction with the calendar. To use the diary feature, you must first create a *diary file* containing a list of events and their dates. Then Emacs can automatically pick out and display the events for today, for the immediate future, or for any specified date.

By default, Emacs uses '`~/diary`' as the diary file. This is the same file that the `calendar` utility uses. A sample '`~/diary`' file is:

```
12/22/1988  Twentieth wedding anniversary!!
&1/1.       Happy New Year!
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, 1989 Income tax due.
&* 15       time cards due.
```

This example uses extra spaces to align the event descriptions of most of the entries. Such formatting is purely a matter of taste.

Although you probably will start by creating a diary manually, Emacs provides a number of commands to let you view, add, and change diary entries.

29.10.1 Commands Displaying Diary Entries

Once you have created a '`~/diary`' file, you can use the calendar to view it. You can also view today's events outside of Calendar mode.

`d` Display all diary entries for the selected date (`view-diary-entries`).

`Mouse-2 Diary`
Display all diary entries for the date you click on.

`s` Display the entire diary file (`show-all-diary-entries`).

`m` Mark all visible dates that have diary entries (`mark-diary-entries`).

`u` Unmark the calendar window (`calendar-unmark`).

`M-x print-diary-entries`
Print hard copy of the diary display as it appears.

`M-x diary` Display all diary entries for today's date.

`M-x diary-mail-entries`
Mail yourself email reminders about upcoming diary entries.

Displaying the diary entries with `d` shows in a separate window the diary entries for the selected date in the calendar. The mode line of the new window shows the date of the diary entries and any holidays that fall on that date. If you specify a numeric argument with `d`, it shows all the diary entries for that many successive days. Thus, `2 d` displays all the entries for the selected date and for the following day.

Another way to display the diary entries for a date is to click `Mouse-2` on the date, and then choose `Diary` from the menu that appears.

To get a broader view of which days are mentioned in the diary, use the `m` command. This displays the dates that have diary entries in a different face (or places a '`+`' after these dates, if display with multiple faces is not available). The command applies both to the currently visible months and to other months that subsequently become visible by scrolling. To turn marking off and erase the current marks, type `u`, which also turns off holiday marks (see Section 29.6 [Holidays], page 406).

To see the full diary file, rather than just some of the entries, use the `s` command.

Display of selected diary entries uses the selective display feature to hide entries that don't apply.

The diary buffer as you see it is an illusion, so simply printing the buffer does not print what you see on your screen. There is a special command to print hard copy of the diary buffer *as it appears*; this command is `M-x print-diary-entries`. It sends the data directly to the printer. You can customize it like `lpr-region` (see Section 30.5 [Hardcopy], page 438).

The command `M-x diary` displays the diary entries for the current date, independently of the calendar display, and optionally for the next few days as well; the variable `number-of-diary-entries` specifies how many days to include. See section "Calendar" in *The Emacs Lisp Reference Manual*.

If you put `(diary)` in your '`.emacs`' file, this automatically displays a window with the day's diary entries, when you enter Emacs. The mode line of the displayed window shows the date and any holidays that fall on that date.

Many users like to receive notice of events in their diary as email. To send such mail to yourself, use the command `M-x diary-mail-entries`. A prefix argument specifies how many days (starting with today) to check; otherwise, the variable `diary-mail-days` says how many days.

29.10.2 The Diary File

Your *diary file* is a file that records events associated with particular dates. The name of the diary file is specified by the variable `diary-file`; '`~/diary`' is the default. The `calendar` utility program supports a subset of the format allowed by the Emacs diary facilities, so you can use that utility to view the diary file, with reasonable results aside from the entries it cannot understand.

Each entry in the diary file describes one event and consists of one or more lines. An entry always begins with a date specification at the left margin. The rest of the entry is simply text to describe the event. If the entry has more than one line, then the lines after the first must begin with whitespace to indicate they continue a previous entry. Lines that do not begin with valid dates and do not continue a preceding entry are ignored.

You can inhibit the marking of certain diary entries in the calendar window; to do this, insert an ampersand (‘&’) at the beginning of the entry, before the date. This has no effect on display of the entry in the diary window; it affects only marks on dates in the calendar window. Nonmarking entries are especially useful for generic entries that would otherwise mark many different dates.

If the first line of a diary entry consists only of the date or day name with no following blanks or punctuation, then the diary window display doesn't include that line; only the continuation lines appear. For example, this entry:

```
02/11/1989
      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
```

appears in the diary window without the date line at the beginning. This style of entry looks neater when you display just a single day's entries, but can cause confusion if you ask for more than one day's entries.

You can edit the diary entries as they appear in the window, but it is important to remember that the buffer displayed contains the *entire* diary file, with portions of it concealed from view. This means, for instance, that the `C-f` (`forward-char`) command can put point at what appears to be the end of the line, but what is in reality the middle of some concealed line.

Be careful when editing the diary entries! Inserting additional lines or adding/deleting characters in the middle of a visible line cannot cause problems, but editing at the end of a line may not do what you expect. Deleting a line may delete other invisible entries that follow it. Before editing the diary, it is best to display the entire file with `s` (`show-all-diary-entries`).

29.10.3 Date Formats

Here are some sample diary entries, illustrating different ways of formatting a date. The examples all show dates in American order (month, day, year), but Calendar mode supports European order (day, month, year) as an option.

```
4/20/93  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
```

The first entry appears only once, on April 20, 1993. The second and third appear every year on the specified dates, and the fourth uses a wildcard

(asterisk) for the month, so it appears on the 25th of every month. The final entry appears every week on Friday.

You can use just numbers to express a date, as in '*month*`/`*day*' or '*month*`/`*day*`/`*year*'. This must be followed by a nondigit. In the date itself, *month* and *day* are numbers of one or two digits. The optional *year* is also a number, and may be abbreviated to the last two digits; that is, you can use '`11/12/1989`' or '`11/12/89`'.

Dates can also have the form '*monthname day*' or '*monthname day*`,` *year*', where the month's name can be spelled in full or abbreviated to three characters (with or without a period). Case is not significant.

A date may be *generic*; that is, partially unspecified. Then the entry applies to all dates that match the specification. If the date does not contain a year, it is generic and applies to any year. Alternatively, *month*, *day*, or *year* can be a '`*`'; this matches any month, day, or year, respectively. Thus, a diary entry '`3/*/*`' matches any day in March of any year; so does '`march *`'.

If you prefer the European style of writing dates—in which the day comes before the month—type `M-x european-calendar` while in the calendar, or set the variable `european-calendar-style` to `t` *before* using any calendar or diary command. This mode interprets all dates in the diary in the European manner, and also uses European style for displaying diary dates. (Note that there is no comma after the *monthname* in the European style.) To go back to the (default) American style of writing dates, type `M-x american-calendar`.

You can use the name of a day of the week as a generic date which applies to any date falling on that day of the week. You can abbreviate the day of the week to three letters (with or without a period) or spell it in full; case is not significant.

29.10.4 Commands to Add to the Diary

While in the calendar, there are several commands to create diary entries:

`i d` Add a diary entry for the selected date (`insert-diary-entry`).

`i w` Add a diary entry for the selected day of the week (`insert-weekly-diary-entry`).

`i m` Add a diary entry for the selected day of the month (`insert-monthly-diary-entry`).

`i y` Add a diary entry for the selected day of the year (`insert-yearly-diary-entry`).

You can make a diary entry for a specific date by selecting that date in the calendar window and typing the `i d` command. This command displays

the end of your diary file in another window and inserts the date; you can then type the rest of the diary entry.

If you want to make a diary entry that applies to a specific day of the week, select that day of the week (any occurrence will do) and type `i w`. This inserts the day-of-week as a generic date; you can then type the rest of the diary entry. You can make a monthly diary entry in the same fashion. Select the day of the month, use the `i m` command, and type rest of the entry. Similarly, you can insert a yearly diary entry with the `i y` command.

All of the above commands make marking diary entries by default. To make a nonmarking diary entry, give a numeric argument to the command. For example, `C-u i w` makes a nonmarking weekly diary entry.

When you modify the diary file, be sure to save the file before exiting Emacs.

29.10.5 Special Diary Entries

In addition to entries based on calendar dates, the diary file can contain *sexp entries* for regular events such as anniversaries. These entries are based on Lisp expressions (sexps) that Emacs evaluates as it scans the diary file. Instead of a date, a sexp entry contains '`%%`' followed by a Lisp expression which must begin and end with parentheses. The Lisp expression determines which dates the entry applies to.

Calendar mode provides commands to insert certain commonly used sexp entries:

`i a` Add an anniversary diary entry for the selected date (`insert-anniversary-diary-entry`).

`i b` Add a block diary entry for the current region (`insert-block-diary-entry`).

`i c` Add a cyclic diary entry starting at the date (`insert-cyclic-diary-entry`).

If you want to make a diary entry that applies to the anniversary of a specific date, move point to that date and use the `i a` command. This displays the end of your diary file in another window and inserts the anniversary description; you can then type the rest of the diary entry. The entry looks like this:

```
%%(diary-anniversary 10 31 1948) Arthur's birthday
```

This entry applies to October 31 in any year after 1948; '`10 31 1948`' specifies the date. (If you are using the European calendar style, the month and day are interchanged.) The reason this expression requires a beginning year is that advanced diary functions can use it to calculate the number of elapsed years.

A *block* diary entry applies to a specified range of consecutive dates. Here is a block diary entry that applies to all dates from June 24, 1990 through July 10, 1990:

```
%%(diary-block 6 24 1990 7 10 1990) Vacation
```

The '`6 24 1990`' indicates the starting date and the '`7 10 1990`' indicates the stopping date. (Again, if you are using the European calendar style, the month and day are interchanged.)

To insert a block entry, place point and the mark on the two dates that begin and end the range, and type `i b`. This command displays the end of your diary file in another window and inserts the block description; you can then type the diary entry.

Cyclic diary entries repeat after a fixed interval of days. To create one, select the starting date and use the `i c` command. The command prompts for the length of interval, then inserts the entry, which looks like this:

```
%%(diary-cyclic 50 3 1 1990) Renew medication
```

This entry applies to March 1, 1990 and every 50th day following; '`3 1 1990`' specifies the starting date. (If you are using the European calendar style, the month and day are interchanged.)

All three of these commands make marking diary entries. To insert a nonmarking entry, give a numeric argument to the command. For example, `C-u i a` makes a nonmarking anniversary diary entry.

Marking sexp diary entries in the calendar is *extremely* time-consuming, since every date visible in the calendar window must be individually checked. So it's a good idea to make sexp diary entries nonmarking (with '`&`') when possible.

Another sophisticated kind of sexp entry, a *floating* diary entry, specifies a regularly occurring event by offsets specified in days, weeks, and months. It is comparable to a crontab entry interpreted by the `cron` utility. Here is a nonmarking, floating diary entry that applies to the last Thursday in November:

```
&%%(diary-float 11 4 -1) American Thanksgiving
```

The 11 specifies November (the eleventh month), the 4 specifies Thursday (the fourth day of the week, where Sunday is numbered zero), and the -1 specifies "last" (1 would mean "first," 2 would mean "second," -2 would mean "second-to-last," and so on). The month can be a single month or a list of months. Thus you could change the 11 above to '`'(1 2 3)`' and have the entry apply to the last Thursday of January, February, and March. If the month is `t`, the entry applies to all months of the year.

Most generally, sexp diary entries can perform arbitrary computations to determine when they apply. See section "Sexp Diary Entries" in *The Emacs Lisp Reference Manual*.

29.11 Appointments

If you have a diary entry for an appointment, and that diary entry begins with a recognizable time of day, Emacs can warn you, several minutes beforehand, that that appointment is pending. Emacs alerts you to the appointment by displaying a message in the mode line.

To enable appointment notification, you must enable the time display feature of Emacs, `M-x display-time` (see Section 1.3 [Mode Line], page 26). You must also add the function `appt-make-list` to the `diary-hook`, like this:

```
(add-hook 'diary-hook 'appt-make-list)
```

Adding this text to your '`.emacs`' file does the whole job:

```
(display-time)
(add-hook 'diary-hook 'appt-make-list)
(diary 0)
```

With these preparations done, when you display the diary (either with the `d` command in the calendar window or with the `M-x diary` command), it sets up an appointment list of all the diary entries found with recognizable times of day, and reminds you just before each of them.

For example, suppose the diary file contains these lines:

```
Monday
  9:30am Coffee break
 12:00pm Lunch
```

Then on Mondays, after you have displayed the diary, you will be reminded at 9:20am about your coffee break and at 11:50am about lunch.

You can write times in am/pm style (with '`12:00am`' standing for midnight and '`12:00pm`' standing for noon), or 24-hour European/military style. You need not be consistent; your diary file can have a mixture of the two styles.

Emacs updates the appointments list automatically just after midnight. This also displays the next day's diary entries in the diary buffer, unless you set `appt-display-diary` to `nil`.

You can also use the appointment notification facility like an alarm clock. The command `M-x appt-add` adds entries to the appointment list without affecting your diary file. You delete entries from the appointment list with `M-x appt-delete`.

You can turn off the appointment notification feature at any time by setting `appt-issue-message` to `nil`.

29.12 Daylight Savings Time

Emacs understands the difference between standard time and daylight savings time—the times given for sunrise, sunset, solstices, equinoxes, and the phases of the moon take that into account. The rules for daylight savings time vary from place to place and have also varied historically from year to year. To do the job properly, Emacs needs to know which rules to use.

Some operating systems keep track of the rules that apply to the place where you are; on these systems, Emacs gets the information it needs from the system automatically. If some or all of this information is missing, Emacs fills in the gaps with the rules currently used in Cambridge, Massachusetts. If the resulting rules are not what you want, you can tell Emacs the rules to use by setting certain variables: `calendar-daylight-savings-starts` and `calendar-daylight-savings-ends`.

These values should be Lisp expressions that refer to the variable `year`, and evaluate to the Gregorian date on which daylight savings time starts or (respectively) ends, in the form of a list (*month day year*). The values should be `nil` if your area does not use daylight savings time.

Emacs uses these expressions to determine the starting date of daylight savings time for the holiday list and for correcting times of day in the solar and lunar calculations.

The values for Cambridge, Massachusetts are as follows:

```
(calendar-nth-named-day 1 0 4 year)
(calendar-nth-named-day -1 0 10 year)
```

That is, the first 0th day (Sunday) of the fourth month (April) in the year specified by `year`, and the last Sunday of the tenth month (October) of that year. If daylight savings time were changed to start on October 1, you would set `calendar-daylight-savings-starts` to this:

```
(list 10 1 year)
```

If there is no daylight savings time at your location, or if you want all times in standard time, set `calendar-daylight-savings-starts` and `calendar-daylight-savings-ends` to `nil`.

The variable `calendar-daylight-time-offset` specifies the difference between daylight savings time and standard time, measured in minutes. The value for Cambridge, Massachusetts is 60.

The two variables `calendar-daylight-savings-starts-time` and `calendar-daylight-savings-ends-time` specify the number of minutes after midnight local time when the transition to and from daylight savings time should occur. For Cambridge, Massachusetts both variables' values are 120.

29.13 Summing Time Intervals

The timeclock feature adds up time intervals, so you can (for instance) keep track of how much time you spend working.

Use the `M-x timeclock-in` command when you start working on a project, and `M-x timeclock-out` command when you're done. Each time you do this, it adds one time interval to the record of the project.

Once you've collected data from a number of time intervals, you can use `M-x timeclock-workday-remaining` to see how much time is left to work today (assuming a typical average of 8 hours a day), and `M-x timeclock-when-to-leave` which will calculate when you're "done."

If you want Emacs to display the amount of time "left" of your workday in the mode line, either customize the `timeclock-modeline-display` variable and set its value to `t`, or invoke the `M-x timeclock-modeline-display` command.

Terminating the current Emacs session might or might not mean that you have stopped working on the project. If you'd like Emacs to ask you about this, set the value of the variable `timeclock-ask-before-exiting` to `t` (via `M-x customize`). By default, only an explicit `M-x timeclock-out` tells Emacs that the current interval is over.

The timeclock functions work by accumulating the data on a file called '`.timelog`' in your home directory. (On MS-DOS, this file is called '`_timelog`', since an initial period is not allowed in file names on MS-DOS.) You can specify a different name for this file by customizing the variable `timeclock-file`. If you edit the timeclock file manually, or if you change the value of any of timeclock's customizable variables, you should run the command `M-x timeclock-reread-log` to update the data in Emacs from the file.

30 Miscellaneous Commands

This chapter contains several brief topics that do not fit anywhere else: reading netnews, running shell commands and shell subprocesses, using a single shared Emacs for utilities that expect to run an editor as a subprocess, printing hardcopy, sorting text, narrowing display to part of the buffer, editing double-column files and binary files, saving an Emacs session for later resumption, emulating other editors, and various diversions and amusements.

30.1 Gnus

Gnus is an Emacs package primarily designed for reading and posting Usenet news. It can also be used to read and respond to messages from a number of other sources—mail, remote directories, digests, and so on.

Here we introduce Gnus and describe several basic features. For full details on Gnus, type `M-x info` and then select the Gnus manual.

To start Gnus, type `M-x gnus` ⟨RET⟩.

30.1.1 Gnus Buffers

As opposed to most normal Emacs packages, Gnus uses a number of different buffers to display information and to receive commands. The three buffers users spend most of their time in are the *group buffer*, the *summary buffer* and the *article buffer*.

The *group buffer* contains a list of groups. This is the first buffer Gnus displays when it starts up. It normally displays only the groups to which you subscribe and that contain unread articles. Use this buffer to select a specific group.

The *summary buffer* lists one line for each article in a single group. By default, the author, the subject and the line number are displayed for each article, but this is customizable, like most aspects of Gnus display. The summary buffer is created when you select a group in the group buffer, and is killed when you exit the group. Use this buffer to select an article.

The *article buffer* displays the article. In normal Gnus usage, you don't select this buffer—all useful article-oriented commands work in the summary buffer. But you can select the article buffer, and execute all Gnus commands from that buffer, if you want to.

30.1.2 When Gnus Starts Up

At startup, Gnus reads your '`.newsrc`' news initialization file and attempts to communicate with the local news server, which is a repository

of news articles. The news server need not be the same computer you are logged in on.

If you start Gnus and connect to the server, but do not see any newsgroups listed in the group buffer, type `L` or `A k` to get a listing of all the groups. Then type `u` to toggle subscription to groups.

The first time you start Gnus, Gnus subscribes you to a few selected groups. All other groups start out as *killed groups* for you; you can list them with `A k`. All new groups that subsequently come to exist at the news server become *zombie groups* for you; type `A z` to list them. You can subscribe to a group shown in these lists using the `u` command.

When you quit Gnus with `q`, it automatically records in your '`.newsrc`' and '`.newsrc.eld`' initialization files the subscribed or unsubscribed status of all groups. You should normally not edit these files manually, but you may if you know how.

30.1.3 Summary of Gnus Commands

Reading news is a two step process:

1. Choose a group in the group buffer.
2. Select articles from the summary buffer. Each article selected is displayed in the article buffer in a large window, below the summary buffer in its small window.

Each Gnus buffer has its own special commands; however, the meanings of any given key in the various Gnus buffers are usually analogous, even if not identical. Here are commands for the group and summary buffers:

`q` In the group buffer, update your '`.newsrc`' initialization file and quit Gnus.

In the summary buffer, exit the current group and return to the group buffer. Thus, typing `q` twice quits Gnus.

`L` In the group buffer, list all the groups available on your news server (except those you have killed). This may be a long list!

`l` In the group buffer, list only the groups to which you subscribe and which contain unread articles.

`u` In the group buffer, unsubscribe from (or subscribe to) the group listed in the line that point is on. When you quit Gnus by typing `q`, Gnus lists in your '`.newsrc`' file which groups you have subscribed to. The next time you start Gnus, you won't see this group, because Gnus normally displays only subscribed-to groups.

`C-k` In the group buffer, "kill" the current line's group—don't even list it in '`.newsrc`' from now on. This affects future Gnus sessions as well as the present session.

When you quit Gnus by typing `q`, Gnus writes information in the file '`.newsrc`' describing all newsgroups except those you have "killed."

⟨SPC⟩ In the group buffer, select the group on the line under the cursor and display the first unread article in that group.

In the summary buffer,

- Select the article on the line under the cursor if none is selected.
- Scroll the text of the selected article (if there is one).
- Select the next unread article if at the end of the current article.

Thus, you can move through all the articles by repeatedly typing ⟨SPC⟩.

⟨DEL⟩ In the group buffer, move point to the previous group containing unread articles.

In the summary buffer, scroll the text of the article backwards.

`n` Move point to the next unread group, or select the next unread article.

`p` Move point to the previous unread group, or select the previous unread article.

`C-n`
`C-p` Move point to the next or previous item, even if it is marked as read. This does not select the article or group on that line.

`s` In the summary buffer, do an incremental search of the current text in the article buffer, just as if you switched to the article buffer and typed `C-s`.

`M-s` *regexp* ⟨RET⟩
In the summary buffer, search forward for articles containing a match for *regexp*.

30.2 Running Shell Commands from Emacs

Emacs has commands for passing single command lines to inferior shell processes; it can also run a shell interactively with input and output to an Emacs buffer named '`*shell*`' or run a shell inside a terminal emulator window.

There is a shell implemented entirely in Emacs, documented in a separate manual. See section "Eshell " in *Eshell: The Emacs Shell*.

`M-!` *cmd* ⟨RET⟩
: Run the shell command line *cmd* and display the output (`shell-command`).

`M-|` *cmd* ⟨RET⟩
: Run the shell command line *cmd* with region contents as input; optionally replace the region with the output (`shell-command-on-region`).

`M-x shell`
: Run a subshell with input and output through an Emacs buffer. You can then give commands interactively.

`M-x term`
: Run a subshell with input and output through an Emacs buffer. You can then give commands interactively. Full terminal emulation is available.

`M-x eshell`
: Start the Emacs shell.

30.2.1 Single Shell Commands

`M-!` (`shell-command`) reads a line of text using the minibuffer and executes it as a shell command in a subshell made just for that command. Standard input for the command comes from the null device. If the shell command produces any output, the output goes into an Emacs buffer named '`*Shell Command Output*`', which is displayed in another window but not selected. A numeric argument, as in `M-1 M-!`, directs this command to insert any output into the current buffer. In that case, point is left before the output and the mark is set after the output.

If the shell command line ends in '`&`', it runs asynchronously. For a synchronous shell command, `shell-command` returns the command's exit status (0 means success), when it is called from a Lisp program.

`M-|` (`shell-command-on-region`) is like `M-!` but passes the contents of the region as the standard input to the shell command, instead of no input. If a numeric argument is used, meaning insert the output in the current buffer, then the old region is deleted first and the output replaces it as the contents of the region. It returns the command's exit status when it is called from a Lisp program.

Both `M-!` and `M-|` use `shell-file-name` to specify the shell to use. This variable is initialized based on your `SHELL` environment variable when Emacs is started. If the file name does not specify a directory, the directories in the list `exec-path` are searched; this list is initialized based on the environment variable `PATH` when Emacs is started. Your '`.emacs`' file can override either or both of these default initializations.

Both `M-!` and `M-|` wait for the shell command to complete. To stop waiting, type `C-g` to quit; that terminates the shell command with the signal `SIGINT`—the same signal that `C-c` normally generates in the shell. Emacs waits until the command actually terminates. If the shell command doesn't stop (because it ignores the `SIGINT` signal), type `C-g` again; this sends the command a `SIGKILL` signal which is impossible to ignore.

To specify a coding system for `M-!` or `M-|`, use the command `C-x` ⟨RET⟩ `c` immediately beforehand. See Section 18.8 [Specify Coding], page 227.

Error output from the command is normally intermixed with the regular output. If you set the variable `shell-command-default-error-buffer` to a string, which is a buffer name, error output is inserted before point in the buffer of that name.

30.2.2 Interactive Inferior Shell

To run a subshell interactively, putting its typescript in an Emacs buffer, use `M-x shell`. This creates (or reuses) a buffer named '`*shell*`' and runs a subshell with input coming from and output going to that buffer. That is to say, any "terminal output" from the subshell goes into the buffer, advancing point, and any "terminal input" for the subshell comes from text in the buffer. To give input to the subshell, go to the end of the buffer and type the input, terminated by ⟨RET⟩.

Emacs does not wait for the subshell to do anything. You can switch windows or buffers and edit them while the shell is waiting, or while it is running a command. Output from the subshell waits until Emacs has time to process it; this happens whenever Emacs is waiting for keyboard input or for time to elapse.

Input lines, once you submit them, are displayed using the face `comint-highlight-input`, and prompts are displayed using the face `comint-highlight-prompt`. This makes it easier to see previous input lines in the buffer. See Section 11.1 [Faces], page 103.

To make multiple subshells, rename the buffer '`*shell*`' to something different using `M-x rename-uniquely`. Then type `M-x shell` again to create a new buffer '`*shell*`' with its own subshell. If you rename this buffer as well, you can create a third one, and so on. All the subshells run independently and in parallel.

The file name used to load the subshell is the value of the variable `explicit-shell-file-name`, if that is non-`nil`. Otherwise, the environment variable `ESHELL` is used, or the environment variable `SHELL` if there is no `ESHELL`. If the file name specified is relative, the directories in the list `exec-path` are searched; this list is initialized based on the environment variable `PATH` when Emacs is started. Your '`.emacs`' file can override either or both of these default initializations.

Emacs sends the new shell the contents of the file '`~/.emacs_`*shellname*' as input, if it exists, where *shellname* is the name of the file that the shell was loaded from. For example, if you use bash, the file sent to it is '`~/.emacs_bash`'.

To specify a coding system for the shell, you can use the command `C-x` ⟨RET⟩ `c` immediately before `M-x shell`. You can also specify a coding system after starting the shell by using `C-x` ⟨RET⟩ `p` in the shell buffer. See Section 18.8 [Specify Coding], page 227.

Emacs defines the environment variable `EMACS` in the subshell, with value `t`. A shell script can check this variable to determine whether it has been run from an Emacs subshell.

30.2.3 Shell Mode

Shell buffers use Shell mode, which defines several special keys attached to the `C-c` prefix. They are chosen to resemble the usual editing and job control characters present in shells that are not under Emacs, except that you must type `C-c` first. Here is a complete list of the special key bindings of Shell mode:

⟨RET⟩ At end of buffer send line as input; otherwise, copy current line to end of buffer and send it (`comint-send-input`). When a line is copied, any prompt at the beginning if the line (text output by programs preceding your input) is omitted. See also `comint-use-prompt-regexp-instead-of-fields`).

⟨TAB⟩ Complete the command name or file name before point in the shell buffer (`comint-dynamic-complete`). ⟨TAB⟩ also completes history references (see Section 30.2.4.3 [History References], page 432) and environment variable names.

The variable `shell-completion-fignore` specifies a list of file name extensions to ignore in Shell mode completion. The default setting ignores file names ending in '`~`', '`#`' or '`%`'. Other related Comint modes use the variable `comint-completion-fignore` instead.

`M-?` Display temporarily a list of the possible completions of the file name before point in the shell buffer (`comint-dynamic-list-filename-completions`).

`C-d` Either delete a character or send EOF (`comint-delchar-or-maybe-eof`). Typed at the end of the shell buffer, `C-d` sends EOF to the subshell. Typed at any other position in the buffer, `C-d` deletes a character as usual.

`C-c C-a` Move to the beginning of the line, but after the prompt if any (`comint-bol`). If you repeat this command twice in a row, the

second time it moves back to the process mark, which is the beginning of the input that you have not yet sent to the subshell. (Normally that is the same place—the end of the prompt on this line—but after `C-c` ⟨SPC⟩ the process mark may be in a previous line.)

`C-c` ⟨SPC⟩ Accumulate multiple lines of input, then send them together. This command inserts a newline before point, but does not send the preceding text as input to the subshell—at least, not yet. Both lines, the one before this newline and the one after, will be sent together (along with the newline that separates them), when you type ⟨RET⟩.

`C-c C-u` Kill all text pending at end of buffer to be sent as input (`comint-kill-input`).

`C-c C-w` Kill a word before point (`backward-kill-word`).

`C-c C-c` Interrupt the shell or its current subjob if any (`comint-interrupt-subjob`). This command also kills any shell input pending in the shell buffer and not yet sent.

`C-c C-z` Stop the shell or its current subjob if any (`comint-stop-subjob`). This command also kills any shell input pending in the shell buffer and not yet sent.

`C-c C-\` Send quit signal to the shell or its current subjob if any (`comint-quit-subjob`). This command also kills any shell input pending in the shell buffer and not yet sent.

`C-c C-o` Delete the last batch of output from a shell command (`comint-delete-output`). This is useful if a shell command spews out lots of output that just gets in the way. This command used to be called `comint-kill-output`.

`C-c C-s` Write the last batch of output from a shell command to a file (`comint-write-output`). With a prefix argument, the file is appended to instead. Any prompt at the end of the output is not written.

`C-c C-r`
`C-M-l` Scroll to display the beginning of the last batch of output at the top of the window; also move the cursor there (`comint-show-output`).

`C-c C-e` Scroll to put the end of the buffer at the bottom of the window (`comint-show-maximum-output`).

`C-c C-f` Move forward across one shell command, but not beyond the current line (`shell-forward-command`). The variable `shell-command-regexp` specifies how to recognize the end of a command.

`C-c C-b` Move backward across one shell command, but not beyond the current line (`shell-backward-command`).

`C-c C-l` Display the buffer's history of shell commands in another window (`comint-dynamic-list-input-ring`).

`M-x dirs` Ask the shell what its current directory is, so that Emacs can agree with the shell.

`M-x send-invisible` ⟨RET⟩ *text* ⟨RET⟩
Send *text* as input to the shell, after reading it without echoing. This is useful when a shell command runs a program that asks for a password.

Alternatively, you can arrange for Emacs to notice password prompts and turn off echoing for them, as follows:

```
(add-hook 'comint-output-filter-functions
          'comint-watch-for-password-prompt)
```

`M-x comint-continue-subjob`
Continue the shell process. This is useful if you accidentally suspend the shell process.[1]

`M-x comint-strip-ctrl-m`
Discard all control-M characters from the current group of shell output. The most convenient way to use this command is to make it run automatically when you get output from the subshell. To do that, evaluate this Lisp expression:

```
(add-hook 'comint-output-filter-functions
          'comint-strip-ctrl-m)
```

`M-x comint-truncate-buffer`
This command truncates the shell buffer to a certain maximum number of lines, specified by the variable `comint-buffer-maximum-size`. Here's how to do this automatically each time you get output from the subshell:

```
(add-hook 'comint-output-filter-functions
          'comint-truncate-buffer)
```

Shell mode also customizes the paragraph commands so that only shell prompts start new paragraphs. Thus, a paragraph consists of an input command plus the output that follows it in the buffer.

Shell mode is a derivative of Comint mode, a general-purpose mode for communicating with interactive subprocesses. Most of the features of Shell mode actually come from Comint mode, as you can see from the command

[1] You should not suspend the shell process. Suspending a subjob of the shell is a completely different matter—that is normal practice, but you must use the shell to continue the subjob; this command won't do it.

names listed above. The special features of Shell mode include the directory tracking feature, and a few user commands.

Other Emacs features that use variants of Comint mode include GUD (see Section 23.5 [Debuggers], page 334) and `M-x run-lisp` (see Section 23.10 [External Lisp], page 342).

You can use `M-x comint-run` to execute any program of your choice in a subprocess using unmodified Comint mode—without the specializations of Shell mode.

30.2.4 Shell Command History

Shell buffers support three ways of repeating earlier commands. You can use the same keys used in the minibuffer; these work much as they do in the minibuffer, inserting text from prior commands while point remains always at the end of the buffer. You can move through the buffer to previous inputs in their original place, then resubmit them or copy them to the end. Or you can use a '`!`'-style history reference.

30.2.4.1 Shell History Ring

`M-p` Fetch the next earlier old shell command.

`M-n` Fetch the next later old shell command.

`M-r` *regexp* ⟨RET⟩
`M-s` *regexp* ⟨RET⟩
Search backwards or forwards for old shell commands that match *regexp*.

`C-c C-x` (Shell mode)
Fetch the next subsequent command from the history.

Shell buffers provide a history of previously entered shell commands. To reuse shell commands from the history, use the editing commands `M-p`, `M-n`, `M-r` and `M-s`. These work just like the minibuffer history commands except that they operate on the text at the end of the shell buffer, where you would normally insert text to send to the shell.

`M-p` fetches an earlier shell command to the end of the shell buffer. Successive use of `M-p` fetches successively earlier shell commands, each replacing any text that was already present as potential shell input. `M-n` does likewise except that it finds successively more recent shell commands from the buffer.

The history search commands `M-r` and `M-s` read a regular expression and search through the history for a matching command. Aside from the choice of which command to fetch, they work just like `M-p` and `M-r`. If you enter an empty regexp, these commands reuse the same regexp used last time.

When you find the previous input you want, you can resubmit it by typing ⟨RET⟩, or you can edit it first and then resubmit it if you wish.

Often it is useful to reexecute several successive shell commands that were previously executed in sequence. To do this, first find and reexecute the first command of the sequence. Then type `C-c C-x`; that will fetch the following command—the one that follows the command you just repeated. Then type ⟨RET⟩ to reexecute this command. You can reexecute several successive commands by typing `C-c C-x` ⟨RET⟩ over and over.

These commands get the text of previous shell commands from a special history list, not from the shell buffer itself. Thus, editing the shell buffer, or even killing large parts of it, does not affect the history that these commands access.

Some shells store their command histories in files so that you can refer to previous commands from previous shell sessions. Emacs reads the command history file for your chosen shell, to initialize its own command history. The file name is '`~/.bash_history`' for bash, '`~/.sh_history`' for ksh, and '`~/.history`' for other shells.

30.2.4.2 Shell History Copying

`C-c C-p` Move point to the previous prompt (`comint-previous-prompt`).

`C-c C-n` Move point to the following prompt (`comint-next-prompt`).

`C-c` ⟨RET⟩ Copy the input command which point is in, inserting the copy at the end of the buffer (`comint-copy-old-input`). This is useful if you move point back to a previous command. After you copy the command, you can submit the copy as input with ⟨RET⟩. If you wish, you can edit the copy before resubmitting it.

Moving to a previous input and then copying it with `C-c` ⟨RET⟩ produces the same results—the same buffer contents—that you would get by using `M-p` enough times to fetch that previous input from the history list. However, `C-c` ⟨RET⟩ copies the text from the buffer, which can be different from what is in the history list if you edit the input text in the buffer after it has been sent.

30.2.4.3 Shell History References

Various shells including csh and bash support *history references* that begin with '`!`' and '`^`'. Shell mode recognizes these constructs, and can perform the history substitution for you.

If you insert a history reference and type ⟨TAB⟩, this searches the input history for a matching command, performs substitution if necessary, and

places the result in the buffer in place of the history reference. For example, you can fetch the most recent command beginning with 'mv' with ! m v ⟨TAB⟩. You can edit the command if you wish, and then resubmit the command to the shell by typing ⟨RET⟩.

Shell mode can optionally expand history references in the buffer when you send them to the shell. To request this, set the variable comint-input-autoexpand to input. You can make ⟨SPC⟩ perform history expansion by binding ⟨SPC⟩ to the command comint-magic-space.

Shell mode recognizes history references when they follow a prompt. Normally, any text output by a program at the beginning of an input line is considered a prompt. However, if the variable comint-use-prompt-regexp-instead-of-fields is non-nil, then Comint mode uses a regular expression to recognize prompts. In general, the variable comint-prompt-regexp specifies the regular expression; Shell mode uses the variable shell-prompt-pattern to set up comint-prompt-regexp in the shell buffer.

30.2.5 Directory Tracking

Shell mode keeps track of 'cd', 'pushd' and 'popd' commands given to the inferior shell, so it can keep the '*shell*' buffer's default directory the same as the shell's working directory. It recognizes these commands syntactically, by examining lines of input that are sent.

If you use aliases for these commands, you can tell Emacs to recognize them also. For example, if the value of the variable shell-pushd-regexp matches the beginning of a shell command line, that line is regarded as a pushd command. Change this variable when you add aliases for 'pushd'. Likewise, shell-popd-regexp and shell-cd-regexp are used to recognize commands with the meaning of 'popd' and 'cd'. These commands are recognized only at the beginning of a shell command line.

If Emacs gets an error while trying to handle what it believes is a 'cd', 'pushd' or 'popd' command, it runs the hook shell-set-directory-error-hook (see Section 31.2.3 [Hooks], page 465).

If Emacs gets confused about changes in the current directory of the subshell, use the command M-x dirs to ask the shell what its current directory is. This command works for shells that support the most common command syntax; it may not work for unusual shells.

You can also use M-x dirtrack-mode to enable (or disable) an alternative and more aggressive method of tracking changes in the current directory.

30.2.6 Shell Mode Options

If the variable `comint-scroll-to-bottom-on-input` is non-`nil`, insertion and yank commands scroll the selected window to the bottom before inserting.

If `comint-scroll-show-maximum-output` is non-`nil`, then scrolling due to arrival of output tries to place the last line of text at the bottom line of the window, so as to show as much useful text as possible. (This mimics the scrolling behavior of many terminals.) The default is `nil`.

By setting `comint-scroll-to-bottom-on-output`, you can opt for having point jump to the end of the buffer whenever output arrives—no matter where in the buffer point was before. If the value is `this`, point jumps in the selected window. If the value is `all`, point jumps in each window that shows the comint buffer. If the value is `other`, point jumps in all nonselected windows that show the current buffer. The default value is `nil`, which means point does not jump to the end.

The variable `comint-input-ignoredups` controls whether successive identical inputs are stored in the input history. A non-`nil` value means to omit an input that is the same as the previous input. The default is `nil`, which means to store each input even if it is equal to the previous input.

Three variables customize file name completion. The variable `comint-completion-addsuffix` controls whether completion inserts a space or a slash to indicate a fully completed file or directory name (non-`nil` means do insert a space or slash). `comint-completion-recexact`, if non-`nil`, directs ⟨TAB⟩ to choose the shortest possible completion if the usual Emacs completion algorithm cannot add even a single character. `comint-completion-autolist`, if non-`nil`, says to list all the possible completions whenever completion is not exact.

The command `comint-dynamic-complete-variable` does variable-name completion using the environment variables as set within Emacs. The variables controlling file name completion apply to variable-name completion too. This command is normally available through the menu bar.

Command completion normally considers only executable files. If you set `shell-command-execonly` to `nil`, it considers nonexecutable files as well.

You can configure the behavior of '`pushd`'. Variables control whether '`pushd`' behaves like '`cd`' if no argument is given (`shell-pushd-tohome`), pop rather than rotate with a numeric argument (`shell-pushd-dextract`), and only add directories to the directory stack if they are not already on it (`shell-pushd-dunique`). The values you choose should match the underlying shell, of course.

30.2.7 Emacs Terminal Emulator

To run a subshell in a terminal emulator, putting its typescript in an Emacs buffer, use `M-x term`. This creates (or reuses) a buffer named

'`*term*`', and runs a subshell with input coming from your keyboard, and output going to that buffer.

The terminal emulator uses Term mode, which has two input modes. In line mode, Term basically acts like Shell mode; see Section 30.2.3 [Shell Mode], page 428.

In char mode, each character is sent directly to the inferior subshell, as "terminal input." Any "echoing" of your input is the responsibility of the subshell. The sole exception is the terminal escape character, which by default is `C-c` (see Section 30.2.8 [Term Mode], page 435). Any "terminal output" from the subshell goes into the buffer, advancing point.

Some programs (such as Emacs itself) need to control the appearance on the terminal screen in detail. They do this by sending special control codes. The exact control codes needed vary from terminal to terminal, but nowadays most terminals and terminal emulators (including `xterm`) understand the ANSI-standard (VT100-style) escape sequences. Term mode recognizes these escape sequences, and handles each one appropriately, changing the buffer so that the appearance of the window matches what it would be on a real terminal. You can actually run Emacs inside an Emacs Term window.

The file name used to load the subshell is determined the same way as for Shell mode. To make multiple terminal emulators, rename the buffer '`*term*`' to something different using `M-x rename-uniquely`, just as with Shell mode.

Unlike Shell mode, Term mode does not track the current directory by examining your input. But some shells can tell Term what the current directory is. This is done automatically by `bash` version 1.15 and later.

30.2.8 Term Mode

The terminal emulator uses Term mode, which has two input modes. In line mode, Term basically acts like Shell mode; see Section 30.2.3 [Shell Mode], page 428. In char mode, each character is sent directly to the inferior subshell, except for the Term escape character, normally `C-c`.

To switch between line and char mode, use these commands:

`C-c C-k` Switch to line mode. Do nothing if already in line mode.

`C-c C-j` Switch to char mode. Do nothing if already in char mode.

The following commands are only available in char mode:

`C-c C-c` Send a literal ⟨C-c⟩ to the sub-shell.

`C-c C-x` A prefix command to access the global ⟨C-x⟩ commands conveniently. For example, `C-c C-x o` invokes the global binding of `C-x o`, which is normally '`other-window`'.

30.2.9 Page-At-A-Time Output

Term mode has a page-at-a-time feature. When enabled it makes output pause at the end of each screenful.

`C-c C-q` Toggle the page-at-a-time feature. This command works in both line and char modes. When page-at-a-time is enabled, the mode-line displays the word '`page`'.

With page-at-a-time enabled, whenever Term receives more than a screenful of output since your last input, it pauses, displaying '`**MORE**`' in the mode-line. Type ⟨SPC⟩ to display the next screenful of output. Type `?` to see your other options. The interface is similar to the Unix `more` program.

30.2.10 Remote Host Shell

You can login to a remote computer, using whatever commands you would from a regular terminal (e.g. using the `telnet` or `rlogin` commands), from a Term window.

A program that asks you for a password will normally suppress echoing of the password, so the password will not show up in the buffer. This will happen just as if you were using a real terminal, if the buffer is in char mode. If it is in line mode, the password is temporarily visible, but will be erased when you hit return. (This happens automatically; there is no special password processing.)

When you log in to a different machine, you need to specify the type of terminal your using. Terminal types '`ansi`' or '`vt100`' will work on most systems.

30.3 Using Emacs as a Server

Various programs such as `mail` can invoke your choice of editor to edit a particular piece of text, such as a message that you are sending. By convention, most of these programs use the environment variable `EDITOR` to specify which editor to run. If you set `EDITOR` to '`emacs`', they invoke Emacs—but in an inconvenient fashion, by starting a new, separate Emacs process. This is inconvenient because it takes time and because the new Emacs process doesn't share the buffers in the existing Emacs process.

You can arrange to use your existing Emacs process as the editor for programs like `mail` by using the Emacs client and Emacs server programs. Here is how.

First, the preparation. Within Emacs, call the function `server-start`. (Your '`.emacs`' file can do this automatically if you add the expression

`(server-start)` to it.) Then, outside Emacs, set the `EDITOR` environment variable to '`emacsclient`'. (Note that some programs use a different environment variable; for example, to make TEX use '`emacsclient`', you should set the `TEXEDIT` environment variable to '`emacsclient +%d %s`'.)

Then, whenever any program invokes your specified `EDITOR` program, the effect is to send a message to your principal Emacs telling it to visit a file. (That's what the program `emacsclient` does.) Emacs displays the buffer immediately and you can immediately begin editing it.

When you've finished editing that buffer, type `C-x #` (`server-edit`). This saves the file and sends a message back to the `emacsclient` program telling it to exit. The programs that use `EDITOR` wait for the "editor" (actually, `emacsclient`) to exit. `C-x #` also checks for other pending external requests to edit various files, and selects the next such file.

You can switch to a server buffer manually if you wish; you don't have to arrive at it with `C-x #`. But `C-x #` is the only way to say that you are "finished" with one.

Finishing with a server buffer also kills the buffer, unless it already existed in the Emacs session before the server asked to create it. However, if you set `server-kill-new-buffers` to `nil`, then a different criterion is used: finishing with a server buffer kills it if the file name matches the regular expression `server-temp-file-regexp`. This is set up to distinguish certain "temporary" files.

If you set the variable `server-window` to a window or a frame, `C-x #` displays the server buffer in that window or in that frame.

While `mail` or another application is waiting for `emacsclient` to finish, `emacsclient` does not read terminal input. So the terminal that `mail` was using is effectively blocked for the duration. In order to edit with your principal Emacs, you need to be able to use it without using that terminal. There are three ways to do this:

- Using a window system, run `mail` and the principal Emacs in two separate windows. While `mail` is waiting for `emacsclient`, the window where it was running is blocked, but you can use Emacs by switching windows.
- Using virtual terminals, run `mail` in one virtual terminal and run Emacs in another.
- Use Shell mode or Term mode in Emacs to run the other program such as `mail`; then, `emacsclient` blocks only the subshell under Emacs, and you can still use Emacs to edit the file.

If you run `emacsclient` with the option '`--no-wait`', it returns immediately without waiting for you to "finish" the buffer in Emacs. Note that server buffers created in this way are not killed automatically when you finish with them.

30.4 Invoking `emacsclient`

To run the `emacsclient` program, specify file names as arguments, and optionally line numbers as well. Do it like this:

```
emacsclient {[+line] filename}...
```

This tells Emacs to visit each of the specified files; if you specify a line number for a certain file, Emacs moves to that line in the file.

Ordinarily, `emacsclient` does not return until you use the `C-x #` command on each of these buffers. When that happens, Emacs sends a message to the `emacsclient` program telling it to return.

But if you use the option '`-n`' or '`--no-wait`' when running `emacsclient`, then it returns immediately. (You can take as long as you like to edit the files in Emacs.)

The option '`--alternate-editor=`*command*' is useful when running `emacsclient` in a script. It specifies a command to run if `emacsclient` fails to contact Emacs. For example, the following setting for the *EDITOR* environment variable will always give an editor, even if Emacs is not running:

```
EDITOR="emacsclient --alternate-editor vi +%d %s"
```

The environment variable *ALTERNATE_EDITOR* has the same effect, but the value of the '`--alternate-editor`' takes precedence.

Alternatively, the file '`etc/emacs.bash`' defines a bash function which will communicate with a running Emacs server, or start one if none exists.

30.5 Hardcopy Output

The Emacs commands for making hardcopy let you print either an entire buffer or just part of one, either with or without page headers. See also the hardcopy commands of Dired (see Section 14.10 [Misc File Ops], page 181) and the diary (see Section 29.10.1 [Diary Commands], page 414).

`M-x print-buffer`
: Print hardcopy of current buffer with page headings containing the file name and page number.

`M-x lpr-buffer`
: Print hardcopy of current buffer without page headings.

`M-x print-region`
: Like `print-buffer` but print only the current region.

`M-x lpr-region`
: Like `lpr-buffer` but print only the current region.

The hardcopy commands (aside from the Postscript commands) pass extra switches to the `lpr` program based on the value of the variable `lpr-switches`. Its value should be a list of strings, each string an option starting with '`-`'. For example, to specify a line width of 80 columns for all the printing you do in Emacs, set `lpr-switches` like this:

```
(setq lpr-switches '("-w80"))
```

You can specify the printer to use by setting the variable `printer-name`.

The variable `lpr-command` specifies the name of the printer program to run; the default value depends on your operating system type. On most systems, the default is `"lpr"`. The variable `lpr-headers-switches` similarly specifies the extra switches to use to make page headers. The variable `lpr-add-switches` controls whether to supply '`-T`' and '`-J`' options (suitable for `lpr`) to the printer program: `nil` means don't add them. `lpr-add-switches` should be `nil` if your printer program is not compatible with `lpr`.

30.6 PostScript Hardcopy

These commands convert buffer contents to PostScript, either printing it or leaving it in another Emacs buffer.

`M-x ps-print-buffer`
: Print hardcopy of the current buffer in PostScript form.

`M-x ps-print-region`
: Print hardcopy of the current region in PostScript form.

`M-x ps-print-buffer-with-faces`
: Print hardcopy of the current buffer in PostScript form, showing the faces used in the text by means of PostScript features.

`M-x ps-print-region-with-faces`
: Print hardcopy of the current region in PostScript form, showing the faces used in the text.

`M-x ps-spool-buffer`
: Generate PostScript for the current buffer text.

`M-x ps-spool-region`
: Generate PostScript for the current region.

`M-x ps-spool-buffer-with-faces`
: Generate PostScript for the current buffer, showing the faces used.

`M-x ps-spool-region-with-faces`
: Generate PostScript for the current region, showing the faces used.

`M-x handwrite`
: Generates/prints PostScript for the current buffer as if handwritten.

The PostScript commands, `ps-print-buffer` and `ps-print-region`, print buffer contents in PostScript form. One command prints the entire buffer; the other, just the region. The corresponding '`-with-faces`' commands, `ps-print-buffer-with-faces` and `ps-print-region-with-faces`, use PostScript features to show the faces (fonts and colors) in the text properties of the text being printed.

If you are using a color display, you can print a buffer of program code with color highlighting by turning on Font-Lock mode in that buffer, and using `ps-print-buffer-with-faces`.

The commands whose names have '`spool`' instead of '`print`' generate the PostScript output in an Emacs buffer instead of sending it to the printer.

`M-x handwrite` is more frivolous. It generates a PostScript rendition of the current buffer as a cursive handwritten document. It can be customized in group `handwrite`.

30.7 Variables for PostScript Hardcopy

All the PostScript hardcopy commands use the variables `ps-lpr-command` and `ps-lpr-switches` to specify how to print the output. `ps-lpr-command` specifies the command name to run, `ps-lpr-switches` specifies command line options to use, and `ps-printer-name` specifies the printer. If you don't set the first two variables yourself, they take their initial values from `lpr-command` and `lpr-switches`. If `ps-printer-name` is `nil`, `printer-name` is used.

The variable `ps-print-header` controls whether these commands add header lines to each page—set it to `nil` to turn headers off.

If your printer doesn't support colors, you should turn off color processing by setting `ps-print-color-p` to `nil`. By default, if the display supports colors, Emacs produces hardcopy output with color information; on black-and-white printers, colors are emulated with shades of gray. This might produce illegible output, even if your screen colors only use shades of gray.

By default, PostScript printing ignores the background colors of the faces, unless the variable `ps-use-face-background` is non-`nil`. This is to avoid unwanted interference with the zebra stripes and background image/text.

The variable `ps-paper-type` specifies which size of paper to format for; legitimate values include `a4`, `a3`, `a4small`, `b4`, `b5`, `executive`, `ledger`, `legal`, `letter`, `letter-small`, `statement`, `tabloid`. The default is `letter`. You can define additional paper sizes by changing the variable `ps-page-dimensions-database`.

The variable `ps-landscape-mode` specifies the orientation of printing on the page. The default is `nil`, which stands for "portrait" mode. Any non-`nil` value specifies "landscape" mode.

The variable `ps-number-of-columns` specifies the number of columns; it takes effect in both landscape and portrait mode. The default is 1.

The variable `ps-font-family` specifies which font family to use for printing ordinary text. Legitimate values include `Courier`, `Helvetica`, `NewCenturySchlbk`, `Palatino` and `Times`. The variable `ps-font-size` specifies the size of the font for ordinary text. It defaults to 8.5 points.

Many other customization variables for these commands are defined and described in the Lisp file '`ps-print.el`'.

30.8 Sorting Text

Emacs provides several commands for sorting text in the buffer. All operate on the contents of the region (the text between point and the mark). They divide the text of the region into many *sort records*, identify a *sort key* for each record, and then reorder the records into the order determined by the sort keys. The records are ordered so that their keys are in alphabetical order, or, for numeric sorting, in numeric order. In alphabetic sorting, all upper-case letters 'A' through 'Z' come before lower-case 'a', in accord with the ASCII character sequence.

The various sort commands differ in how they divide the text into sort records and in which part of each record is used as the sort key. Most of the commands make each line a separate sort record, but some commands use paragraphs or pages as sort records. Most of the sort commands use each entire sort record as its own sort key, but some use only a portion of the record as the sort key.

`M-x sort-lines`
: Divide the region into lines, and sort by comparing the entire text of a line. A numeric argument means sort into descending order.

`M-x sort-paragraphs`
: Divide the region into paragraphs, and sort by comparing the entire text of a paragraph (except for leading blank lines). A numeric argument means sort into descending order.

`M-x sort-pages`
: Divide the region into pages, and sort by comparing the entire text of a page (except for leading blank lines). A numeric argument means sort into descending order.

`M-x sort-fields`
: Divide the region into lines, and sort by comparing the contents of one field in each line. Fields are defined as separated by whitespace, so the first run of consecutive non-whitespace characters in a line constitutes field 1, the second such run constitutes field 2, etc.

  Specify which field to sort by with a numeric argument: 1 to sort by field 1, etc. A negative argument means count fields from the right instead of from the left; thus, minus 1 means sort by the last field. If several lines have identical contents in the field being sorted, they keep same relative order that they had in the original buffer.

`M-x sort-numeric-fields`
: Like `M-x sort-fields` except the specified field is converted to an integer for each line, and the numbers are compared. '`10`' comes before '`2`' when considered as text, but after it when considered as a number. By default, numbers are interpreted according to `sort-numeric-base`, but numbers beginning with '`0x`' or '`0`' are interpreted as hexadecimal and octal, respectively.

`M-x sort-columns`
: Like `M-x sort-fields` except that the text within each line used for comparison comes from a fixed range of columns. See below for an explanation.

`M-x reverse-region`
: Reverse the order of the lines in the region. This is useful for sorting into descending order by fields or columns, since those sort commands do not have a feature for doing that.

For example, if the buffer contains this:

```
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.
```

applying `M-x sort-lines` to the entire buffer produces this:

```
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
```

where the upper-case '`O`' sorts before all lower-case letters. If you use `C-u 2 M-x sort-fields` instead, you get this:

```
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
```

where the sort keys were 'Emacs', 'If', 'buffer', 'systems' and 'the'.

M-x sort-columns requires more explanation. You specify the columns by putting point at one of the columns and the mark at the other column. Because this means you cannot put point or the mark at the beginning of the first line of the text you want to sort, this command uses an unusual definition of "region": all of the line point is in is considered part of the region, and so is all of the line the mark is in, as well as all the lines in between.

For example, to sort a table by information found in columns 10 to 15, you could put the mark on column 10 in the first line of the table, and point on column 15 in the last line of the table, and then run sort-columns. Equivalently, you could run it with the mark on column 15 in the first line and point on column 10 in the last line.

This can be thought of as sorting the rectangle specified by point and the mark, except that the text on each line to the left or right of the rectangle moves along with the text inside the rectangle. See Section 9.4 [Rectangles], page 93.

Many of the sort commands ignore case differences when comparing, if sort-fold-case is non-nil.

30.9 Narrowing

Narrowing means focusing in on some portion of the buffer, making the rest temporarily inaccessible. The portion which you can still get to is called the *accessible portion*. Canceling the narrowing, which makes the entire buffer once again accessible, is called *widening*. The amount of narrowing in effect in a buffer at any time is called the buffer's *restriction*.

Narrowing can make it easier to concentrate on a single subroutine or paragraph by eliminating clutter. It can also be used to restrict the range of operation of a replace command or repeating keyboard macro.

C-x n n Narrow down to between point and mark (narrow-to-region).

C-x n w Widen to make the entire buffer accessible again (widen).

C-x n p Narrow down to the current page (narrow-to-page).

C-x n d Narrow down to the current defun (narrow-to-defun).

When you have narrowed down to a part of the buffer, that part appears to be all there is. You can't see the rest, you can't move into it (motion commands won't go outside the accessible part), you can't change it in any way. However, it is not gone, and if you save the file all the inaccessible

text will be saved. The word '`Narrow`' appears in the mode line whenever narrowing is in effect.

The primary narrowing command is `C-x n n` (`narrow-to-region`). It sets the current buffer's restrictions so that the text in the current region remains accessible but all text before the region or after the region is inaccessible. Point and mark do not change.

Alternatively, use `C-x n p` (`narrow-to-page`) to narrow down to the current page. See Section 21.4 [Pages], page 247, for the definition of a page. `C-x n d` (`narrow-to-defun`) narrows down to the defun containing point (see Section 22.4 [Defuns], page 277).

The way to cancel narrowing is to widen with `C-x n w` (`widen`). This makes all text in the buffer accessible again.

You can get information on what part of the buffer you are narrowed down to using the `C-x =` command. See Section 4.9 [Position Info], page 50.

Because narrowing can easily confuse users who do not understand it, `narrow-to-region` is normally a disabled command. Attempting to use this command asks for confirmation and gives you the option of enabling it; if you enable the command, confirmation will no longer be required for it. See Section 31.4.11 [Disabling], page 484.

30.10 Two-Column Editing

Two-column mode lets you conveniently edit two side-by-side columns of text. It uses two side-by-side windows, each showing its own buffer.

There are three ways to enter two-column mode:

⟨F2⟩ `2` or `C-x 6 2`
: Enter two-column mode with the current buffer on the left, and on the right, a buffer whose name is based on the current buffer's name (`2C-two-columns`). If the right-hand buffer doesn't already exist, it starts out empty; the current buffer's contents are not changed.

  This command is appropriate when the current buffer is empty or contains just one column and you want to add another column.

⟨F2⟩ `s` or `C-x 6 s`
: Split the current buffer, which contains two-column text, into two buffers, and display them side by side (`2C-split`). The current buffer becomes the left-hand buffer, but the text in the right-hand column is moved into the right-hand buffer. The current column specifies the split point. Splitting starts with the current line and continues to the end of the buffer.

This command is appropriate when you have a buffer that already contains two-column text, and you wish to separate the columns temporarily.

⟨F2⟩ `b` *buffer* ⟨RET⟩
`C-x 6 b` *buffer* ⟨RET⟩
Enter two-column mode using the current buffer as the left-hand buffer, and using buffer *buffer* as the right-hand buffer (`2C-associate-buffer`).

⟨F2⟩ `s` or `C-x 6 s` looks for a column separator, which is a string that appears on each line between the two columns. You can specify the width of the separator with a numeric argument to ⟨F2⟩ `s`; that many characters, before point, constitute the separator string. By default, the width is 1, so the column separator is the character before point.

When a line has the separator at the proper place, ⟨F2⟩ `s` puts the text after the separator into the right-hand buffer, and deletes the separator. Lines that don't have the column separator at the proper place remain unsplit; they stay in the left-hand buffer, and the right-hand buffer gets an empty line to correspond. (This is the way to write a line that "spans both columns while in two-column mode": write it in the left-hand buffer, and put an empty line in the right-hand buffer.)

The command `C-x 6` ⟨RET⟩ or ⟨F2⟩ ⟨RET⟩ (`2C-newline`) inserts a newline in each of the two buffers at corresponding positions. This is the easiest way to add a new line to the two-column text while editing it in split buffers.

When you have edited both buffers as you wish, merge them with ⟨F2⟩ `1` or `C-x 6 1` (`2C-merge`). This copies the text from the right-hand buffer as a second column in the other buffer. To go back to two-column editing, use ⟨F2⟩ `s`.

Use ⟨F2⟩ `d` or `C-x 6 d` to dissociate the two buffers, leaving each as it stands (`2C-dissociate`). If the other buffer, the one not current when you type ⟨F2⟩ `d`, is empty, ⟨F2⟩ `d` kills it.

30.11 Editing Binary Files

There is a special major mode for editing binary files: Hexl mode. To use it, use `M-x hexl-find-file` instead of `C-x C-f` to visit the file. This command converts the file's contents to hexadecimal and lets you edit the translation. When you save the file, it is converted automatically back to binary.

You can also use `M-x hexl-mode` to translate an existing buffer into hex. This is useful if you visit a file normally and then discover it is a binary file.

Ordinary text characters overwrite in Hexl mode. This is to reduce the risk of accidentally spoiling the alignment of data in the file. There are

special commands for insertion. Here is a list of the commands of Hexl mode:

`C-M-d` Insert a byte with a code typed in decimal.

`C-M-o` Insert a byte with a code typed in octal.

`C-M-x` Insert a byte with a code typed in hex.

`C-x [` Move to the beginning of a 1k-byte "page."

`C-x ]` Move to the end of a 1k-byte "page."

`M-g` Move to an address specified in hex.

`M-j` Move to an address specified in decimal.

`C-c C-c` Leave Hexl mode, going back to the major mode this buffer had before you invoked `hexl-mode`.

Other Hexl commands let you insert strings (sequences) of binary bytes, move by short's or int's, etc.; type `C-h a hexl- RET` for details.

30.12 Saving Emacs Sessions

You can use the Desktop library to save the state of Emacs from one session to another. Saving the state means that Emacs starts up with the same set of buffers, major modes, buffer positions, and so on that the previous Emacs session had.

To use Desktop, you should use the Customization buffer (see Section 31.2.2 [Easy Customization], page 459) to set `desktop-enable` to a non-`nil` value, or add these lines at the end of your '`.emacs`' file:

```
(desktop-load-default)
(desktop-read)
```

The first time you save the state of the Emacs session, you must do it manually, with the command `M-x desktop-save`. Once you have done that, exiting Emacs will save the state again—not only the present Emacs session, but also subsequent sessions. You can also save the state at any time, without exiting Emacs, by typing `M-x desktop-save` again.

In order for Emacs to recover the state from a previous session, you must start it with the same current directory as you used when you started the previous session. This is because `desktop-read` looks in the current directory for the file to read. This means that you can have separate saved sessions in different directories; the directory in which you start Emacs will control which saved session to use.

The variable `desktop-files-not-to-save` controls which files are excluded from state saving. Its value is a regular expression that matches the files to exclude. By default, remote (ftp-accessed) files are excluded; this is

because visiting them again in the subsequent session would be slow. If you want to include these files in state saving, set `desktop-files-not-to-save` to `"^$"`. See Section 14.12 [Remote Files], page 183.

The Saveplace library provides a simpler feature that records your position in each file when you kill its buffer (or kill Emacs), and jumps to the same position when you visit the file again (even in another Emacs session). Use `M-x toggle-save-place` to turn on place-saving in a given file. Customize the option `save-place` to turn it on for all files in each session.

30.13 Recursive Editing Levels

A *recursive edit* is a situation in which you are using Emacs commands to perform arbitrary editing while in the middle of another Emacs command. For example, when you type `C-r` inside of a `query-replace`, you enter a recursive edit in which you can change the current buffer. On exiting from the recursive edit, you go back to the `query-replace`.

Exiting the recursive edit means returning to the unfinished command, which continues execution. The command to exit is `C-M-c` (`exit-recursive-edit`).

You can also *abort* the recursive edit. This is like exiting, but also quits the unfinished command immediately. Use the command `C-]` (`abort-recursive-edit`) to do this. See Section 32.1 [Quitting], page 491.

The mode line shows you when you are in a recursive edit by displaying square brackets around the parentheses that always surround the major and minor mode names. Every window's mode line shows this, in the same way, since being in a recursive edit is true of Emacs as a whole rather than any particular window or buffer.

It is possible to be in recursive edits within recursive edits. For example, after typing `C-r` in a `query-replace`, you may type a command that enters the debugger. This begins a recursive editing level for the debugger, within the recursive editing level for `C-r`. Mode lines display a pair of square brackets for each recursive editing level currently in progress.

Exiting the inner recursive edit (such as, with the debugger `c` command) resumes the command running in the next level up. When that command finishes, you can then use `C-M-c` to exit another recursive editing level, and so on. Exiting applies to the innermost level only. Aborting also gets out of only one level of recursive edit; it returns immediately to the command level of the previous recursive edit. If you wish, you can then abort the next recursive editing level.

Alternatively, the command `M-x top-level` aborts all levels of recursive edits, returning immediately to the top-level command reader.

The text being edited inside the recursive edit need not be the same text that you were editing at top level. It depends on what the recursive edit

is for. If the command that invokes the recursive edit selects a different buffer first, that is the buffer you will edit recursively. In any case, you can switch buffers within the recursive edit in the normal manner (as long as the buffer-switching keys have not been rebound). You could probably do all the rest of your editing inside the recursive edit, visiting files and all. But this could have surprising effects (such as stack overflow) from time to time. So remember to exit or abort the recursive edit when you no longer need it.

In general, we try to minimize the use of recursive editing levels in GNU Emacs. This is because they constrain you to "go back" in a particular order—from the innermost level toward the top level. When possible, we present different activities in separate buffers so that you can switch between them as you please. Some commands switch to a new major mode which provides a command to switch back. These approaches give you more flexibility to go back to unfinished tasks in the order you choose.

30.14 Emulation

GNU Emacs can be programmed to emulate (more or less) most other editors. Standard facilities can emulate these:

CRiSP/Brief (PC editor)
: You can turn on keybindings to emulate the CRiSP/Brief editor with `M-x crisp-mode`. Note that this rebinds `M-x` to exit Emacs unless you change the user option `crisp-override-meta-x`. You can also use the command `M-x scroll-all-mode` or set the user option `crisp-load-scroll-all` to emulate CRiSP's scroll-all feature (scrolling all windows together).

EDT (DEC VMS editor)
: Turn on EDT emulation with `M-x edt-emulation-on`. `M-x edt-emulation-off` restores normal Emacs command bindings.

  Most of the EDT emulation commands are keypad keys, and most standard Emacs key bindings are still available. The EDT emulation rebindings are done in the global keymap, so there is no problem switching buffers or major modes while in EDT emulation.

"PC" bindings
: The command `M-x pc-bindings-mode` sets up certain key bindings for "PC compatibility"—what people are often used to on PCs—as follows: `Delete` and its variants delete forward instead of backward, `C-Backspace` kills backward a word (as `C-Delete` normally would), `M-Backspace` does undo, `Home` and `End` move to beginning and end of line, `C-Home` and `C-End` move to beginning and end of buffer and `C-Escape` does `list-buffers`.

PC Selection mode
: The command `M-x pc-selection-mode` enables a global minor mode that emulates the mark, copy, cut and paste commands of various other systems—an interface known as CUA. It establishes the keybindings of PC mode, and also modifies the bindings of the cursor keys and the `next`, `prior`, `home` and `end` keys. It does not provide the full set of CUA keybindings—the fundamental Emacs keys `C-c`, `C-v` and `C-x` are not changed.

  The standard keys for moving around (`right`, `left`, `up`, `down`, `home`, `end`, `prior`, `next`, called "move-keys") deactivate the mark in PC selection mode. However, using `Shift` together with the "move keys" activates the region over which they move. The copy, cut and paste functions are available on `C-insert`, `S-delete` and `S-insert` respectively.

  The `s-region` package provides similar, but less complete, facilities.

TPU (DEC VMS editor)
: `M-x tpu-edt-on` turns on emulation of the TPU editor emulating EDT.

vi (Berkeley editor)
: Viper is the newest emulator for vi. It implements several levels of emulation; level 1 is closest to vi itself, while level 5 departs somewhat from strict emulation to take advantage of the capabilities of Emacs. To invoke Viper, type `M-x viper-mode`; it will guide you the rest of the way and ask for the emulation level. See Info file '`viper`', node '`Top`'.

vi (another emulator)
: `M-x vi-mode` enters a major mode that replaces the previously established major mode. All of the vi commands that, in real vi, enter "input" mode are programmed instead to return to the previous major mode. Thus, ordinary Emacs serves as vi's "input" mode.

  Because vi emulation works through major modes, it does not work to switch buffers during emulation. Return to normal Emacs first.

  If you plan to use vi emulation much, you probably want to bind a key to the `vi-mode` command.

vi (alternate emulator)
: `M-x vip-mode` invokes another vi emulator, said to resemble real vi more thoroughly than `M-x vi-mode`. "Input" mode in this emulator is changed from ordinary Emacs so you can use ⟨ESC⟩ to go back to emulated vi command mode. To get from emulated vi command mode back to ordinary Emacs, type `C-z`.

This emulation does not work through major modes, and it is possible to switch buffers in various ways within the emulator. It is not so necessary to assign a key to the command `vip-mode` as it is with `vi-mode` because terminating insert mode does not use it.

See Info file '`vip`', node '`Top`', for full information.

WordStar (old wordprocessor)
`M-x wordstar-mode` provides a major mode with WordStar-like keybindings.

30.15 Hyperlinking and Navigation Features

Various modes documented elsewhere have hypertext features so that you can follow links, usually by clicking `Mouse-2` on the link or typing ⟨RET⟩ while point is on the link. Info mode, Help mode and the Dired-like modes are examples. The Tags facility links between uses and definitions in source files, see Section 22.16 [Tags], page 301. Imenu provides navigation amongst items indexed in the current buffer, see Section 22.17 [Imenu], page 310. Info-lookup provides mode-specific lookup of definitions in Info indexes, see Section 22.13 [Documentation], page 297. Speedbar maintains a frame in which links to files, and locations in files are displayed, see Section 17.9 [Speedbar], page 211.

Other non-mode-specific facilities described in this section enable following links from the current buffer in a context-sensitive fashion.

30.15.1 Following URLs

`M-x browse-url` ⟨RET⟩ *url* ⟨ret⟩
Load a URL into a Web browser.

The Browse-URL package provides facilities for following URLs specifying links on the World Wide Web. Usually this works by invoking a web browser, but you can, for instance, arrange to invoke `compose-mail` from '`mailto:`' URLs.

The general way to use this feature is to type `M-x browse-url`, which displays a specified URL. If point is located near a plausible URL, that URL is used as the default. Other commands are available which you might like to bind to keys, such as `browse-url-at-point` and `browse-url-at-mouse`.

You can customize Browse-URL's behaviour via various options in the `browse-url` Customize group, particularly `browse-url-browser-function`. You can invoke actions dependent on the type of URL by

defining `browse-url-browser-function` as an association list. The package's commentary available via `C-h p` provides more information. Packages with facilities for following URLs should always go through Browse-URL, so that the customization options for Browse-URL will affect all browsing in Emacs.

30.15.2 Activating URLs

`M-x goto-address`
: Activate URLs and e-mail addresses in the current buffer.

You can make URLs in the current buffer active with `M-x goto-address`. This finds all the URLs in the buffer, and establishes bindings for `Mouse-2` and `C-c` ⟨RET⟩ on them. After activation, if you click on a URL with `Mouse-2`, or move to a URL and type `C-c` ⟨RET⟩, that will display the web page that the URL specifies. For a '`mailto`' URL, it sends mail instead, using your selected mail-composition method (see Section 26.6 [Mail Methods], page 366).

It can be useful to add `goto-address` to mode hooks and the hooks used to display an incoming message. `rmail-show-message-hook` is the appropriate hook for Rmail, and `mh-show-mode-hook` for MH-E. This is not needed for Gnus, which has a similar feature of its own.

30.15.3 Finding Files and URLs at Point

FFAP mode replaces certain key bindings for finding files, including `C-x C-f`, with commands that provide more sensitive defaults. These commands behave like the ordinary ones when given a prefix argument. Otherwise, they get the default file name or URL from the text around point. If what is found in the buffer has the form of a URL rather than a file name, the commands use `browse-url` to view it.

This feature is useful for following references in mail or news buffers, '`README`' files, '`MANIFEST`' files, and so on. The '`ffap`' package's commentary available via `C-h p` and the `ffap` Custom group provide details.

You can turn on FFAP minor mode to make the following key bindings and to install hooks for using `ffap` in Rmail, Gnus and VM article buffers.

`C-x C-f` *filename* ⟨RET⟩
: Find *filename*, guessing a default from text around point (`find-file-at-point`).

`C-x 4 f`
: `ffap-other-window`, analogous to `find-file-other-window`.

`C-x 5 f`
: `ffap-other-frame`, analogous to `find-file-other-frame`.

`M-x ffap-next`
: Search buffer for next file name or URL, then find that file or URL.

`C-x d` *directory* ⟨RET⟩
: Start Dired on *directory*, defaulting to the directory name at point (`ffap-dired-at-point`).

`S-Mouse-3`
: `ffap-at-mouse` finds the file guessed from text around the position of a mouse click.

`C-S-Mouse-3`
: Display a menu of files and URLs mentioned in current buffer, then find the one you select (`ffap-menu`).

30.15.4 Finding Function and Variable Definitions

`M-x find-function` ⟨RET⟩ *function* ⟨RET⟩
: Find the definition *function* in its source file.

`M-x find-variable` ⟨RET⟩ *variable* ⟨RET⟩
: Find the definition of *variable* in its source file.

`M-x find-function-on-key` ⟨RET⟩ *key*
: Find the definition of the function that *key* invokes.

These commands provide an easy way to find the definitions of Emacs Lisp functions and variables. They are similar in purpose to the Tags facility (see Section 22.16 [Tags], page 301), but don't require a tags table; on the other hand, they only works for function and variable definitions that are already loaded in the Emacs session.

To find the definition of a function, use `M-x find-function`. `M-x find-variable` finds the definition of a specified variable. `M-x find-function-on-key` finds the definition of the function bound to a specified key.

To use these commands, you must have the Lisp source ('`.el`') files available along with the compiled ('`.elc`') files, in directories in `load-path`. You can use compressed source files if you enable Auto Compression mode. These commands only handle definitions written in Lisp, not primitive functions or variables defined in the C code of Emacs.

30.16 Dissociated Press

`M-x dissociated-press` is a command for scrambling a file of text either word by word or character by character. Starting from a buffer of straight English, it produces extremely amusing output. The input comes from the

current Emacs buffer. Dissociated Press writes its output in a buffer named '`*Dissociation*`', and redisplays that buffer after every couple of lines (approximately) so you can read the output as it comes out.

Dissociated Press asks every so often whether to continue generating output. Answer `n` to stop it. You can also stop at any time by typing `C-g`. The dissociation output remains in the '`*Dissociation*`' buffer for you to copy elsewhere if you wish.

Dissociated Press operates by jumping at random from one point in the buffer to another. In order to produce plausible output rather than gibberish, it insists on a certain amount of overlap between the end of one run of consecutive words or characters and the start of the next. That is, if it has just printed out 'president' and then decides to jump to a different point in the file, it might spot the 'ent' in 'pentagon' and continue from there, producing 'presidentagon'.[2] Long sample texts produce the best results.

A positive argument to `M-x dissociated-press` tells it to operate character by character, and specifies the number of overlap characters. A negative argument tells it to operate word by word and specifies the number of overlap words. In this mode, whole words are treated as the elements to be permuted, rather than characters. No argument is equivalent to an argument of two. For your againformation, the output goes only into the buffer '`*Dissociation*`'. The buffer you start with is not changed.

Dissociated Press produces nearly the same results as a Markov chain based on a frequency table constructed from the sample text. It is, however, an independent, ignoriginal invention. Dissociated Press techniquitously copies several consecutive characters from the sample between random choices, whereas a Markov chain would choose randomly for each word or character. This makes for more plausible sounding results, and runs faster.

It is a mustatement that too much use of Dissociated Press can be a developediment to your real work. Sometimes to the point of outragedy. And keep dissociwords out of your documentation, if you want it to be well userenced and properbose. Have fun. Your buggestions are welcome.

30.17 Other Amusements

If you are a little bit bored, you can try `M-x hanoi`. If you are considerably bored, give it a numeric argument. If you are very very bored, try an argument of 9. Sit back and watch.

If you want a little more personal involvement, try `M-x gomoku`, which plays the game Go Moku with you.

`M-x blackbox`, `M-x mpuz` and `M-x 5x5` are kinds of puzzles. `blackbox` challenges you to determine the location of objects inside a box by tomog-

[2] This dissociword actually appeared during the Vietnam War, when it was very appropriate.

raphy. `mpuz` displays a multiplication puzzle with letters standing for digits in a code that you must guess—to guess a value, type a letter and then the digit you think it stands for. The aim of `5x5` is to fill in all the squares.

`M-x decipher` helps you to cryptanalyze a buffer which is encrypted in a simple monoalphabetic substitution cipher.

`M-x dunnet` runs an adventure-style exploration game, which is a bigger sort of puzzle.

`M-x lm` runs a relatively non-participatory game in which a robot attempts to maneuver towards a tree at the center of the window based on unique olfactory cues from each of the four directions.

`M-x life` runs Conway's "Life" cellular automaton.

`M-x morse-region` converts text in a region to Morse code and `M-x unmorse-region` converts it back. No cause for remorse.

`M-x pong` plays a Pong-like game, bouncing the ball off opposing bats.

`M-x solitaire` plays a game of solitaire in which you jump pegs across other pegs.

`M-x studlify-region` studlify-cases the region, producing text like this:

```
M-x stUdlIfY-RegioN stUdlIfY-CaSeS thE region.
```

`M-x tetris` runs an implementation of the well-known Tetris game. Likewise, `M-x snake` provides an implementation of Snake.

When you are frustrated, try the famous Eliza program. Just do `M-x doctor`. End each input by typing ⟨RET⟩ twice.

When you are feeling strange, type `M-x yow`.

The command `M-x zone` plays games with the display when Emacs is idle.

31 Customization

This chapter talks about various topics relevant to adapting the behavior of Emacs in minor ways. See *The Emacs Lisp Reference Manual* for how to make more far-reaching changes.

All kinds of customization affect only the particular Emacs session that you do them in. They are completely lost when you kill the Emacs session, and have no effect on other Emacs sessions you may run at the same time or later. The only way an Emacs session can affect anything outside of it is by writing a file; in particular, the only way to make a customization "permanent" is to put something in your '`.emacs`' file or other appropriate file to do the customization in each session. See Section 31.7 [Init File], page 486.

31.1 Minor Modes

Minor modes are optional features which you can turn on or off. For example, Auto Fill mode is a minor mode in which ⟨SPC⟩ breaks lines between words as you type. All the minor modes are independent of each other and of the selected major mode. Most minor modes say in the mode line when they are on; for example, '`Fill`' in the mode line means that Auto Fill mode is on.

Append `-mode` to the name of a minor mode to get the name of a command function that turns the mode on or off. Thus, the command to enable or disable Auto Fill mode is called `M-x auto-fill-mode`. These commands are usually invoked with `M-x`, but you can bind keys to them if you wish. With no argument, the function turns the mode on if it was off and off if it was on. This is known as *toggling*. A positive argument always turns the mode on, and an explicit zero argument or a negative argument always turns it off.

Enabling or disabling some minor modes applies only to the current buffer; each buffer is independent of the other buffers. Therefore, you can enable the mode in particular buffers and disable it in others. The per-buffer minor modes include Abbrev mode, Auto Fill mode, Auto Save mode, Font-Lock mode, ISO Accents mode, Outline minor mode, Overwrite mode, and Binary Overwrite mode.

Abbrev mode allows you to define abbreviations that automatically expand as you type them. For example, '`amd`' might expand to '`abbrev mode`'. See Chapter 24 [Abbrevs], page 345, for full information.

Auto Fill mode allows you to enter filled text without breaking lines explicitly. Emacs inserts newlines as necessary to prevent lines from becoming too long. See Section 21.5 [Filling], page 248.

Auto Save mode causes the contents of a buffer to be saved periodically to reduce the amount of work you can lose in case of a system crash. See Section 14.5 [Auto Save], page 158.

Enriched mode enables editing and saving of formatted text. See Section 21.11 [Formatted Text], page 265.

Flyspell mode automatically highlights misspelled words. See Section 13.4 [Spelling], page 139.

Font-Lock mode automatically highlights certain textual units found in programs, such as comments, strings, and function names being defined. This requires a window system that can display multiple fonts. See Section 11.1 [Faces], page 103.

ISO Accents mode makes the characters '`', '´', '¨', '^', '/' and '~' combine with the following letter, to produce an accented letter in the ISO Latin-1 character set. See ⟨undefined⟩ [Single-Byte Character Support], page ⟨undefined⟩.

Outline minor mode provides the same facilities as the major mode called Outline mode; but since it is a minor mode instead, you can combine it with any major mode. See Section 21.8 [Outline Mode], page 255.

Overwrite mode causes ordinary printing characters to replace existing text instead of shoving it to the right. For example, if point is in front of the 'B' in 'FOOBAR', then in Overwrite mode typing a G changes it to 'FOOGAR', instead of producing 'FOOGBAR' as usual. In Overwrite mode, the command C-q inserts the next character whatever it may be, even if it is a digit—this gives you a way to insert a character instead of replacing an existing character.

Binary Overwrite mode is a variant of Overwrite mode for editing binary files; it treats newlines and tabs like other characters, so that they overwrite other characters and can be overwritten by them.

The following minor modes normally apply to all buffers at once. Since each is enabled or disabled by the value of a variable, you *can* set them differently for particular buffers, by explicitly making the corresponding variables local in those buffers. See Section 31.2.4 [Locals], page 466.

Icomplete mode displays an indication of available completions when you are in the minibuffer and completion is active. See Section 5.3.4 [Completion Options], page 60.

Line Number mode enables continuous display in the mode line of the line number of point, and Column Number mode enables display of the column number. See Section 1.3 [Mode Line], page 26.

Scroll Bar mode gives each window a scroll bar (see Section 17.13 [Scroll Bars], page 214). Menu Bar mode gives each frame a menu bar (see Section 17.15 [Menu Bars], page 215). Both of these modes are enabled by default when you use the X Window System.

In Transient Mark mode, every change in the buffer contents "deactivates" the mark, so that commands that operate on the region will get an

error. This means you must either set the mark, or explicitly "reactivate" it, before each command that uses the region. The advantage of Transient Mark mode is that Emacs can display the region highlighted (currently only when using X). See Chapter 8 [Mark], page 77.

For most minor modes, the command name is also the name of a variable which directly controls the mode. The mode is enabled whenever this variable's value is non-`nil`, and the minor-mode command works by setting the variable. For example, the command `outline-minor-mode` works by setting the value of `outline-minor-mode` as a variable; it is this variable that directly turns Outline minor mode on and off. To check whether a given minor mode works this way, use `C-h v` to ask for documentation on the variable name.

These minor-mode variables provide a good way for Lisp programs to turn minor modes on and off; they are also useful in a file's local variables list. But please think twice before setting minor modes with a local variables list, because most minor modes are matter of user preference—other users editing the same file might not want the same minor modes you prefer.

31.2 Variables

A *variable* is a Lisp symbol which has a value. The symbol's name is also called the name of the variable. A variable name can contain any characters that can appear in a file, but conventionally variable names consist of words separated by hyphens. A variable can have a documentation string which describes what kind of value it should have and how the value will be used.

Lisp allows any variable to have any kind of value, but most variables that Emacs uses require a value of a certain type. Often the value should always be a string, or should always be a number. Sometimes we say that a certain feature is turned on if a variable is "non-`nil`," meaning that if the variable's value is `nil`, the feature is off, but the feature is on for *any* other value. The conventional value to use to turn on the feature—since you have to pick one particular value when you set the variable—is `t`.

Emacs uses many Lisp variables for internal record keeping, as any Lisp program must, but the most interesting variables for you are the ones that exist for the sake of customization. Emacs does not (usually) change the values of these variables; instead, you set the values, and thereby alter and control the behavior of certain Emacs commands. These variables are called *user options*. Most user options are documented in this manual, and appear in the Variable Index (see [Variable Index], page 589).

One example of a variable which is a user option is `fill-column`, which specifies the position of the right margin (as a number of characters from the left margin) to be used by the fill commands (see Section 21.5 [Filling], page 248).

31.2.1 Examining and Setting Variables

`C-h v` *var* ⟨RET⟩
: Display the value and documentation of variable *var* (`describe-variable`).

`M-x set-variable` ⟨RET⟩ *var* ⟨RET⟩ *value* ⟨RET⟩
: Change the value of variable *var* to *value*.

To examine the value of a single variable, use `C-h v` (`describe-variable`), which reads a variable name using the minibuffer, with completion. It displays both the value and the documentation of the variable. For example,

```
C-h v fill-column ⟨RET⟩
```

displays something like this:

```
fill-column's value is 75

Documentation:
*Column beyond which automatic line-wrapping should happen.
Automatically becomes buffer-local when set in any fashion.
```

The star at the beginning of the documentation indicates that this variable is a user option. `C-h v` is not restricted to user options; it allows any variable name.

The most convenient way to set a specific user option is with `M-x set-variable`. This reads the variable name with the minibuffer (with completion), and then reads a Lisp expression for the new value using the minibuffer a second time. For example,

```
M-x set-variable ⟨RET⟩ fill-column ⟨RET⟩ 75 ⟨RET⟩
```

sets `fill-column` to 75.

`M-x set-variable` is limited to user option variables, but you can set any variable with a Lisp expression, using the function `setq`. Here is a `setq` expression to set `fill-column`:

```
(setq fill-column 75)
```

To execute an expression like this one, go to the '`*scratch*`' buffer, type in the expression, and then type `C-j`. See Section 23.9 [Lisp Interaction], page 342.

Setting variables, like all means of customizing Emacs except where otherwise stated, affects only the current Emacs session.

31.2.2 Easy Customization Interface

A convenient way to find the user option variables that you want to change, and then change them, is with `M-x customize`. This command creates a *customization buffer* with which you can browse through the Emacs user options in a logically organized structure, then edit and set their values. You can also use the customization buffer to save settings permanently. (Not all Emacs user options are included in this structure as of yet, but we are adding the rest.)

The appearance of the example buffers in the following is typically different under a window system where faces can be used to indicate the active fields and other features.

31.2.2.1 Customization Groups

For customization purposes, user options are organized into *groups* to help you find them. Groups are collected into bigger groups, all the way up to a master group called `Emacs`.

`M-x customize` creates a customization buffer that shows the top-level `Emacs` group and the second-level groups immediately under it. It looks like this, in part:

```
/- Emacs group: ------------------------------------------------\
      [State]: visible group members are all at standard settings.
   Customization of the One True Editor.
   See also [Manual].

Editing group: [Go to Group]
Basic text editing facilities.

External group: [Go to Group]
Interfacing to external utilities.

more second-level groups

\- Emacs group end -----------------------------------------------/
```

This says that the buffer displays the contents of the `Emacs` group. The other groups are listed because they are its contents. But they are listed differently, without indentation and dashes, because *their* contents are not included. Each group has a single-line documentation string; the `Emacs` group also has a '`[State]`' line.

Most of the text in the customization buffer is read-only, but it typically includes some *editable fields* that you can edit. There are also *active fields*; this means a field that does something when you *invoke* it. To invoke an

active field, either click on it with `Mouse-1`, or move point to it and type ⟨RET⟩.

For example, the phrase '`[Go to Group]`' that appears in a second-level group is an active field. Invoking the '`[Go to Group]`' field for a group creates a new customization buffer, which shows that group and its contents. This field is a kind of hypertext link to another group.

The `Emacs` group does not include any user options itself, but other groups do. By examining various groups, you will eventually find the options and faces that belong to the feature you are interested in customizing. Then you can use the customization buffer to set them.

You can view the structure of customization groups on a larger scale with `M-x customize-browse`. This command creates a special kind of customization buffer which shows only the names of the groups (and options and faces), and their structure.

In this buffer, you can show the contents of a group by invoking '`[+]`'. When the group contents are visible, this button changes to '`[-]`'; invoking that hides the group contents.

Each group, option or face name in this buffer has an active field which says '`[Group]`', '`[Option]`' or '`[Face]`'. Invoking that active field creates an ordinary customization buffer showing just that group and its contents, just that option, or just that face. This is the way to set values in it.

31.2.2.2 Changing an Option

Here is an example of what a user option looks like in the customization buffer:

```
Kill Ring Max: [Hide] 30
   [State]: this option is unchanged from its standard setting.
Maximum length of kill ring before oldest elements are thrown away.
```

The text following '`[Hide]`', '`30`' in this case, indicates the current value of the option. If you see '`[Show]`' instead of '`[Hide]`', it means that the value is hidden; the customization buffer initially hides values that take up several lines. Invoke '`[Show]`' to show the value.

The line after the option name indicates the *customization state* of the option: in the example above, it says you have not changed the option yet. The word '`[State]`' at the beginning of this line is active; you can get a menu of various operations by invoking it with `Mouse-1` or ⟨RET⟩. These operations are essential for customizing the variable.

The line after the '`[State]`' line displays the beginning of the option's documentation string. If there are more lines of documentation, this line ends with '`[More]`'; invoke this to show the full documentation string.

To enter a new value for '`Kill Ring Max`', move point to the value and edit it textually. For example, you can type `M-d`, then insert another number.

When you begin to alter the text, you will see the '`[State]`' line change to say that you have edited the value:

```
[State]: you have edited the value as text, but not set the option.
```

Editing the value does not actually set the option variable. To do that, you must *set* the option. To do this, invoke the word '`[State]`' and choose '`Set for Current Session`'.

The state of the option changes visibly when you set it:

```
[State]: you have set this option, but not saved it for future sessions.
```

You don't have to worry about specifying a value that is not valid; setting the option checks for validity and will not really install an unacceptable value.

While editing a value or field that is a file name, directory name, command name, or anything else for which completion is defined, you can type `M-`⟨TAB⟩ (`widget-complete`) to do completion.

Some options have a small fixed set of possible legitimate values. These options don't let you edit the value textually. Instead, an active field '`[Value Menu]`' appears before the value; invoke this field to edit the value. For a boolean "on or off" value, the active field says '`[Toggle]`', and it changes to the other value. '`[Value Menu]`' and '`[Toggle]`' edit the buffer; the changes take effect when you use the '`Set for Current Session`' operation.

Some options have values with complex structure. For example, the value of `file-coding-system-alist` is an association list. Here is how it appears in the customization buffer:

```
File Coding System Alist: [Hide]
[INS] [DEL] File regexp: \.elc\'
            Choice: [Value Menu] Encoding/decoding pair:
            Decoding: emacs-mule
            Encoding: emacs-mule
[INS] [DEL] File regexp: \(\`\|/\)loaddefs.el\'
            Choice: [Value Menu] Encoding/decoding pair:
            Decoding: no-conversion
            Encoding: no-conversion
[INS] [DEL] File regexp: \.tar\'
            Choice: [Value Menu] Encoding/decoding pair:
            Decoding: no-conversion
            Encoding: no-conversion
[INS] [DEL] File regexp:
            Choice: [Value Menu] Encoding/decoding pair:
            Decoding: undecided
            Encoding: nil
[INS]
   [State]: this option is unchanged from its standard setting.
Alist to decide a coding system to use for a file I/O operation. [Hide]
The format is ((PATTERN . VAL) ...),
```

```
where PATTERN is a regular expression matching a file name,
```
[. . . more lines of documentation. . .]

Each association in the list appears on four lines, with several editable or "active" fields. You can edit the regexps and coding systems using ordinary editing commands. You can also invoke '`[Value Menu]`' to switch to a kind of value—for instance, to specify a function instead of a pair of coding systems.

To delete an association from the list, invoke the '`[DEL]`' button for that item. To add an association, invoke '`[INS]`' at the position where you want to add it. There is an '`[INS]`' button between each pair of association, another at the beginning and another at the end, so you can add the new association at any position in the list.

Two special commands, ⟨TAB⟩ and `S-`⟨TAB⟩, are useful for moving through the customization buffer. ⟨TAB⟩ (`widget-forward`) moves forward to the next active or editable field; `S-`⟨TAB⟩ (`widget-backward`) moves backward to the previous active or editable field.

Typing ⟨RET⟩ on an editable field also moves forward, just like ⟨TAB⟩. We set it up this way because people often type ⟨RET⟩ when they are finished editing a field. To insert a newline within an editable field, use `C-o` or `C-q C-j`.

Setting the option changes its value in the current Emacs session; *saving* the value changes it for future sessions as well. This works by writing code into your '`~/.emacs`' file so as to set the option variable again each time you start Emacs. To save the option, invoke '`[State]`' and select the '`Save for Future Sessions`' operation.

You can also restore the option to its standard value by invoking '`[State]`' and selecting the '`Erase Customization`' operation. There are actually three reset operations:

'`Reset`'
: If you have made some modifications and not yet set the option, this restores the text in the customization buffer to match the actual value.

'`Reset to Saved`'
: This restores the value of the option to the last saved value, and updates the text accordingly.

'`Erase Customization`'
: This sets the option to its standard value, and updates the text accordingly. This also eliminates any saved value for the option, so that you will get the standard value in future Emacs sessions.

Sometimes it is useful to record a comment about a specific customization. Use the '`Add Comment`' item from the '`[State]`' menu to create a field for entering the comment. The comment you enter will be saved, and displayed again if you again view the same option in a customization buffer, even in another session.

The state of a group indicates whether anything in that group has been edited, set or saved. You can select '`Set for Current Session`', '`Save for Future Sessions`' and the various kinds of '`Reset`' operation for the group; these operations on the group apply to all options in the group and its subgroups.

Near the top of the customization buffer there are two lines containing several active fields:

```
[Set for Current Session] [Save for Future Sessions]
[Reset] [Reset to Saved] [Erase Customization]   [Finish]
```

Invoking '`[Finish]`' either buries or kills this customization buffer according to the setting of the option `custom-buffer-done-function`; the default is to bury the buffer. Each of the other fields performs an operation—set, save or reset—on each of the items in the buffer that could meaningfully be set, saved or reset.

31.2.2.3 Customizing Faces

In addition to user options, some customization groups also include faces. When you show the contents of a group, both the user options and the faces in the group appear in the customization buffer. Here is an example of how a face looks:

```
Custom Changed Face: (sample) [Hide]
   [State]: this face is unchanged from its standard setting.
Parent groups: [Custom Magic Faces]
Attributes: [ ] Font family: [Value Menu] *
            [ ] Width: [Value Menu] *
            [ ] Height: [Value Menu] *
            [ ] Weight: [Value Menu] *
            [ ] Slant: [Value Menu] *
            [ ] Underline: [Value Menu] *
            [ ] Overline: [Value Menu] *
            [ ] Strike-through: [Value Menu] *
            [ ] Box around text: [Value Menu] Off
            [ ] Inverse-video: [Value Menu] *
            [X] Foreground: [Value Menu] Color: white       (sample)
            [X] Background: [Value Menu] Color: blue        (sample)
            [ ] Stipple: [Value Menu] *
```

Each face attribute has its own line. The '`[x]`' field before the attribute name indicates whether the attribute is *enabled*; '`X`' means that it is. You can enable or disable the attribute by invoking that field. When the attribute is enabled, you can change the attribute value in the usual ways.

On a black-and-white display, the colors you can use for the background are '`black`', '`white`', '`gray`', '`gray1`', and '`gray3`'. Emacs supports these shades of gray by using background stipple patterns instead of a color.

Setting, saving and resetting a face work like the same operations for options (see Section 31.2.2.2 [Changing an Option], page 460).

A face can specify different appearances for different types of display. For example, a face can make text red on a color display, but use a bold font on a monochrome display. To specify multiple appearances for a face, select '`Show Display Types`' in the menu you get from invoking '`[State]`'.

Another more basic way to set the attributes of a specific face is with `M-x modify-face`. This command reads the name of a face, then reads the attributes one by one. For the color and stipple attributes, the attribute's current value is the default—type just ⟨RET⟩ if you don't want to change that attribute. Type '`none`' if you want to clear out the attribute.

31.2.2.4 Customizing Specific Items

Instead of finding the options you want to change by moving down through the structure of groups, you can specify the particular option, face or group that you want to customize.

`M-x customize-option` ⟨RET⟩ *option* ⟨RET⟩
: Set up a customization buffer with just one option, *option*.

`M-x customize-face` ⟨RET⟩ *face* ⟨RET⟩
: Set up a customization buffer with just one face, *face*.

`M-x customize-group` ⟨RET⟩ *group* ⟨RET⟩
: Set up a customization buffer with just one group, *group*.

`M-x customize-apropos` ⟨RET⟩ *regexp* ⟨RET⟩
: Set up a customization buffer with all the options, faces and groups that match *regexp*.

`M-x customize-changed-options` ⟨RET⟩ *version* ⟨RET⟩
: Set up a customization buffer with all the options, faces and groups whose meaning has changed since Emacs version *version*.

`M-x customize-saved`
: Set up a customization buffer containing all options and faces that you have saved with customization buffers.

`M-x customize-customized`
: Set up a customization buffer containing all options and faces that you have customized but not saved.

If you want to alter a particular user option variable with the customization buffer, and you know its name, you can use the command `M-x`

`customize-option` and specify the option name. This sets up the customization buffer with just one option—the one that you asked for. Editing, setting and saving the value work as described above, but only for the specified option.

Likewise, you can modify a specific face, chosen by name, using `M-x customize-face`.

You can also set up the customization buffer with a specific group, using `M-x customize-group`. The immediate contents of the chosen group, including option variables, faces, and other groups, all appear as well. However, these subgroups' own contents start out hidden. You can show their contents in the usual way, by invoking '`[Show]`'.

To control more precisely what to customize, you can use `M-x customize-apropos`. You specify a regular expression as argument; then all options, faces and groups whose names match this regular expression are set up in the customization buffer. If you specify an empty regular expression, this includes *all* groups, options and faces in the customization buffer (but that takes a long time).

When you upgrade to a new Emacs version, you might want to customize new options and options whose meanings or default values have changed. To do this, use `M-x customize-changed-options` and specify a previous Emacs version number using the minibuffer. It creates a customization buffer which shows all the options (and groups) whose definitions have been changed since the specified version.

If you change option values and then decide the change was a mistake, you can use two special commands to revisit your previous changes. Use `customize-saved` to look at the options and faces that you have saved. Use `M-x customize-customized` to look at the options and faces that you have set but not saved.

31.2.3 Hooks

Hooks are an important mechanism for customization of Emacs. A hook is a Lisp variable which holds a list of functions, to be called on some well-defined occasion. (This is called *running the hook*.) The individual functions in the list are called the *hook functions* of the hook. With rare exceptions, hooks in Emacs are empty when Emacs starts up, so the only hook functions in any given hook are the ones you explicitly put there as customization.

Most major modes run one or more *mode hooks* as the last step of initialization. This makes it easy for you to customize the behavior of the mode, by setting up a hook function to override the local variable assignments already made by the mode. But hooks are also used in other contexts. For example, the hook `suspend-hook` runs just before Emacs suspends itself (see Section 3.1 [Exiting], page 38).

Most Emacs hooks are *normal hooks*. This means that running the hook operates by calling all the hook functions, unconditionally, with no arguments. We have made an effort to keep most hooks normal so that you can use them in a uniform way. Every variable in Emacs whose name ends in '`-hook`' is a normal hook.

There are also a few *abnormal hooks*. These variables' names end in '`-hooks`' or '`-functions`', instead of '`-hook`'. What makes these hooks abnormal is that there is something peculiar about the way its functions are called—perhaps they are given arguments, or perhaps the values they return are used in some way. For example, `find-file-not-found-hooks` (see Section 14.2 [Visiting], page 145) is abnormal because as soon as one hook function returns a non-`nil` value, the rest are not called at all. The documentation of each abnormal hook variable explains in detail what is peculiar about it.

The recommended way to add a hook function to a hook (either normal or abnormal) is by calling `add-hook`. You can use any valid Lisp function as the hook function, provided it can handle the proper number of arguments (zero arguments, in the case of a normal hook). Of course, not every Lisp function is *useful* in any particular hook.

For example, here's how to set up a hook to turn on Auto Fill mode when entering Text mode and other modes based on Text mode:

```
(add-hook 'text-mode-hook 'turn-on-auto-fill)
```

The next example shows how to use a hook to customize the indentation of C code. (People often have strong personal preferences for one format compared to another.) Here the hook function is an anonymous lambda expression.

```
(setq my-c-style
  '((c-comment-only-line-offset . 4)
    (c-cleanup-list . (scope-operator
       empty-defun-braces
       defun-close-semi))
    (c-offsets-alist . ((arglist-close . c-lineup-arglist)
(substatement-open . 0)))))

(add-hook 'c-mode-common-hook
  '(lambda ()
     (c-add-style "my-style" my-c-style t)))
```

It is best to design your hook functions so that the order in which they are executed does not matter. Any dependence on the order is "asking for trouble." However, the order is predictable: the most recently added hook functions are executed first.

31.2.4 Local Variables

`M-x make-local-variable` ⟨RET⟩ *var* ⟨RET⟩
: Make variable *var* have a local value in the current buffer.

`M-x kill-local-variable` ⟨RET⟩ *var* ⟨RET⟩
: Make variable *var* use its global value in the current buffer.

`M-x make-variable-buffer-local` ⟨RET⟩ *var* ⟨RET⟩
: Mark variable *var* so that setting it will make it local to the buffer that is current at that time.

Almost any variable can be made *local* to a specific Emacs buffer. This means that its value in that buffer is independent of its value in other buffers. A few variables are always local in every buffer. Every other Emacs variable has a *global* value which is in effect in all buffers that have not made the variable local.

`M-x make-local-variable` reads the name of a variable and makes it local to the current buffer. Further changes in this buffer will not affect others, and further changes in the global value will not affect this buffer.

`M-x make-variable-buffer-local` reads the name of a variable and changes the future behavior of the variable so that it will become local automatically when it is set. More precisely, once a variable has been marked in this way, the usual ways of setting the variable automatically do `make-local-variable` first. We call such variables *per-buffer* variables.

Major modes (see Chapter 19 [Major Modes], page 235) always make variables local to the buffer before setting the variables. This is why changing major modes in one buffer has no effect on other buffers. Minor modes also work by setting variables—normally, each minor mode has one controlling variable which is non-`nil` when the mode is enabled (see Section 31.1 [Minor Modes], page 455). For most minor modes, the controlling variable is per buffer.

Emacs contains a number of variables that are always per-buffer. These include `abbrev-mode`, `auto-fill-function`, `case-fold-search`, `comment-column`, `ctl-arrow`, `fill-column`, `fill-prefix`, `indent-tabs-mode`, `left-margin`, `mode-line-format`, `overwrite-mode`, `selective-display-ellipses`, `selective-display`, `tab-width`, and `truncate-lines`. Some other variables are always local in every buffer, but they are used for internal purposes.

A few variables cannot be local to a buffer because they are always local to each display instead (see Section 17.10 [Multiple Displays], page 211). If you try to make one of these variables buffer-local, you'll get an error message.

`M-x kill-local-variable` reads the name of a variable and makes it cease to be local to the current buffer. The global value of the variable henceforth is in effect in this buffer. Setting the major mode kills all the local variables of the buffer except for a few variables specially marked as *permanent locals*.

To set the global value of a variable, regardless of whether the variable has a local value in the current buffer, you can use the Lisp construct `setq-default`. This construct is used just like `setq`, but it sets variables' global values instead of their local values (if any). When the current buffer does have a local value, the new global value may not be visible until you switch to another buffer. Here is an example:

```
(setq-default fill-column 75)
```

`setq-default` is the only way to set the global value of a variable that has been marked with `make-variable-buffer-local`.

Lisp programs can use `default-value` to look at a variable's default value. This function takes a symbol as argument and returns its default value. The argument is evaluated; usually you must quote it explicitly. For example, here's how to obtain the default value of `fill-column`:

```
(default-value 'fill-column)
```

31.2.5 Local Variables in Files

A file can specify local variable values for use when you edit the file with Emacs. Visiting the file checks for local variable specifications; it automatically makes these variables local to the buffer, and sets them to the values specified in the file.

There are two ways to specify local variable values: in the first line, or with a local variables list. Here's how to specify them in the first line:

`-*- mode:` *modename*`; `*var*`:` *value*`; ... -*-`

You can specify any number of variables/value pairs in this way, each pair with a colon and semicolon as shown above. `mode:` *modename*`;` specifies the major mode; this should come first in the line. The *value*s are not evaluated; they are used literally. Here is an example that specifies Lisp mode and sets two variables with numeric values:

```
;; -*- mode: Lisp; fill-column: 75; comment-column: 50; -*-
```

You can also specify the coding system for a file in this way: just specify a value for the "variable" named `coding`. The "value" must be a coding system name that Emacs recognizes. See Section 18.6 [Coding Systems], page 223.

The `eval` pseudo-variable, described below, can be specified in the first line as well.

In shell scripts, the first line is used to identify the script interpreter, so you cannot put any local variables there. To accomodate for this, when Emacs visits a shell script, it looks for local variable specifications in the *second* line.

A *local variables list* goes near the end of the file, in the last page. (It is often best to put it on a page by itself.) The local variables list starts with a line containing the string '`Local Variables:`', and ends with a line

containing the string '`End:`'. In between come the variable names and values, one set per line, as '*variable*`:` *value*'. The *value*s are not evaluated; they are used literally. If a file has both a local variables list and a '`-*-`' line, Emacs processes *everything* in the '`-*-`' line first, and *everything* in the local variables list afterward.

Here is an example of a local variables list:

```
;;; Local Variables: ***
;;; mode:lisp ***
;;; comment-column:0 ***
;;; comment-start: ";;; "  ***
;;; comment-end:"***" ***
;;; End: ***
```

As you see, each line starts with the prefix '`;;; `' and each line ends with the suffix '` ***`'. Emacs recognizes these as the prefix and suffix based on the first line of the list, by finding them surrounding the magic string '`Local Variables:`'; then it automatically discards them from the other lines of the list.

The usual reason for using a prefix and/or suffix is to embed the local variables list in a comment, so it won't confuse other programs that the file is intended as input for. The example above is for a language where comment lines start with '`;;; `' and end with '`***`'; the local values for `comment-start` and `comment-end` customize the rest of Emacs for this unusual syntax. Don't use a prefix (or a suffix) if you don't need one.

Two "variable names" have special meanings in a local variables list: a value for the variable `mode` really sets the major mode, and a value for the variable `eval` is simply evaluated as an expression and the value is ignored. `mode` and `eval` are not real variables; setting variables named `mode` and `eval` in any other context has no special meaning. If `mode` is used to set a major mode, it should be the first "variable" in the list.

You can use the `mode` "variable" to set minor modes as well as major modes; in fact, you can use it more than once, first to set the major mode and then to set minor modes which are specific to particular buffers. But most minor modes should not be specified in the file in any fashion, because they represent user preferences.

For example, you may be tempted to try to turn on Auto Fill mode with a local variable list. That is a mistake. The choice of Auto Fill mode or not is a matter of individual taste, not a matter of the contents of particular files. If you want to use Auto Fill, set up major mode hooks with your '`.emacs`' file to turn it on (when appropriate) for you alone (see Section 31.7 [Init File], page 486). Don't use a local variable list to impose your taste on everyone.

The start of the local variables list must be no more than 3000 characters from the end of the file, and must be in the last page if the file is divided into pages. Otherwise, Emacs will not notice it is there. The purpose of

this rule is so that a stray '`Local Variables:`' not in the last page does not confuse Emacs, and so that visiting a long file that is all one page and has no local variables list need not take the time to search the whole file.

Use the command `normal-mode` to reset the local variables and major mode of a buffer according to the file name and contents, including the local variables list if any. See Section 19.1 [Choosing Modes], page 236.

The variable `enable-local-variables` controls whether to process local variables in files, and thus gives you a chance to override them. Its default value is `t`, which means do process local variables in files. If you set the value to `nil`, Emacs simply ignores local variables in files. Any other value says to query you about each file that has local variables, showing you the local variable specifications so you can judge.

The `eval` "variable," and certain actual variables, create a special risk; when you visit someone else's file, local variable specifications for these could affect your Emacs in arbitrary ways. Therefore, the option `enable-local-eval` controls whether Emacs processes `eval` variables, as well variables with names that end in '`-hook`', '`-hooks`', '`-function`' or '`-functions`', and certain other variables. The three possibilities for the option's value are `t`, `nil`, and anything else, just as for `enable-local-variables`. The default is `maybe`, which is neither `t` nor `nil`, so normally Emacs does ask for confirmation about file settings for these variables.

31.3 Keyboard Macros

A *keyboard macro* is a command defined by the user to stand for another sequence of keys. For example, if you discover that you are about to type `C-n C-d` forty times, you can speed your work by defining a keyboard macro to do `C-n C-d` and calling it with a repeat count of forty.

`C-x (` Start defining a keyboard macro (`start-kbd-macro`).

`C-x )` End the definition of a keyboard macro (`end-kbd-macro`).

`C-x e` Execute the most recent keyboard macro (`call-last-kbd-macro`).

`C-u C-x (` Re-execute last keyboard macro, then add more keys to its definition.

`C-x q` When this point is reached during macro execution, ask for confirmation (`kbd-macro-query`).

`M-x name-last-kbd-macro`
Give a command name (for the duration of the session) to the most recently defined keyboard macro.

`M-x insert-kbd-macro`
Insert in the buffer a keyboard macro's definition, as Lisp code.

`C-x C-k` Edit a previously defined keyboard macro (`edit-kbd-macro`).

`M-x apply-macro-to-region-lines`
Run the last keyboard macro on each complete line in the region.

Keyboard macros differ from ordinary Emacs commands in that they are written in the Emacs command language rather than in Lisp. This makes it easier for the novice to write them, and makes them more convenient as temporary hacks. However, the Emacs command language is not powerful enough as a programming language to be useful for writing anything intelligent or general. For such things, Lisp must be used.

You define a keyboard macro while executing the commands which are the definition. Put differently, as you define a keyboard macro, the definition is being executed for the first time. This way, you can see what the effects of your commands are, so that you don't have to figure them out in your head. When you are finished, the keyboard macro is defined and also has been, in effect, executed once. You can then do the whole thing over again by invoking the macro.

31.3.1 Basic Use

To start defining a keyboard macro, type the `C-x (` command (`start-kbd-macro`). From then on, your keys continue to be executed, but also become part of the definition of the macro. '`Def`' appears in the mode line to remind you of what is going on. When you are finished, the `C-x )` command (`end-kbd-macro`) terminates the definition (without becoming part of it!). For example,

```
C-x ( M-f foo C-x )
```

defines a macro to move forward a word and then insert '`foo`'.

The macro thus defined can be invoked again with the `C-x e` command (`call-last-kbd-macro`), which may be given a repeat count as a numeric argument to execute the macro many times. `C-x )` can also be given a repeat count as an argument, in which case it repeats the macro that many times right after defining it, but defining the macro counts as the first repetition (since it is executed as you define it). Therefore, giving `C-x )` an argument of 4 executes the macro immediately 3 additional times. An argument of zero to `C-x e` or `C-x )` means repeat the macro indefinitely (until it gets an error or you type `C-g` or, on MS-DOS, `C-`⟨BREAK⟩).

If you wish to repeat an operation at regularly spaced places in the text, define a macro and include as part of the macro the commands to move to the next place you want to use it. For example, if you want to change each line, you should position point at the start of a line, and define a macro to change that line and leave point at the start of the next line. Then repeating the macro will operate on successive lines.

After you have terminated the definition of a keyboard macro, you can add to the end of its definition by typing `C-u C-x (`. This is equivalent to plain `C-x (` followed by retyping the whole definition so far. As a consequence it re-executes the macro as previously defined.

You can use function keys in a keyboard macro, just like keyboard keys. You can even use mouse events, but be careful about that: when the macro replays the mouse event, it uses the original mouse position of that event, the position that the mouse had while you were defining the macro. The effect of this may be hard to predict. (Using the current mouse position would be even less predictable.)

One thing that doesn't always work well in a keyboard macro is the command `C-M-c` (`exit-recursive-edit`). When this command exits a recursive edit that started within the macro, it works as you'd expect. But if it exits a recursive edit that started before you invoked the keyboard macro, it also necessarily exits the keyboard macro as part of the process.

You can edit a keyboard macro already defined by typing `C-x C-k` (`edit-kbd-macro`). Follow that with the keyboard input that you would use to invoke the macro—`C-x e` or `M-x` *name* or some other key sequence. This formats the macro definition in a buffer and enters a specialized major mode for editing it. Type `C-h m` once in that buffer to display details of how to edit the macro. When you are finished editing, type `C-c C-c`.

The command `M-x apply-macro-to-region-lines` repeats the last defined keyboard macro on each complete line within the current region. It does this line by line, by moving point to the beginning of the line and then executing the macro.

31.3.2 Naming and Saving Keyboard Macros

If you wish to save a keyboard macro for longer than until you define the next one, you must give it a name using `M-x name-last-kbd-macro`. This reads a name as an argument using the minibuffer and defines that name to execute the macro. The macro name is a Lisp symbol, and defining it in this way makes it a valid command name for calling with `M-x` or for binding a key to with `global-set-key` (see Section 31.4.1 [Keymaps], page 474). If you specify a name that has a prior definition other than another keyboard macro, an error message is printed and nothing is changed.

Once a macro has a command name, you can save its definition in a file. Then it can be used in another editing session. First, visit the file you want to save the definition in. Then use this command:

`M-x insert-kbd-macro` ⟨RET⟩ *macroname* ⟨RET⟩

This inserts some Lisp code that, when executed later, will define the same macro with the same definition it has now. (You need not understand Lisp code to do this, because `insert-kbd-macro` writes the Lisp code for you.)

Then save the file. You can load the file later with `load-file` (see Section 23.7 [Lisp Libraries], page 339). If the file you save in is your init file '`~/.emacs`' (see Section 31.7 [Init File], page 486) then the macro will be defined each time you run Emacs.

If you give `insert-kbd-macro` a numeric argument, it makes additional Lisp code to record the keys (if any) that you have bound to the keyboard macro, so that the macro will be reassigned the same keys when you load the file.

31.3.3 Executing Macros with Variations

Using `C-x q` (`kbd-macro-query`), you can get an effect similar to that of `query-replace`, where the macro asks you each time around whether to make a change. While defining the macro, type `C-x q` at the point where you want the query to occur. During macro definition, the `C-x q` does nothing, but when you run the macro later, `C-x q` asks you interactively whether to continue.

The valid responses when `C-x q` asks are ⟨SPC⟩ (or `y`), ⟨DEL⟩ (or `n`), ⟨RET⟩ (or `q`), `C-l` and `C-r`. The answers are the same as in `query-replace`, though not all of the `query-replace` options are meaningful.

These responses include ⟨SPC⟩ to continue, and ⟨DEL⟩ to skip the remainder of this repetition of the macro and start right away with the next repetition. ⟨RET⟩ means to skip the remainder of this repetition and cancel further repetitions. `C-l` redraws the screen and asks you again for a character to say what to do.

`C-r` enters a recursive editing level, in which you can perform editing which is not part of the macro. When you exit the recursive edit using `C-M-c`, you are asked again how to continue with the keyboard macro. If you type a ⟨SPC⟩ at this time, the rest of the macro definition is executed. It is up to you to leave point and the text in a state such that the rest of the macro will do what you want.

`C-u C-x q`, which is `C-x q` with a numeric argument, performs a completely different function. It enters a recursive edit reading input from the keyboard, both when you type it during the definition of the macro, and when it is executed from the macro. During definition, the editing you do inside the recursive edit does not become part of the macro. During macro execution, the recursive edit gives you a chance to do some particularized editing on each repetition. See Section 30.13 [Recursive Edit], page 447.

Another way to vary the behavior of a keyboard macro is to use a register as a counter, incrementing it on each repetition of the macro. See Section 10.5 [RegNumbers], page 99.

31.4 Customizing Key Bindings

This section describes *key bindings*, which map keys to commands, and *keymaps*, which record key bindings. It also explains how to customize key bindings.

Recall that a command is a Lisp function whose definition provides for interactive use. Like every Lisp function, a command has a function name which usually consists of lower-case letters and hyphens.

31.4.1 Keymaps

The bindings between key sequences and command functions are recorded in data structures called *keymaps*. Emacs has many of these, each used on particular occasions.

Recall that a *key sequence* (*key*, for short) is a sequence of *input events* that have a meaning as a unit. Input events include characters, function keys and mouse buttons—all the inputs that you can send to the computer with your terminal. A key sequence gets its meaning from its *binding*, which says what command it runs. The function of keymaps is to record these bindings.

The *global* keymap is the most important keymap because it is always in effect. The global keymap defines keys for Fundamental mode; most of these definitions are common to most or all major modes. Each major or minor mode can have its own keymap which overrides the global definitions of some keys.

For example, a self-inserting character such as `g` is self-inserting because the global keymap binds it to the command `self-insert-command`. The standard Emacs editing characters such as `C-a` also get their standard meanings from the global keymap. Commands to rebind keys, such as `M-x global-set-key`, actually work by storing the new binding in the proper place in the global map. See Section 31.4.5 [Rebinding], page 477.

Meta characters work differently; Emacs translates each Meta character into a pair of characters starting with ⟨ESC⟩. When you type the character `M-a` in a key sequence, Emacs replaces it with ⟨ESC⟩ `a`. A meta key comes in as a single input event, but becomes two events for purposes of key bindings. The reason for this is historical, and we might change it someday.

Most modern keyboards have function keys as well as character keys. Function keys send input events just as character keys do, and keymaps can have bindings for them.

On many terminals, typing a function key actually sends the computer a sequence of characters; the precise details of the sequence depends on which function key and on the model of terminal you are using. (Often the sequence

starts with ⟨ESC⟩ [.) If Emacs understands your terminal type properly, it recognizes the character sequences forming function keys wherever they occur in a key sequence (not just at the beginning). Thus, for most purposes, you can pretend the function keys reach Emacs directly and ignore their encoding as character sequences.

Mouse buttons also produce input events. These events come with other data—the window and position where you pressed or released the button, and a time stamp. But only the choice of button matters for key bindings; the other data matters only if a command looks at it. (Commands designed for mouse invocation usually do look at the other data.)

A keymap records definitions for single events. Interpreting a key sequence of multiple events involves a chain of keymaps. The first keymap gives a definition for the first event; this definition is another keymap, which is used to look up the second event in the sequence, and so on.

Key sequences can mix function keys and characters. For example, `C-x` ⟨SELECT⟩ is meaningful. If you make ⟨SELECT⟩ a prefix key, then ⟨SELECT⟩ `C-n` makes sense. You can even mix mouse events with keyboard events, but we recommend against it, because such sequences are inconvenient to type in.

As a user, you can redefine any key; but it might be best to stick to key sequences that consist of `C-c` followed by a letter. These keys are "reserved for users," so they won't conflict with any properly designed Emacs extension. The function keys ⟨F5⟩ through ⟨F9⟩ are also reserved for users. If you redefine some other key, your definition may be overridden by certain extensions or major modes which redefine the same key.

31.4.2 Prefix Keymaps

A prefix key such as `C-x` or ⟨ESC⟩ has its own keymap, which holds the definition for the event that immediately follows that prefix.

The definition of a prefix key is usually the keymap to use for looking up the following event. The definition can also be a Lisp symbol whose function definition is the following keymap; the effect is the same, but it provides a command name for the prefix key that can be used as a description of what the prefix key is for. Thus, the binding of `C-x` is the symbol `Ctl-X-Prefix`, whose function definition is the keymap for `C-x` commands. The definitions of `C-c`, `C-x`, `C-h` and ⟨ESC⟩ as prefix keys appear in the global map, so these prefix keys are always available.

Aside from ordinary prefix keys, there is a fictitious "prefix key" which represents the menu bar; see section "Menu Bar" in *The Emacs Lisp Reference Manual*, for special information about menu bar key bindings. Mouse button events that invoke pop-up menus are also prefix keys; see section "Menu Keymaps" in *The Emacs Lisp Reference Manual*, for more details.

Some prefix keymaps are stored in variables with names:

- `ctl-x-map` is the variable name for the map used for characters that follow `C-x`.
- `help-map` is for characters that follow `C-h`.
- `esc-map` is for characters that follow ⟨ESC⟩. Thus, all Meta characters are actually defined by this map.
- `ctl-x-4-map` is for characters that follow `C-x 4`.
- `mode-specific-map` is for characters that follow `C-c`.

31.4.3 Local Keymaps

So far we have explained the ins and outs of the global map. Major modes customize Emacs by providing their own key bindings in *local keymaps*. For example, C mode overrides ⟨TAB⟩ to make it indent the current line for C code. Portions of text in the buffer can specify their own keymaps to substitute for the keymap of the buffer's major mode.

Minor modes can also have local keymaps. Whenever a minor mode is in effect, the definitions in its keymap override both the major mode's local keymap and the global keymap.

The local keymaps for Lisp mode and several other major modes always exist even when not in use. These are kept in variables named `lisp-mode-map` and so on. For major modes less often used, the local keymap is normally constructed only when the mode is used for the first time in a session. This is to save space. If you wish to change one of these keymaps, you must use the major mode's *mode hook*—see below.

All minor mode keymaps are created in advance. There is no way to defer their creation until the first time the minor mode is enabled.

A local keymap can locally redefine a key as a prefix key by defining it as a prefix keymap. If the key is also defined globally as a prefix, then its local and global definitions (both keymaps) effectively combine: both of them are used to look up the event that follows the prefix key. Thus, if the mode's local keymap defines `C-c` as another keymap, and that keymap defines `C-z` as a command, this provides a local meaning for `C-c C-z`. This does not affect other sequences that start with `C-c`; if those sequences don't have their own local bindings, their global bindings remain in effect.

Another way to think of this is that Emacs handles a multi-event key sequence by looking in several keymaps, one by one, for a binding of the whole key sequence. First it checks the minor mode keymaps for minor modes that are enabled, then it checks the major mode's keymap, and then it checks the global keymap. This is not precisely how key lookup works, but it's good enough for understanding ordinary circumstances.

To change the local bindings of a major mode, you must change the mode's local keymap. Normally you must wait until the first time the mode is used, because most major modes don't create their keymaps until then.

If you want to specify something in your '`~/.emacs`' file to change a major mode's bindings, you must use the mode's mode hook to delay the change until the mode is first used.

For example, the command `texinfo-mode` to select Texinfo mode runs the hook `texinfo-mode-hook`. Here's how you can use the hook to add local bindings (not very useful, we admit) for `C-c n` and `C-c p` in Texinfo mode:

```
(add-hook 'texinfo-mode-hook
          '(lambda ()
             (define-key texinfo-mode-map "\C-cp"
                         'backward-paragraph)
             (define-key texinfo-mode-map "\C-cn"
                         'forward-paragraph)))
```

See Section 31.2.3 [Hooks], page 465.

31.4.4 Minibuffer Keymaps

The minibuffer has its own set of local keymaps; they contain various completion and exit commands.

- `minibuffer-local-map` is used for ordinary input (no completion).
- `minibuffer-local-ns-map` is similar, except that ⟨SPC⟩ exits just like ⟨RET⟩. This is used mainly for Mocklisp compatibility.
- `minibuffer-local-completion-map` is for permissive completion.
- `minibuffer-local-must-match-map` is for strict completion and for cautious completion.

31.4.5 Changing Key Bindings Interactively

The way to redefine an Emacs key is to change its entry in a keymap. You can change the global keymap, in which case the change is effective in all major modes (except those that have their own overriding local definitions for the same key). Or you can change the current buffer's local map, which affects all buffers using the same major mode.

`M-x global-set-key` ⟨RET⟩ *key cmd* ⟨RET⟩
: Define *key* globally to run *cmd*.

`M-x local-set-key` ⟨RET⟩ *key cmd* ⟨RET⟩
: Define *key* locally (in the major mode now in effect) to run *cmd*.

`M-x global-unset-key` ⟨RET⟩ *key*
: Make *key* undefined in the global map.

`M-x local-unset-key` ⟨RET⟩ *key*
: Make *key* undefined locally (in the major mode now in effect).

For example, suppose you like to execute commands in a subshell within an Emacs buffer, instead of suspending Emacs and executing commands in your login shell. Normally, `C-z` is bound to the function `suspend-emacs` (when not using the X Window System), but you can change `C-z` to invoke an interactive subshell within Emacs, by binding it to `shell` as follows:

```
M-x global-set-key ⟨RET⟩ C-z shell ⟨RET⟩
```

`global-set-key` reads the command name after the key. After you press the key, a message like this appears so that you can confirm that you are binding the key you want:

```
Set key C-z to command:
```

You can redefine function keys and mouse events in the same way; just type the function key or click the mouse when it's time to specify the key to rebind.

You can rebind a key that contains more than one event in the same way. Emacs keeps reading the key to rebind until it is a complete key (that is, not a prefix key). Thus, if you type `C-f` for *key*, that's the end; the minibuffer is entered immediately to read *cmd*. But if you type `C-x`, another character is read; if that is `4`, another character is read, and so on. For example,

```
M-x global-set-key ⟨RET⟩ C-x 4 $ spell-other-window ⟨RET⟩
```

redefines `C-x 4 $` to run the (fictitious) command `spell-other-window`.

The two-character keys consisting of `C-c` followed by a letter are reserved for user customizations. Lisp programs are not supposed to define these keys, so the bindings you make for them will be available in all major modes and will never get in the way of anything.

You can remove the global definition of a key with `global-unset-key`. This makes the key *undefined*; if you type it, Emacs will just beep. Similarly, `local-unset-key` makes a key undefined in the current major mode keymap, which makes the global definition (or lack of one) come back into effect in that major mode.

If you have redefined (or undefined) a key and you subsequently wish to retract the change, undefining the key will not do the job—you need to redefine the key with its standard definition. To find the name of the standard definition of a key, go to a Fundamental mode buffer and use `C-h c`. The documentation of keys in this manual also lists their command names.

If you want to prevent yourself from invoking a command by mistake, it is better to disable the command than to undefine the key. A disabled command is less work to invoke when you really want to. See Section 31.4.11 [Disabling], page 484.

31.4.6 Rebinding Keys in Your Init File

If you have a set of key bindings that you like to use all the time, you can specify them in your '`.emacs`' file by using their Lisp syntax. (See Section 31.7 [Init File], page 486.)

The simplest method for doing this works for ASCII characters and Meta-modified ASCII characters only. This method uses a string to represent the key sequence you want to rebind. For example, here's how to bind `C-z` to `shell`:

```
(global-set-key "\C-z" 'shell)
```

This example uses a string constant containing one character, `C-z`. The single-quote before the command name, `shell`, marks it as a constant symbol rather than a variable. If you omit the quote, Emacs would try to evaluate `shell` immediately as a variable. This probably causes an error; it certainly isn't what you want.

Here is another example that binds a key sequence two characters long:

```
(global-set-key "\C-xl" 'make-symbolic-link)
```

When the key sequence includes function keys or mouse button events, or non-ASCII characters such as `C-=` or `H-a`, you must use the more general method of rebinding, which uses a vector to specify the key sequence.

The way to write a vector in Emacs Lisp is with square brackets around the vector elements. Use spaces to separate the elements. If an element is a symbol, simply write the symbol's name—no other delimiters or punctuation are needed. If a vector element is a character, write it as a Lisp character constant: '`?`' followed by the character as it would appear in a string.

Here are examples of using vectors to rebind `C-=` (a control character outside of ASCII), `H-a` (a Hyper character; ASCII doesn't have Hyper at all), ⟨F7⟩ (a function key), and `C-Mouse-1` (a keyboard-modified mouse button):

```
(global-set-key [?\C-=] 'make-symbolic-link)
(global-set-key [?\H-a] 'make-symbolic-link)
(global-set-key [f7] 'make-symbolic-link)
(global-set-key [C-mouse-1] 'make-symbolic-link)
```

You can use a vector for the simple cases too. Here's how to rewrite the first two examples, above, to use vectors:

```
(global-set-key [?\C-z] 'shell)

(global-set-key [?\C-x ?l] 'make-symbolic-link)
```

31.4.7 Rebinding Function Keys

Key sequences can contain function keys as well as ordinary characters. Just as Lisp characters (actually integers) represent keyboard characters, Lisp symbols represent function keys. If the function key has a word as its

label, then that word is also the name of the corresponding Lisp symbol. Here are the conventional Lisp names for common function keys:

`left`, `up`, `right`, `down`
: Cursor arrow keys.

`begin`, `end`, `home`, `next`, `prior`
: Other cursor repositioning keys.

`select`, `print`, `execute`, `backtab`
`insert`, `undo`, `redo`, `clearline`
`insertline`, `deleteline`, `insertchar`, `deletechar`,
: Miscellaneous function keys.

`f1`, `f2`, ... `f35`
: Numbered function keys (across the top of the keyboard).

`kp-add`, `kp-subtract`, `kp-multiply`, `kp-divide`
`kp-backtab`, `kp-space`, `kp-tab`, `kp-enter`
`kp-separator`, `kp-decimal`, `kp-equal`
: Keypad keys (to the right of the regular keyboard), with names or punctuation.

`kp-0`, `kp-1`, ... `kp-9`
: Keypad keys with digits.

`kp-f1`, `kp-f2`, `kp-f3`, `kp-f4`
: Keypad PF keys.

These names are conventional, but some systems (especially when using X) may use different names. To make certain what symbol is used for a given function key on your terminal, type `C-h c` followed by that key.

A key sequence which contains function key symbols (or anything but ASCII characters) must be a vector rather than a string. The vector syntax uses spaces between the elements, and square brackets around the whole vector. Thus, to bind function key '`f1`' to the command `rmail`, write the following:

```
(global-set-key [f1] 'rmail)
```

To bind the right-arrow key to the command `forward-char`, you can use this expression:

```
(global-set-key [right] 'forward-char)
```

This uses the Lisp syntax for a vector containing the symbol `right`. (This binding is present in Emacs by default.)

See Section 31.4.6 [Init Rebinding], page 479, for more information about using vectors for rebinding.

You can mix function keys and characters in a key sequence. This example binds `C-x` ⟨NEXT⟩ to the command `forward-page`.

```
(global-set-key [?\C-x next] 'forward-page)
```

where `?\C-x` is the Lisp character constant for the character `C-x`. The vector element `next` is a symbol and therefore does not take a question mark.

You can use the modifier keys ⟨CTRL⟩, ⟨META⟩, ⟨HYPER⟩, ⟨SUPER⟩, ⟨ALT⟩ and ⟨SHIFT⟩ with function keys. To represent these modifiers, add the strings '`C-`', '`M-`', '`H-`', '`s-`', '`A-`' and '`S-`' at the front of the symbol name. Thus, here is how to make `Hyper-Meta-`⟨RIGHT⟩ move forward a word:

```
(global-set-key [H-M-right] 'forward-word)
```

31.4.8 Named ASCII Control Characters

⟨TAB⟩, ⟨RET⟩, ⟨BS⟩, ⟨LFD⟩, ⟨ESC⟩ and ⟨DEL⟩ started out as names for certain ASCII control characters, used so often that they have special keys of their own. Later, users found it convenient to distinguish in Emacs between these keys and the "same" control characters typed with the ⟨CTRL⟩ key.

Emacs distinguishes these two kinds of input, when the keyboard reports these keys to Emacs. It treats the "special" keys as function keys named `tab`, `return`, `backspace`, `linefeed`, `escape`, and `delete`. These function keys translate automatically into the corresponding ASCII characters *if* they have no bindings of their own. As a result, neither users nor Lisp programs need to pay attention to the distinction unless they care to.

If you do not want to distinguish between (for example) ⟨TAB⟩ and `C-i`, make just one binding, for the ASCII character ⟨TAB⟩ (octal code 011). If you do want to distinguish, make one binding for this ASCII character, and another for the "function key" `tab`.

With an ordinary ASCII terminal, there is no way to distinguish between ⟨TAB⟩ and `C-i` (and likewise for other such pairs), because the terminal sends the same character in both cases.

31.4.9 Non-ASCII Characters on the Keyboard

If your keyboard has keys that send non-ASCII characters, such as accented letters, rebinding these keys is a bit tricky. There are two solutions you can use. One is to specify a keyboard coding system, using `set-keyboard-coding-system` (see Section 18.8 [Specify Coding], page 227). Then you can bind these keys in the usual way[1], like this:

```
(global-set-key [?char] 'some-function)
```

Type `C-q` followed by the key you want to bind, to insert *char*.

If you don't specify the keyboard coding system, that approach won't work. Instead, you need to find out the actual code that

[1] Note that you should avoid the string syntax for binding 8-bit characters, since they will be interpreted as meta keys. See section "Strings of Events" in *The Emacs Lisp Reference Manual*.

the terminal sends. The easiest way to do this in Emacs is to create an empty buffer with `C-x b temp` ⟨RET⟩, make it unibyte with `M-x toggle-enable-multibyte-characters` ⟨RET⟩, then type the key to insert the character into this buffer.

Move point before the character, then type `C-x =`. This displays a message in the minibuffer, showing the character code in three ways, octal, decimal and hexadecimal, all within a set of parentheses. Use the second of the three numbers, the decimal one, inside the vector to bind:

```
(global-set-key [decimal-code] 'some-function)
```

If you bind 8-bit characters like this in your init file, you my find it convenient to specify that it is unibyte. See Section 18.2 [Enabling Multibyte], page 219.

31.4.10 Rebinding Mouse Buttons

Emacs uses Lisp symbols to designate mouse buttons, too. The ordinary mouse events in Emacs are *click* events; these happen when you press a button and release it without moving the mouse. You can also get *drag* events, when you move the mouse while holding the button down. Drag events happen when you finally let go of the button.

The symbols for basic click events are `mouse-1` for the leftmost button, `mouse-2` for the next, and so on. Here is how you can redefine the second mouse button to split the current window:

```
(global-set-key [mouse-2] 'split-window-vertically)
```

The symbols for drag events are similar, but have the prefix '`drag-`' before the word '`mouse`'. For example, dragging the first button generates a `drag-mouse-1` event.

You can also define bindings for events that occur when a mouse button is pressed down. These events start with '`down-`' instead of '`drag-`'. Such events are generated only if they have key bindings. When you get a button-down event, a corresponding click or drag event will always follow.

If you wish, you can distinguish single, double, and triple clicks. A double click means clicking a mouse button twice in approximately the same place. The first click generates an ordinary click event. The second click, if it comes soon enough, generates a double-click event instead. The event type for a double-click event starts with '`double-`': for example, `double-mouse-3`.

This means that you can give a special meaning to the second click at the same place, but it must act on the assumption that the ordinary single click definition has run when the first click was received.

This constrains what you can do with double clicks, but user interface designers say that this constraint ought to be followed in any case. A double click should do something similar to the single click, only "more so." The

command for the double-click event should perform the extra work for the double click.

If a double-click event has no binding, it changes to the corresponding single-click event. Thus, if you don't define a particular double click specially, it executes the single-click command twice.

Emacs also supports triple-click events whose names start with '`triple-`'. Emacs does not distinguish quadruple clicks as event types; clicks beyond the third generate additional triple-click events. However, the full number of clicks is recorded in the event list, so you can distinguish if you really want to. We don't recommend distinct meanings for more than three clicks, but sometimes it is useful for subsequent clicks to cycle through the same set of three meanings, so that four clicks are equivalent to one click, five are equivalent to two, and six are equivalent to three.

Emacs also records multiple presses in drag and button-down events. For example, when you press a button twice, then move the mouse while holding the button, Emacs gets a '`double-drag-`' event. And at the moment when you press it down for the second time, Emacs gets a '`double-down-`' event (which is ignored, like all button-down events, if it has no binding).

The variable `double-click-time` specifies how long may elapse between clicks that are recognized as a pair. Its value is measured in milliseconds. If the value is `nil`, double clicks are not detected at all. If the value is `t`, then there is no time limit.

The symbols for mouse events also indicate the status of the modifier keys, with the usual prefixes '`C-`', '`M-`', '`H-`', '`s-`', '`A-`' and '`S-`'. These always precede '`double-`' or '`triple-`', which always precede '`drag-`' or '`down-`'.

A frame includes areas that don't show text from the buffer, such as the mode line and the scroll bar. You can tell whether a mouse button comes from a special area of the screen by means of dummy "prefix keys." For example, if you click the mouse in the mode line, you get the prefix key `mode-line` before the ordinary mouse-button symbol. Thus, here is how to define the command for clicking the first button in a mode line to run `scroll-up`:

```
(global-set-key [mode-line mouse-1] 'scroll-up)
```

Here is the complete list of these dummy prefix keys and their meanings:

`mode-line`
: The mouse was in the mode line of a window.

`vertical-line`
: The mouse was in the vertical line separating side-by-side windows. (If you use scroll bars, they appear in place of these vertical lines.)

`vertical-scroll-bar`
: The mouse was in a vertical scroll bar. (This is the only kind of scroll bar Emacs currently supports.)

You can put more than one mouse button in a key sequence, but it isn't usual to do so.

31.4.11 Disabling Commands

Disabling a command marks the command as requiring confirmation before it can be executed. The purpose of disabling a command is to prevent beginning users from executing it by accident and being confused.

An attempt to invoke a disabled command interactively in Emacs displays a window containing the command's name, its documentation, and some instructions on what to do immediately; then Emacs asks for input saying whether to execute the command as requested, enable it and execute it, or cancel. If you decide to enable the command, you are asked whether to do this permanently or just for the current session. Enabling permanently works by automatically editing your '`.emacs`' file.

The direct mechanism for disabling a command is to put a non-`nil` `disabled` property on the Lisp symbol for the command. Here is the Lisp program to do this:

```
(put 'delete-region 'disabled t)
```

If the value of the `disabled` property is a string, that string is included in the message printed when the command is used:

```
(put 'delete-region 'disabled
     "It's better to use `kill-region' instead.\n")
```

You can make a command disabled either by editing the '`.emacs`' file directly or with the command `M-x disable-command`, which edits the '`.emacs`' file for you. Likewise, `M-x enable-command` edits '`.emacs`' to enable a command permanently. See Section 31.7 [Init File], page 486.

Whether a command is disabled is independent of what key is used to invoke it; disabling also applies if the command is invoked using `M-x`. Disabling a command has no effect on calling it as a function from Lisp programs.

31.5 Keyboard Translations

Some keyboards do not make it convenient to send all the special characters that Emacs uses. The most common problem case is the ⟨DEL⟩ character. Some keyboards provide no convenient way to type this very important character—usually because they were designed to expect the character `C-h` to be used for deletion. On these keyboards, if you press the key normally used for deletion, Emacs handles the `C-h` as a prefix character and offers you a list of help options, which is not what you want.

You can work around this problem within Emacs by setting up keyboard translations to turn `C-h` into ⟨DEL⟩ and ⟨DEL⟩ into `C-h`, as follows:

```
;; Translate C-h to ⟨DEL⟩.
(keyboard-translate ?\C-h ?\C-?)

;; Translate ⟨DEL⟩ to C-h.
(keyboard-translate ?\C-? ?\C-h)
```

Keyboard translations are not the same as key bindings in keymaps (see Section 31.4.1 [Keymaps], page 474). Emacs contains numerous keymaps that apply in different situations, but there is only one set of keyboard translations, and it applies to every character that Emacs reads from the terminal. Keyboard translations take place at the lowest level of input processing; the keys that are looked up in keymaps contain the characters that result from keyboard translation.

On a window system, the keyboard key named ⟨DELETE⟩ is a function key and is distinct from the ASCII character named ⟨DEL⟩. See Section 31.4.8 [Named ASCII Chars], page 481. Keyboard translations affect only ASCII character input, not function keys; thus, the above example used on a window system does not affect the ⟨DELETE⟩ key. However, the translation above isn't necessary on window systems, because Emacs can also distinguish between the ⟨BACKSPACE⟩ key and `C-h`; and it normally treats ⟨BACKSPACE⟩ as ⟨DEL⟩.

For full information about how to use keyboard translations, see section "Translating Input" in *The Emacs Lisp Reference Manual*.

31.6 The Syntax Table

All the Emacs commands which parse words or balance parentheses are controlled by the *syntax table*. The syntax table says which characters are opening delimiters, which are parts of words, which are string quotes, and so on. Each major mode has its own syntax table (though sometimes related major modes use the same one) which it installs in each buffer that uses that major mode. The syntax table installed in the current buffer is the one that all commands use, so we call it "the" syntax table. A syntax table is a Lisp object, a char-table, whose elements are numbers.

To display a description of the contents of the current syntax table, type `C-h s` (`describe-syntax`). The description of each character includes both the string you would have to give to `modify-syntax-entry` to set up that character's current syntax, and some English to explain that string if necessary.

For full information on the syntax table, see section "Syntax Tables" in *The Emacs Lisp Reference Manual*.

31.7 The Init File, '`~/.emacs`'

When Emacs is started, it normally loads a Lisp program from the file '`.emacs`' or '`.emacs.el`' in your home directory. We call this file your *init file* because it specifies how to initialize Emacs for you. You can use the command line switch '`-q`' to prevent loading your init file, and '`-u`' (or '`--user`') to specify a different user's init file (see Chapter 3 [Entering Emacs], page 37).

There can also be a *default init file*, which is the library named '`default.el`', found via the standard search path for libraries. The Emacs distribution contains no such library; your site may create one for local customizations. If this library exists, it is loaded whenever you start Emacs (except when you specify '`-q`'). But your init file, if any, is loaded first; if it sets `inhibit-default-init` non-`nil`, then '`default`' is not loaded.

Your site may also have a *site startup file*; this is named '`site-start.el`', if it exists. Emacs loads this library before it loads your init file. To inhibit loading of this library, use the option '`-no-site-file`'. See Section B.2 [Initial Options], page 508.

If you have a large amount of code in your '`.emacs`' file, you should rename it to '`~/.emacs.el`', and byte-compile it. See section "Byte Compilation" in *the Emacs Lisp Reference Manual*, for more information about compiling Emacs Lisp programs.

If you are going to write actual Emacs Lisp programs that go beyond minor customization, you should read the *Emacs Lisp Reference Manual*.

31.7.1 Init File Syntax

The '`.emacs`' file contains one or more Lisp function call expressions. Each of these consists of a function name followed by arguments, all surrounded by parentheses. For example, `(setq fill-column 60)` calls the function `setq` to set the variable `fill-column` (see Section 21.5 [Filling], page 248) to 60.

The second argument to `setq` is an expression for the new value of the variable. This can be a constant, a variable, or a function call expression. In '`.emacs`', constants are used most of the time. They can be:

Numbers: Numbers are written in decimal, with an optional initial minus sign.

Strings: Lisp string syntax is the same as C string syntax with a few extra features. Use a double-quote character to begin and end a string constant.

In a string, you can include newlines and special characters literally. But often it is cleaner to use backslash sequences for them: '`\n`' for newline, '`\b`' for backspace, '`\r`' for carriage return, '`\t`' for tab, '`\f`' for formfeed (control-L), '`\e`' for escape, '`\\`' for a

backslash, '`\"`' for a double-quote, or '`\`*ooo*' for the character whose octal code is *ooo*. Backslash and double-quote are the only characters for which backslash sequences are mandatory.

'`\C-`' can be used as a prefix for a control character, as in '`\C-s`' for ASCII control-S, and '`\M-`' can be used as a prefix for a Meta character, as in '`\M-a`' for `Meta-A` or '`\M-\C-a`' for `Control-Meta-A`.

Characters:
: Lisp character constant syntax consists of a '`?`' followed by either a character or an escape sequence starting with '`\`'. Examples: `?x`, `?\n`, `?\"`, `?\)`. Note that strings and characters are not interchangeable in Lisp; some contexts require one and some contexts require the other.

True:
: `t` stands for 'true'.

False:
: `nil` stands for 'false'.

Other Lisp objects:
: Write a single-quote (') followed by the Lisp object you want.

31.7.2 Init File Examples

Here are some examples of doing certain commonly desired things with Lisp expressions:

- Make ⟨TAB⟩ in C mode just insert a tab if point is in the middle of a line.

  ```
  (setq c-tab-always-indent nil)
  ```

  Here we have a variable whose value is normally `t` for 'true' and the alternative is `nil` for 'false'.

- Make searches case sensitive by default (in all buffers that do not override this).

  ```
  (setq-default case-fold-search nil)
  ```

  This sets the default value, which is effective in all buffers that do not have local values for the variable. Setting `case-fold-search` with `setq` affects only the current buffer's local value, which is not what you probably want to do in an init file.

- Specify your own email address, if Emacs can't figure it out correctly.

  ```
  (setq user-mail-address "coon@yoyodyne.com")
  ```

  Various Emacs packages that need your own email address use the value of `user-mail-address`.

- Make Text mode the default mode for new buffers.

```
(setq default-major-mode 'text-mode)
```

Note that `text-mode` is used because it is the command for entering Text mode. The single-quote before it makes the symbol a constant; otherwise, `text-mode` would be treated as a variable name.

- Set up defaults for the Latin-1 character set which supports most of the languages of Western Europe.

```
(set-language-environment "Latin-1")
```

- Turn on Auto Fill mode automatically in Text mode and related modes.

```
(add-hook 'text-mode-hook
  '(lambda () (auto-fill-mode 1)))
```

This shows how to add a hook function to a normal hook variable (see Section 31.2.3 [Hooks], page 465). The function we supply is a list starting with `lambda`, with a single-quote in front of it to make it a list constant rather than an expression.

It's beyond the scope of this manual to explain Lisp functions, but for this example it is enough to know that the effect is to execute `(auto-fill-mode 1)` when Text mode is entered. You can replace that with any other expression that you like, or with several expressions in a row.

Emacs comes with a function named `turn-on-auto-fill` whose definition is `(lambda () (auto-fill-mode 1))`. Thus, a simpler way to write the above example is as follows:

```
(add-hook 'text-mode-hook 'turn-on-auto-fill)
```

- Load the installed Lisp library named '`foo`' (actually a file '`foo.elc`' or '`foo.el`' in a standard Emacs directory).

```
(load "foo")
```

When the argument to `load` is a relative file name, not starting with '`/`' or '`~`', `load` searches the directories in `load-path` (see Section 23.7 [Lisp Libraries], page 339).

- Load the compiled Lisp file '`foo.elc`' from your home directory.

```
(load "~/foo.elc")
```

Here an absolute file name is used, so no searching is done.

- Rebind the key `C-x l` to run the function `make-symbolic-link`.

```
(global-set-key "\C-xl" 'make-symbolic-link)
```

or

```
(define-key global-map "\C-xl" 'make-symbolic-link)
```

Note once again the single-quote used to refer to the symbol `make-symbolic-link` instead of its value as a variable.

- Do the same thing for Lisp mode only.

```
(define-key lisp-mode-map "\C-xl" 'make-symbolic-link)
```

- Redefine all keys which now run `next-line` in Fundamental mode so that they run `forward-line` instead.

```
(substitute-key-definition 'next-line 'forward-line
                           global-map)
```

- Make `C-x C-v` undefined.

  ```
  (global-unset-key "\C-x\C-v")
  ```

  One reason to undefine a key is so that you can make it a prefix. Simply defining `C-x C-v` *anything* will make `C-x C-v` a prefix, but `C-x C-v` must first be freed of its usual non-prefix definition.

- Make '`$`' have the syntax of punctuation in Text mode. Note the use of a character constant for '`$`'.

  ```
  (modify-syntax-entry ?\$ "." text-mode-syntax-table)
  ```

- Enable the use of the command `narrow-to-region` without confirmation.

  ```
  (put 'narrow-to-region 'disabled nil)
  ```

31.7.3 Terminal-specific Initialization

Each terminal type can have a Lisp library to be loaded into Emacs when it is run on that type of terminal. For a terminal type named *termtype*, the library is called '`term/`*termtype*' and it is found by searching the directories `load-path` as usual and trying the suffixes '`.elc`' and '`.el`'. Normally it appears in the subdirectory '`term`' of the directory where most Emacs libraries are kept.

The usual purpose of the terminal-specific library is to map the escape sequences used by the terminal's function keys onto more meaningful names, using `function-key-map`. See the file '`term/lk201.el`' for an example of how this is done. Many function keys are mapped automatically according to the information in the Termcap data base; the terminal-specific library needs to map only the function keys that Termcap does not specify.

When the terminal type contains a hyphen, only the part of the name before the first hyphen is significant in choosing the library name. Thus, terminal types '`aaa-48`' and '`aaa-30-rv`' both use the library '`term/aaa`'. The code in the library can use `(getenv "TERM")` to find the full terminal type name.

The library's name is constructed by concatenating the value of the variable `term-file-prefix` and the terminal type. Your '`.emacs`' file can prevent the loading of the terminal-specific library by setting `term-file-prefix` to `nil`.

Emacs runs the hook `term-setup-hook` at the end of initialization, after both your '`.emacs`' file and any terminal-specific library have been read in. Add hook functions to this hook if you wish to override part of any of the terminal-specific libraries and to define initializations for terminals that do not have a library. See Section 31.2.3 [Hooks], page 465.

31.7.4 How Emacs Finds Your Init File

Normally Emacs uses the environment variable `HOME` to find '`.emacs`'; that's what '`~`' means in a file name. But if you have done `su`, Emacs tries to find your own '`.emacs`', not that of the user you are currently pretending to be. The idea is that you should get your own editor customizations even if you are running as the super user.

More precisely, Emacs first determines which user's init file to use. It gets the user name from the environment variables `LOGNAME` and `USER`; if neither of those exists, it uses effective user-ID. If that user name matches the real user-ID, then Emacs uses `HOME`; otherwise, it looks up the home directory corresponding to that user name in the system's data base of users.

32 Dealing with Common Problems

If you type an Emacs command you did not intend, the results are often mysterious. This chapter tells what you can do to cancel your mistake or recover from a mysterious situation. Emacs bugs and system crashes are also considered.

32.1 Quitting and Aborting

`C-g`
`C-`⟨BREAK⟩ (MS-DOS only)
: Quit: cancel running or partially typed command.

`C-]`
: Abort innermost recursive editing level and cancel the command which invoked it (`abort-recursive-edit`).

⟨ESC⟩ ⟨ESC⟩ ⟨ESC⟩
: Either quit or abort, whichever makes sense (`keyboard-escape-quit`).

`M-x top-level`
: Abort all recursive editing levels that are currently executing.

`C-x u`
: Cancel a previously made change in the buffer contents (`undo`).

There are two ways of canceling commands which are not finished executing: *quitting* with `C-g`, and *aborting* with `C-]` or `M-x top-level`. Quitting cancels a partially typed command or one which is already running. Aborting exits a recursive editing level and cancels the command that invoked the recursive edit. (See Section 30.13 [Recursive Edit], page 447.)

Quitting with `C-g` is used for getting rid of a partially typed command, or a numeric argument that you don't want. It also stops a running command in the middle in a relatively safe way, so you can use it if you accidentally give a command which takes a long time. In particular, it is safe to quit out of killing; either your text will *all* still be in the buffer, or it will *all* be in the kill ring (or maybe both). Quitting an incremental search does special things documented under searching; in general, it may take two successive `C-g` characters to get out of a search (see Section 12.1 [Incremental Search], page 119).

On MS-DOS, the character `C-`⟨BREAK⟩ serves as a quit character like `C-g`. The reason is that it is not feasible, on MS-DOS, to recognize `C-g` while a command is running, between interactions with the user. By contrast, it *is* feasible to recognize `C-`⟨BREAK⟩ at all times. See Section E.1 [MS-DOS Input], page 539.

`C-g` works by setting the variable `quit-flag` to `t` the instant `C-g` is typed; Emacs Lisp checks this variable frequently and quits if it is non-`nil`. `C-g` is

only actually executed as a command if you type it while Emacs is waiting for input. In that case, the command it runs is `keyboard-quit`.

If you quit with `C-g` a second time before the first `C-g` is recognized, you activate the "emergency escape" feature and return to the shell. See Section 32.2.8 [Emergency Escape], page 496.

There may be times when you cannot quit. When Emacs is waiting for the operating system to do something, quitting is impossible unless special pains are taken for the particular system call within Emacs where the waiting occurs. We have done this for the system calls that users are likely to want to quit from, but it's possible you will find another. In one very common case—waiting for file input or output using NFS—Emacs itself knows how to quit, but most NFS implementations simply do not allow user programs to stop waiting for NFS when the NFS server is hung.

Aborting with `C-]` (`abort-recursive-edit`) is used to get out of a recursive editing level and cancel the command which invoked it. Quitting with `C-g` does not do this, and could not do this, because it is used to cancel a partially typed command *within* the recursive editing level. Both operations are useful. For example, if you are in a recursive edit and type `C-u 8` to enter a numeric argument, you can cancel that argument with `C-g` and remain in the recursive edit.

The command ⟨ESC⟩ ⟨ESC⟩ ⟨ESC⟩ (`keyboard-escape-quit`) can either quit or abort. This key was defined because ⟨ESC⟩ is used to "get out" in many PC programs. It can cancel a prefix argument, clear a selected region, or get out of a Query Replace, like `C-g`. It can get out of the minibuffer or a recursive edit, like `C-]`. It can also get out of splitting the frame into multiple windows, like `C-x 1`. One thing it cannot do, however, is stop a command that is running. That's because it executes as an ordinary command, and Emacs doesn't notice it until it is ready for a command.

The command `M-x top-level` is equivalent to "enough" `C-]` commands to get you out of all the levels of recursive edits that you are in. `C-]` gets you out one level at a time, but `M-x top-level` goes out all levels at once. Both `C-]` and `M-x top-level` are like all other commands, and unlike `C-g`, in that they take effect only when Emacs is ready for a command. `C-]` is an ordinary key and has its meaning only because of its binding in the keymap. See Section 30.13 [Recursive Edit], page 447.

`C-x u` (`undo`) is not strictly speaking a way of canceling a command, but you can think of it as canceling a command that already finished executing. See Section 4.4 [Undo], page 45.

32.2 Dealing with Emacs Trouble

This section describes various conditions in which Emacs fails to work normally, and how to recognize them and correct them.

32.2.1 If ⟨DEL⟩ Fails to Delete

Every keyboard has a large key, a little ways above the ⟨RET⟩ or ⟨ENTER⟩ key, which you normally use outside Emacs to erase the last character that you typed. We call this key ⟨DEL⟩.

When Emacs starts up using a window system, it determines automatically which key should be ⟨DEL⟩. In some unusual cases Emacs gets the wrong information from the system. If the ⟨DEL⟩ key deletes forwards instead of backwards, that is probably what happened—Emacs ought to be treating the ⟨DELETE⟩ key as ⟨DEL⟩, but it isn't.

With a window system, if the ⟨DEL⟩ key says ⟨BACKSPACE⟩ and there is a ⟨DELETE⟩ key elsewhere, but the ⟨DELETE⟩ key deletes backward instead of forward, that too suggests Emacs got the wrong information—but in the opposite sense. It ought to be treating the ⟨BACKSPACE⟩ key as ⟨DEL⟩, but it isn't.

On a text-only terminal, if you find the ⟨DEL⟩ key prompts for a Help command like `Control-h`, instead of deleting a character, it means that key is actually sending the ⟨BS⟩ character. Emacs ought to be treating ⟨BS⟩ as ⟨DEL⟩, but it isn't.

In all of those cases, the immediate remedy is the same: use the command `M-x normal-erase-is-backspace-mode`. That should make the proper ⟨DEL⟩ key work. On a text-only terminal, if you do want to ask for help, use ⟨F1⟩ or `C-?`.

To fix the problem automatically for every Emacs session, you can put one of the following lines into your '`.emacs`' file (see Section 31.7 [Init File], page 486). For the first case above, where ⟨DEL⟩ deletes forwards instead of backwards, use this line:

```
(normal-erase-is-backspace-mode 0)
```

For the other two cases, use this line:

```
(normal-erase-is-backspace-mode 1)
```

Another way to fix the problem for every Emacs session is to customize the variable `normal-erase-is-backspace`: the value `t` specifies the mode where ⟨BS⟩ or ⟨BACKSPACE⟩ is ⟨DEL⟩, and `nil` specifies the other mode. See Section 31.2.2 [Easy Customization], page 459.

32.2.2 Recursive Editing Levels

Recursive editing levels are important and useful features of Emacs, but they can seem like malfunctions to the user who does not understand them.

If the mode line has square brackets '`[...]`' around the parentheses that contain the names of the major and minor modes, you have entered a recursive editing level. If you did not do this on purpose, or if you don't

understand what that means, you should just get out of the recursive editing level. To do so, type `M-x top-level`. This is called getting back to top level. See Section 30.13 [Recursive Edit], page 447.

32.2.3 Garbage on the Screen

If the data on the screen looks wrong, the first thing to do is see whether the text is really wrong. Type `C-l` to redisplay the entire screen. If the screen appears correct after this, the problem was entirely in the previous screen update. (Otherwise, see Section 32.2.4 [Text Garbled], page 494.)

Display updating problems often result from an incorrect termcap entry for the terminal you are using. The file '`etc/TERMS`' in the Emacs distribution gives the fixes for known problems of this sort. '`INSTALL`' contains general advice for these problems in one of its sections. Very likely there is simply insufficient padding for certain display operations. To investigate the possibility that you have this sort of problem, try Emacs on another terminal made by a different manufacturer. If problems happen frequently on one kind of terminal but not another kind, it is likely to be a bad termcap entry, though it could also be due to a bug in Emacs that appears for terminals that have or that lack specific features.

32.2.4 Garbage in the Text

If `C-l` shows that the text is wrong, try undoing the changes to it using `C-x u` until it gets back to a state you consider correct. Also try `C-h l` to find out what command you typed to produce the observed results.

If a large portion of text appears to be missing at the beginning or end of the buffer, check for the word '`Narrow`' in the mode line. If it appears, the text you don't see is probably still present, but temporarily off-limits. To make it accessible again, type `C-x n w`. See Section 30.9 [Narrowing], page 443.

32.2.5 Spontaneous Entry to Incremental Search

If Emacs spontaneously displays '`I-search:`' at the bottom of the screen, it means that the terminal is sending `C-s` and `C-q` according to the poorly designed xon/xoff "flow control" protocol.

If this happens to you, your best recourse is to put the terminal in a mode where it will not use flow control, or give it so much padding that it will never send a `C-s`. (One way to increase the amount of padding is to set the variable `baud-rate` to a larger value. Its value is the terminal output speed, measured in the conventional units of baud.)

If you don't succeed in turning off flow control, the next best thing is to tell Emacs to cope with it. To do this, call the function `enable-flow-control`.

Typically there are particular terminal types with which you must use flow control. You can conveniently ask for flow control on those terminal types only, using `enable-flow-control-on`. For example, if you find you must use flow control on VT-100 and H19 terminals, put the following in your '`.emacs`' file:

```
(enable-flow-control-on "vt100" "h19")
```

When flow control is enabled, you must type `C-\` to get the effect of a `C-s`, and type `C-^` to get the effect of a `C-q`. (These aliases work by means of keyboard translations; see Section 31.5 [Keyboard Translations], page 484.)

32.2.6 Running out of Memory

If you get the error message '`Virtual memory exceeded`', save your modified buffers with `C-x s`. This method of saving them has the smallest need for additional memory. Emacs keeps a reserve of memory which it makes available when this error happens; that should be enough to enable `C-x s` to complete its work.

Once you have saved your modified buffers, you can exit this Emacs job and start another, or you can use `M-x kill-some-buffers` to free space in the current Emacs job. If you kill buffers containing a substantial amount of text, you can safely go on editing. Emacs refills its memory reserve automatically when it sees sufficient free space available, in case you run out of memory another time.

Do not use `M-x buffer-menu` to save or kill buffers when you run out of memory, because the buffer menu needs a fair amount memory itself, and the reserve supply may not be enough.

32.2.7 Recovery After a Crash

If Emacs or the computer crashes, you can recover the files you were editing at the time of the crash from their auto-save files. To do this, start Emacs again and type the command `M-x recover-session`.

This command initially displays a buffer which lists interrupted session files, each with its date. You must choose which session to recover from. Typically the one you want is the most recent one. Move point to the one you choose, and type `C-c C-c`.

Then `recover-session` asks about each of the files that you were editing during that session; it asks whether to recover that file. If you answer `y` for a file, it shows the dates of that file and its auto-save file, then asks once again

whether to recover that file. For the second question, you must confirm with `yes`. If you do, Emacs visits the file but gets the text from the auto-save file.

When `recover-session` is done, the files you've chosen to recover are present in Emacs buffers. You should then save them. Only this—saving them—updates the files themselves.

32.2.8 Emergency Escape

Because at times there have been bugs causing Emacs to loop without checking `quit-flag`, a special feature causes Emacs to be suspended immediately if you type a second `C-g` while the flag is already set, so you can always get out of GNU Emacs. Normally Emacs recognizes and clears `quit-flag` (and quits!) quickly enough to prevent this from happening. (On MS-DOS and compatible systems, type `C-`⟨BREAK⟩ twice.)

When you resume Emacs after a suspension caused by multiple `C-g`, it asks two questions before going back to what it had been doing:

```
Auto-save? (y or n)
Abort (and dump core)? (y or n)
```

Answer each one with `y` or `n` followed by ⟨RET⟩.

Saying `y` to '`Auto-save?`' causes immediate auto-saving of all modified buffers in which auto-saving is enabled.

Saying `y` to '`Abort (and dump core)?`' causes an illegal instruction to be executed, dumping core. This is to enable a wizard to figure out why Emacs was failing to quit in the first place. Execution does not continue after a core dump. If you answer `n`, execution does continue. With luck, GNU Emacs will ultimately check `quit-flag` and quit normally. If not, and you type another `C-g`, it is suspended again.

If Emacs is not really hung, just slow, you may invoke the double `C-g` feature without really meaning to. Then just resume and answer `n` to both questions, and you will arrive at your former state. Presumably the quit you requested will happen soon.

The double-`C-g` feature is turned off when Emacs is running under the X Window System, since you can use the window manager to kill Emacs or to create another window and run another program.

On MS-DOS and compatible systems, the emergency escape feature is sometimes unavailable, even if you press `C-`⟨BREAK⟩ twice, when some system call (MS-DOS or BIOS) hangs, or when Emacs is stuck in a very tight endless loop (in C code, **not** in Lisp code).

32.2.9 Help for Total Frustration

If using Emacs (or something else) becomes terribly frustrating and none of the techniques described above solve the problem, Emacs can still help you.

First, if the Emacs you are using is not responding to commands, type `C-g C-g` to get out of it and then start a new one.

Second, type `M-x doctor` ⟨RET⟩.

The doctor will help you feel better. Each time you say something to the doctor, you must end it by typing ⟨RET⟩ ⟨RET⟩. This lets the doctor know you are finished.

32.3 Reporting Bugs

Sometimes you will encounter a bug in Emacs. Although we cannot promise we can or will fix the bug, and we might not even agree that it is a bug, we want to hear about problems you encounter. Often we agree they are bugs and want to fix them.

To make it possible for us to fix a bug, you must report it. In order to do so effectively, you must know when and how to do it.

32.3.1 When Is There a Bug

If Emacs executes an illegal instruction, or dies with an operating system error message that indicates a problem in the program (as opposed to something like "disk full"), then it is certainly a bug.

If Emacs updates the display in a way that does not correspond to what is in the buffer, then it is certainly a bug. If a command seems to do the wrong thing but the problem corrects itself if you type `C-l`, it is a case of incorrect display updating.

Taking forever to complete a command can be a bug, but you must make certain that it was really Emacs's fault. Some commands simply take a long time. Type `C-g` (`C-`⟨BREAK⟩ on MS-DOS) and then `C-h l` to see whether the input Emacs received was what you intended to type; if the input was such that you *know* it should have been processed quickly, report a bug. If you don't know whether the command should take a long time, find out by looking in the manual or by asking for assistance.

If a command you are familiar with causes an Emacs error message in a case where its usual definition ought to be reasonable, it is probably a bug.

If a command does the wrong thing, that is a bug. But be sure you know for certain what it ought to have done. If you aren't familiar with the command, or don't know for certain how the command is supposed to work,

then it might actually be working right. Rather than jumping to conclusions, show the problem to someone who knows for certain.

Finally, a command's intended definition may not be the best possible definition for editing with. This is a very important sort of problem, but it is also a matter of judgment. Also, it is easy to come to such a conclusion out of ignorance of some of the existing features. It is probably best not to complain about such a problem until you have checked the documentation in the usual ways, feel confident that you understand it, and know for certain that what you want is not available. If you are not sure what the command is supposed to do after a careful reading of the manual, check the index and glossary for any terms that may be unclear.

If after careful rereading of the manual you still do not understand what the command should do, that indicates a bug in the manual, which you should report. The manual's job is to make everything clear to people who are not Emacs experts—including you. It is just as important to report documentation bugs as program bugs.

If the on-line documentation string of a function or variable disagrees with the manual, one of them must be wrong; that is a bug.

32.3.2 Understanding Bug Reporting

When you decide that there is a bug, it is important to report it and to report it in a way which is useful. What is most useful is an exact description of what commands you type, starting with the shell command to run Emacs, until the problem happens.

The most important principle in reporting a bug is to report *facts*. Hypotheses and verbal descriptions are no substitute for the detailed raw data. Reporting the facts is straightforward, but many people strain to posit explanations and report them instead of the facts. If the explanations are based on guesses about how Emacs is implemented, they will be useless; meanwhile, lacking the facts, we will have no real information about the bug.

For example, suppose that you type `C-x C-f /glorp/baz.ugh` ⟨RET⟩, visiting a file which (you know) happens to be rather large, and Emacs displayed '`I feel pretty today`'. The best way to report the bug is with a sentence like the preceding one, because it gives all the facts.

A bad way would be to assume that the problem is due to the size of the file and say, "I visited a large file, and Emacs displayed '`I feel pretty today`'." This is what we mean by "guessing explanations." The problem is just as likely to be due to the fact that there is a '`z`' in the file name. If this is so, then when we got your report, we would try out the problem with some "large file," probably with no '`z`' in its name, and not see any problem. There is no way in the world that we could guess that we should try visiting a file with a '`z`' in its name.

Alternatively, the problem might be due to the fact that the file starts with exactly 25 spaces. For this reason, you should make sure that you inform us of the exact contents of any file that is needed to reproduce the bug. What if the problem only occurs when you have typed the `C-x C-a` command previously? This is why we ask you to give the exact sequence of characters you typed since starting the Emacs session.

You should not even say "visit a file" instead of `C-x C-f` unless you *know* that it makes no difference which visiting command is used. Similarly, rather than saying "if I have three characters on the line," say "after I type ⟨RET⟩ `A B C` ⟨RET⟩ `C-p`," if that is the way you entered the text.

So please don't guess any explanations when you report a bug. If you want to actually *debug* the problem, and report explanations that are more than guesses, that is useful—but please include the facts as well.

32.3.3 Checklist for Bug Reports

The best way to send a bug report is to mail it electronically to the Emacs maintainers at `bug-gnu-emacs@gnu.org`, or to `emacs-pretest-bug@gnu.org` if you are pretesting an Emacs beta release. (If you want to suggest a change as an improvement, use the same address.)

If you'd like to read the bug reports, you can find them on the newsgroup '`gnu.emacs.bug`'; keep in mind, however, that as a spectator you should not criticize anything about what you see there. The purpose of bug reports is to give information to the Emacs maintainers. Spectators are welcome only as long as they do not interfere with this. In particular, some bug reports contain large amounts of data; spectators should not complain about this.

Please do not post bug reports using netnews; mail is more reliable than netnews about reporting your correct address, which we may need in order to ask you for more information.

If you can't send electronic mail, then mail the bug report on paper or machine-readable media to this address:

GNU Emacs Bugs
Free Software Foundation
59 Temple Place, Suite 330
Boston, MA 02111-1307 USA

We do not promise to fix the bug; but if the bug is serious, or ugly, or easy to fix, chances are we will want to.

A convenient way to send a bug report for Emacs is to use the command `M-x report-emacs-bug`. This sets up a mail buffer (see Chapter 26 [Sending Mail], page 357) and automatically inserts *some* of the essential information. However, it cannot supply all the necessary information; you should still read and follow the guidelines below, so you can enter the other crucial information by hand before you send the message.

To enable maintainers to investigate a bug, your report should include all these things:

- The version number of Emacs. Without this, we won't know whether there is any point in looking for the bug in the current version of GNU Emacs.

  You can get the version number by typing `M-x emacs-version` ⟨RET⟩. If that command does not work, you probably have something other than GNU Emacs, so you will have to report the bug somewhere else.
- The type of machine you are using, and the operating system name and version number. `M-x emacs-version` ⟨RET⟩ provides this information too. Copy its output from the '`*Messages*`' buffer, so that you get it all and get it accurately.
- The operands given to the `configure` command when Emacs was installed.
- A complete list of any modifications you have made to the Emacs source. (We may not have time to investigate the bug unless it happens in an unmodified Emacs. But if you've made modifications and you don't tell us, you are sending us on a wild goose chase.)

  Be precise about these changes. A description in English is not enough—send a context diff for them.

  Adding files of your own, or porting to another machine, is a modification of the source.
- Details of any other deviations from the standard procedure for installing GNU Emacs.
- The complete text of any files needed to reproduce the bug.

  If you can tell us a way to cause the problem without visiting any files, please do so. This makes it much easier to debug. If you do need files, make sure you arrange for us to see their exact contents. For example, it can often matter whether there are spaces at the ends of lines, or a newline after the last line in the buffer (nothing ought to care whether the last line is terminated, but try telling the bugs that).
- The precise commands we need to type to reproduce the bug.

  The easy way to record the input to Emacs precisely is to write a dribble file. To start the file, execute the Lisp expression

  ```
  (open-dribble-file "~/dribble")
  ```

  using `M-:` or from the '`*scratch*`' buffer just after starting Emacs. From then on, Emacs copies all your input to the specified dribble file until the Emacs process is killed.
- For possible display bugs, the terminal type (the value of environment variable `TERM`), the complete termcap entry for the terminal from '`/etc/termcap`' (since that file is not identical on all machines), and the output that Emacs actually sent to the terminal.

  The way to collect the terminal output is to execute the Lisp expression

```
(open-termscript "~/termscript")
```

using `M-:` or from the '`*scratch*`' buffer just after starting Emacs. From then on, Emacs copies all terminal output to the specified termscript file as well, until the Emacs process is killed. If the problem happens when Emacs starts up, put this expression into your '`.emacs`' file so that the termscript file will be open when Emacs displays the screen for the first time.

Be warned: it is often difficult, and sometimes impossible, to fix a terminal-dependent bug without access to a terminal of the type that stimulates the bug.

- If non-ASCII text or internationalization is relevant, the locale that was current when you started Emacs. On GNU/Linux and Unix systems, or if you use a Unix-style shell such as Bash, you can use this shell command to view the relevant values:

  ```
  echo LC_ALL=$LC_ALL LC_CTYPE=$LC_CTYPE LANG=$LANG
  ```

  You can use the `M-!` command to execute the shell command from Emacs, and then copy the output from the '`*Messages*`' buffer into the bug report. Alternatively, `M-x getenv` ⟨RET⟩ `LC_ALL` ⟨RET⟩ will print the value of `LC_ALL` in the echo area, and you can copy its output from the '`*Messages*`' buffer.
- A description of what behavior you observe that you believe is incorrect. For example, "The Emacs process gets a fatal signal," or, "The resulting text is as follows, which I think is wrong."

  Of course, if the bug is that Emacs gets a fatal signal, then one can't miss it. But if the bug is incorrect text, the maintainer might fail to notice what is wrong. Why leave it to chance?

  Even if the problem you experience is a fatal signal, you should still say so explicitly. Suppose something strange is going on, such as, your copy of the source is out of sync, or you have encountered a bug in the C library on your system. (This has happened!) Your copy might crash and the copy here might not. If you *said* to expect a crash, then when Emacs here fails to crash, we would know that the bug was not happening. If you don't say to expect a crash, then we would not know whether the bug was happening—we would not be able to draw any conclusion from our observations.
- If the bug is that the Emacs Manual or the Emacs Lisp Reference Manual fails to describe the actual behavior of Emacs, or that the text is confusing, copy in the text from the online manual which you think is at fault. If the section is small, just the section name is enough.
- If the manifestation of the bug is an Emacs error message, it is important to report the precise text of the error message, and a backtrace showing how the Lisp program in Emacs arrived at the error.

  To get the error message text accurately, copy it from the '`*Messages*`' buffer into the bug report. Copy all of it, not just part.

To make a backtrace for the error, evaluate the Lisp expression `(setq debug-on-error t)` before the error happens (that is to say, you must execute that expression and then make the bug happen). This causes the error to run the Lisp debugger, which shows you a backtrace. Copy the text of the debugger's backtrace into the bug report.

This use of the debugger is possible only if you know how to make the bug happen again. If you can't make it happen again, at least copy the whole error message.

- Check whether any programs you have loaded into the Lisp world, including your '`.emacs`' file, set any variables that may affect the functioning of Emacs. Also, see whether the problem happens in a freshly started Emacs without loading your '`.emacs`' file (start Emacs with the `-q` switch to prevent loading the init file). If the problem does *not* occur then, you must report the precise contents of any programs that you must load into the Lisp world in order to cause the problem to occur.
- If the problem does depend on an init file or other Lisp programs that are not part of the standard Emacs system, then you should make sure it is not a bug in those programs by complaining to their maintainers first. After they verify that they are using Emacs in a way that is supposed to work, they should report the bug.
- If you wish to mention something in the GNU Emacs source, show the line of code with a few lines of context. Don't just give a line number.

  The line numbers in the development sources don't match those in your sources. It would take extra work for the maintainers to determine what code is in your version at a given line number, and we could not be certain.
- Additional information from a C debugger such as GDB might enable someone to find a problem on a machine which he does not have available. If you don't know how to use GDB, please read the GDB manual—it is not very long, and using GDB is easy. You can find the GDB distribution, including the GDB manual in online form, in most of the same places you can find the Emacs distribution. To run Emacs under GDB, you should switch to the '`src`' subdirectory in which Emacs was compiled, then do '`gdb emacs`'. It is important for the directory '`src`' to be current so that GDB will read the '`.gdbinit`' file in this directory.

  However, you need to think when you collect the additional information if you want it to show what causes the bug.

  For example, many people send just a backtrace, but that is not very useful by itself. A simple backtrace with arguments often conveys little about what is happening inside GNU Emacs, because most of the arguments listed in the backtrace are pointers to Lisp objects. The numeric values of these pointers have no significance whatever; all that matters is the contents of the objects they point to (and most of the contents are themselves pointers).

To provide useful information, you need to show the values of Lisp objects in Lisp notation. Do this for each variable which is a Lisp object, in several stack frames near the bottom of the stack. Look at the source to see which variables are Lisp objects, because the debugger thinks of them as integers.

To show a variable's value in Lisp syntax, first print its value, then use the user-defined GDB command `pr` to print the Lisp object in Lisp syntax. (If you must use another debugger, call the function `debug_print` with the object as an argument.) The `pr` command is defined by the file '`.gdbinit`', and it works only if you are debugging a running process (not with a core dump).

To make Lisp errors stop Emacs and return to GDB, put a breakpoint at `Fsignal`.

For a short listing of Lisp functions running, type the GDB command `xbacktrace`.

The file '`.gdbinit`' defines several other commands that are useful for examining the data types and contents of Lisp objects. Their names begin with '`x`'. These commands work at a lower level than `pr`, and are less convenient, but they may work even when `pr` does not, such as when debugging a core dump or when Emacs has had a fatal signal.

More detailed advice and other useful techniques for debugging Emacs are available in the file '`etc/DEBUG`' in the Emacs distribution. That file also includes instructions for investigating problems whereby Emacs stops responding (many people assume that Emacs is "hung," whereas in fact it might be in an infinite loop).

In an installed Emacs, the file '`etc/DEBUG`' is in the same directory where the Emacs on-line documentation file '`DOC`', typically in the '`/usr/local/share/emacs/`*version*`/etc/`' directory. The directory for your installation is stored in the variable `data-directory`.

Here are some things that are not necessary in a bug report:

- A description of the envelope of the bug—this is not necessary for a reproducible bug.

  Often people who encounter a bug spend a lot of time investigating which changes to the input file will make the bug go away and which changes will not affect it.

  This is often time-consuming and not very useful, because the way we will find the bug is by running a single example under the debugger with breakpoints, not by pure deduction from a series of examples. You might as well save time by not searching for additional examples.

  Of course, if you can find a simpler example to report *instead* of the original one, that is a convenience. Errors in the output will be easier to spot, running under the debugger will take less time, etc.

  However, simplification is not vital; if you can't do this or don't have time to try, please report the bug with your original test case.

- A system-call trace of Emacs execution.

  System-call traces are very useful for certain special kinds of debugging, but in most cases they give little useful information. It is therefore strange that many people seem to think that *the* way to report information about a crash is to send a system-call trace. Perhaps this is a habit formed from experience debugging programs that don't have source code or debugging symbols.

  In most programs, a backtrace is normally far, far more informative than a system-call trace. Even in Emacs, a simple backtrace is generally more informative, though to give full information you should supplement the backtrace by displaying variable values and printing them as Lisp objects with `pr` (see above).

- A patch for the bug.

  A patch for the bug is useful if it is a good one. But don't omit the other information that a bug report needs, such as the test case, on the assumption that a patch is sufficient. We might see problems with your patch and decide to fix the problem another way, or we might not understand it at all. And if we can't understand what bug you are trying to fix, or why your patch should be an improvement, we mustn't install it.

- A guess about what the bug is or what it depends on.

  Such guesses are usually wrong. Even experts can't guess right about such things without first using the debugger to find the facts.

32.3.4 Sending Patches for GNU Emacs

If you would like to write bug fixes or improvements for GNU Emacs, that is very helpful. When you send your changes, please follow these guidelines to make it easy for the maintainers to use them. If you don't follow these guidelines, your information might still be useful, but using it will take extra work. Maintaining GNU Emacs is a lot of work in the best of circumstances, and we can't keep up unless you do your best to help.

- Send an explanation with your changes of what problem they fix or what improvement they bring about. For a bug fix, just include a copy of the bug report, and explain why the change fixes the bug.

  (Referring to a bug report is not as good as including it, because then we will have to look it up, and we have probably already deleted it if we've already fixed the bug.)

- Always include a proper bug report for the problem you think you have fixed. We need to convince ourselves that the change is right before installing it. Even if it is correct, we might have trouble understanding it if we don't have a way to reproduce the problem.

- Include all the comments that are appropriate to help people reading the source in the future understand why this change was needed.
- Don't mix together changes made for different reasons. Send them *individually*.

  If you make two changes for separate reasons, then we might not want to install them both. We might want to install just one. If you send them all jumbled together in a single set of diffs, we have to do extra work to disentangle them—to figure out which parts of the change serve which purpose. If we don't have time for this, we might have to ignore your changes entirely.

  If you send each change as soon as you have written it, with its own explanation, then two changes never get tangled up, and we can consider each one properly without any extra work to disentangle them.
- Send each change as soon as that change is finished. Sometimes people think they are helping us by accumulating many changes to send them all together. As explained above, this is absolutely the worst thing you could do.

  Since you should send each change separately, you might as well send it right away. That gives us the option of installing it immediately if it is important.
- Use '`diff -c`' to make your diffs. Diffs without context are hard to install reliably. More than that, they are hard to study; we must always study a patch to decide whether we want to install it. Unidiff format is better than contextless diffs, but not as easy to read as '`-c`' format.

  If you have GNU diff, use '`diff -c -F'^[_a-zA-Z0-9$]+ *('`' when making diffs of C code. This shows the name of the function that each change occurs in.
- Avoid any ambiguity as to which is the old version and which is the new. Please make the old version the first argument to diff, and the new version the second argument. And please give one version or the other a name that indicates whether it is the old version or your new changed one.
- Write the change log entries for your changes. This is both to save us the extra work of writing them, and to help explain your changes so we can understand them.

  The purpose of the change log is to show people where to find what was changed. So you need to be specific about what functions you changed; in large functions, it's often helpful to indicate where within the function the change was.

  On the other hand, once you have shown people where to find the change, you need not explain its purpose in the change log. Thus, if you add a new function, all you need to say about it is that it is new. If you feel that the purpose needs explaining, it probably does—but put the explanation in comments in the code. It will be more useful there.

Please read the '`ChangeLog`' files in the '`src`' and '`lisp`' directories to see what sorts of information to put in, and to learn the style that we use. If you would like your name to appear in the header line, showing who made the change, send us the header line. See Section 22.14 [Change Log], page 299.

- When you write the fix, keep in mind that we can't install a change that would break other systems. Please think about what effect your change will have if compiled on another type of system.

  Sometimes people send fixes that *might* be an improvement in general—but it is hard to be sure of this. It's hard to install such changes because we have to study them very carefully. Of course, a good explanation of the reasoning by which you concluded the change was correct can help convince us.

  The safest changes are changes to the configuration files for a particular machine. These are safe because they can't create new bugs on other machines.

  Please help us keep up with the workload by designing the patch in a form that is clearly safe to install.

32.4 Contributing to Emacs Development

If you would like to help pretest Emacs releases to assure they work well, or if you would like to work on improving Emacs, please contact the maintainers at `bug-gnu-emacs@gnu.org`. A pretester should be prepared to investigate bugs as well as report them. If you'd like to work on improving Emacs, please ask for suggested projects or suggest your own ideas.

If you have already written an improvement, please tell us about it. If you have not yet started work, it is useful to contact `bug-gnu-emacs@gnu.org` before you start; it might be possible to suggest ways to make your extension fit in better with the rest of Emacs.

32.5 How To Get Help with GNU Emacs

If you need help installing, using or changing GNU Emacs, there are two ways to find it:

- Send a message to the mailing list `help-gnu-emacs@gnu.org`, or post your request on newsgroup `gnu.emacs.help`. (This mailing list and newsgroup interconnect, so it does not matter which one you use.)
- Look in the service directory for someone who might help you for a fee. The service directory is found in the file named '`etc/SERVICE`' in the Emacs distribution.

Appendix B Command Line Arguments

GNU Emacs supports command line arguments to request various actions when invoking Emacs. These are for compatibility with other editors and for sophisticated activities. We don't recommend using them for ordinary editing.

Arguments starting with '`-`' are *options*. Other arguments specify files to visit. Emacs visits the specified files while it starts up. The last file name on your command line becomes the current buffer; the other files are also present in other buffers. As usual, the special argument '`--`' says that all subsequent arguments are file names, not options, even if they start with '`-`'.

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

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

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

Most options specify how to initialize Emacs, or set parameters for the Emacs session. We call them *initial options*. A few options specify things to do: for example, load libraries, call functions, or exit Emacs. These are called *action options*. These and file names together are called *action arguments*. Emacs processes all the action arguments in the order they are written.

B.1 Action Arguments

Here is a table of the action arguments and options:

'*file*'
'`--visit=`*file*'
'`--file=`*file*'
: Visit *file* using `find-file`. See Section 14.2 [Visiting], page 145.

‘+*linenum file*’
: Visit *file* using `find-file`, then go to line number *linenum* in it.

‘`-l` *file*’
‘`--load=`*file*’
: Load a Lisp library named *file* with the function `load`. See Section 23.7 [Lisp Libraries], page 339. The library can be found either in the current directory, or in the Emacs library search path as specified with `EMACSLOADPATH` (see Section B.5.1 [General Variables], page 511).

‘`-f` *function*’
‘`--funcall=`*function*’
: Call Lisp function *function* with no arguments.

‘`--eval=`*expression*’
‘`--execute=`*expression*’
: Evaluate Lisp expression *expression*.

‘`--insert=`*file*’
: Insert the contents of *file* into the current buffer. This is like what `M-x insert-file` does. See Section 14.10 [Misc File Ops], page 181.

‘`--kill`’ Exit from Emacs without asking for confirmation.

The init file can access the values of the action arguments as the elements of a list in the variable `command-line-args`. The init file can override the normal processing of the action arguments, or define new ones, by reading and setting this variable.

B.2 Initial Options

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

Some initial options affect the loading of init files. The normal actions of Emacs are to first load ‘`site-start.el`’ if it exists, then your own init file ‘`~/.emacs`’ if it exists, and finally ‘`default.el`’ if it exists; certain options prevent loading of some of these files or substitute other files for them.

‘`-t` *device*’
‘`--terminal=`*device*’
: Use *device* as the device for terminal input and output.

‘`-d` *display*’
‘`--display=`*display*’
: Use the X Window System and use the display named *display* to open the initial Emacs frame.

‘`-nw`’
‘`--no-windows`’
: Don’t communicate directly with the window system, disregarding the `DISPLAY` environment variable even if it is set. This forces Emacs to run as if the display were a character terminal.

‘`-batch`’
‘`--batch`’
: Run Emacs in *batch mode*, which means that the text being edited is not displayed and the standard terminal interrupt characters such as `C-z` and `C-c` continue to have their normal effect. Emacs in batch mode outputs to `stderr` only what would normally be printed in the echo area under program control.

  Batch mode is used for running programs written in Emacs Lisp from shell scripts, makefiles, and so on. Normally the ‘`-l`’ option or ‘`-f`’ option will be used as well, to invoke a Lisp program to do the batch processing.

  ‘`-batch`’ implies ‘`-q`’ (do not load an init file). It also causes Emacs to kill itself after all command options have been processed. In addition, auto-saving is not done except in buffers for which it has been explicitly requested.

‘`-q`’
‘`--no-init-file`’
: Do not load your Emacs init file ‘`~/.emacs`’, or ‘`default.el`’ either.

‘`--no-site-file`’
: Do not load ‘`site-start.el`’. The options ‘`-q`’, ‘`-u`’ and ‘`-batch`’ have no effect on the loading of this file—this is the only option that blocks it.

‘`-u` *user*’
‘`--user=`*user*’
: Load *user*’s Emacs init file ‘`~`*user*`/.emacs`’ instead of your own.

‘`--debug-init`’
: Enable the Emacs Lisp debugger for errors in the init file.

‘`--unibyte`’
: Set up to do almost everything with single-byte buffers and strings. All buffers and strings are unibyte unless you (or a Lisp program) explicitly ask for a multibyte buffer or string. (Note that Emacs always loads Lisp files in multibyte mode, even if ‘`--unibyte`’ is specified; see Section 18.2 [Enabling Multibyte],

page 219.) Setting the environment variable `EMACS_UNIBYTE` has the same effect.

‘`--multibyte`’
: Inhibit the effect of `EMACS_UNIBYTE`, so that Emacs uses multibyte characters by default, as usual.

B.3 Command Argument Example

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

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

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

B.4 Resuming Emacs with Arguments

You can specify action arguments for Emacs when you resume it after a suspension. To prepare for this, put the following code in your ‘`.emacs`’ file (see Section 31.2.3 [Hooks], page 465):

```
(add-hook 'suspend-hook 'resume-suspend-hook)
(add-hook 'suspend-resume-hook 'resume-process-args)
```

As further preparation, you must execute the shell script ‘`emacs.csh`’ (if you use csh as your shell) or ‘`emacs.bash`’ (if you use bash as your shell). These scripts define an alias named `edit`, which will resume Emacs giving it new command line arguments such as files to visit.

Only action arguments work properly when you resume Emacs. Initial arguments are not recognized—it’s too late to execute them anyway.

Note that resuming Emacs (with or without arguments) must be done from within the shell that is the parent of the Emacs job. This is why `edit` is an alias rather than a program or a shell script. It is not possible to implement a resumption command that could be run from other subjobs of the shell; no way to define a command that could be made the value of `EDITOR`, for example. Therefore, this feature does not take the place of the Emacs Server feature (see Section 30.3 [Emacs Server], page 436).

The aliases use the Emacs Server feature if you appear to have a server Emacs running. However, they cannot determine this with complete accu-

racy. They may think that a server is still running when in actuality you have killed that Emacs, because the file '`/tmp/.esrv...`' still exists. If this happens, find that file and delete it.

B.5 Environment Variables

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

What makes the environment useful is that subprocesses inherit the environment automatically from their parent process. This means you can set up an environment variable in your login shell, and all the programs you run (including Emacs) will automatically see it. Subprocesses of Emacs (such as shells, compilers, and version-control software) inherit the environment from Emacs, too.

Inside Emacs, the command `M-x getenv` gets the value of an environment variable. `M-x setenv` sets a variable in the Emacs environment. The way to set environment variables outside of Emacs depends on the operating system, and especially the shell that you are using. For example, here's how to set the environment variable `ORGANIZATION` to '`not very much`' using Bash:

```
export ORGANIZATION="not very much"
```

and here's how to do it in csh or tcsh:

```
setenv ORGANIZATION "not very much"
```

When Emacs is set-up to use the X Window System, it inherits the use of a large number of environment variables from the X libraries. See the X documentation for more information.

B.5.1 General Variables

Here is an alphabetical list of specific environment variables that have special meanings in Emacs, giving the name of each variable and its meaning. Most of these variables are also used by some other programs. Emacs does not require any of these environment variables to be set, but it uses their values if they are set.

`CDPATH` Used by the `cd` command to search for the directory you specify, when you specify a relative directory name.

`DOMAINNAME`
The name of the Internet domain that the machine running Emacs is located in. Used by the Gnus package.

`EMACS_UNIBYTE`
: Defining this environment variable with a nonempty value directs Emacs to do almost everything with single-byte buffers and strings. It is equivalent to using the '`--unibyte`' command-line option on each invocation. See Section B.2 [Initial Options], page 508.

`EMACSDATA`
: Directory for the architecture-independent files that come with Emacs. This is used to initialize the Lisp variable `data-directory`.

`EMACSDOC`
: Directory for the documentation string file, '`DOC-`*emacsversion*'. This is used to initialize the Lisp variable `doc-directory`.

`EMACSLOADPATH`
: A colon-separated list of directories[1] to search for Emacs Lisp files—used to initialize `load-path`.

`EMACSPATH`
: A colon-separated list of directories to search for executable files—used to initialize `exec-path`.

`ESHELL`
: Used for shell-mode to override the `SHELL` environment variable.

`HISTFILE`
: The name of the file that shell commands are saved in between logins. This variable defaults to '`~/.bash_history`' if you use Bash, to '`~/.sh_history`' if you use ksh, and to '`~/.history`' otherwise.

`HOME`
: The location of the user's files in the directory tree; used for expansion of file names starting with a tilde ('`~`'). On MS-DOS, it defaults to the directory from which Emacs was started, with '`/bin`' removed from the end if it was present. On Windows, the default value of `HOME` is '`C:/`', the root directory of drive '`C:`'.

`HOSTNAME`
: The name of the machine that Emacs is running on.

`INCPATH`
: A colon-separated list of directories. Used by the `complete` package to search for files.

`INFOPATH`
: A colon-separated list of directories in which to search for Info files.

`LC_ALL`
`LC_CTYPE`
`LANG`
: The user's preferred locale. (The first of these environment variables with a nonempty value specifies the locale.) A locale name

[1] Here and below, whenever we say "colon-separated list of directories", it pertains to Unix and GNU/Linux systems. On MS-DOS and MS-Windows, the directories are separated by semi-colons instead, since DOS/Windows file names might include a colon after a drive letter.

which contains '`8859-`*n*', '`8859_`*n*' or '`8859`*n*', where *n* is between 1 and 4, automatically specifies the '`Latin-`*n*' language environment when Emacs starts up. There are a few extensions: if *n* is 9, that specifies '`Latin-5`', and if *n* is 14 or 15, that specifies '`Latin-8`' and '`Latin-9`', respectively.

The locale value you specify with one of these three variables is matched against entries in `locale-language-names`, `locale-charset-language-names`, and `locale-preferred-coding-systems`, to select a default language environment and coding system. See Section 18.3 [Language Environments], page 220.

`LOGNAME` The user's login name. See also `USER`.

`MAIL` The name of the user's system mail inbox.

`MAILRC` Name of file containing mail aliases. (The default is '`~/.mailrc`'.)

`MH` Name of setup file for the mh system. (The default is '`~/.mh_profile`'.)

`NAME` The real-world name of the user.

`NNTPSERVER`
The name of the news server. Used by the mh and Gnus packages.

`ORGANIZATION`
The name of the organization to which you belong. Used for setting the 'Organization:' header in your posts from the Gnus package.

`PATH` A colon-separated list of directories in which executables reside. This is used to initialize the Emacs Lisp variable `exec-path`.

`PWD` If set, this should be the default directory when Emacs was started.

`REPLYTO` If set, this specifies an initial value for the variable `mail-default-reply-to`. See Section 26.2 [Mail Headers], page 358.

`SAVEDIR` The name of a directory in which news articles are saved by default. Used by the Gnus package.

`SHELL` The name of an interpreter used to parse and execute programs run from inside Emacs.

`TERM` The name of the terminal that Emacs is running on. The variable must be set unless Emacs is run in batch mode. On MS-DOS, it defaults to '`internal`', which specifies a built-in terminal emulation that handles the machine's own display. If the value of `TERM` indicates that Emacs runs in non-windowed mode from `xterm` or a similar terminal emulator, the background mode defaults to '`light`', and Emacs will choose colors that are appropriate for a light background.

`TERMCAP` The name of the termcap library file describing how to program the terminal specified by the `TERM` variable. This defaults to '`/etc/termcap`'.

`TMPDIR` Used by the Emerge package as a prefix for temporary files.

`TZ` This specifies the current time zone and possibly also daylight saving time information. On MS-DOS, if `TZ` is not set in the environment when Emacs starts, Emacs defines a default value as appropriate for the country code returned by DOS. On MS-Windows, Emacs does not use `TZ` at all.

`USER` The user's login name. See also `LOGNAME`. On MS-DOS, this defaults to '`root`'.

`VERSION_CONTROL`
Used to initialize the `version-control` variable (see Section 14.3.1.1 [Backup Names], page 151).

B.5.2 Miscellaneous Variables

These variables are used only on particular configurations:

`COMSPEC` On MS-DOS and MS-Windows, the name of the command interpreter to use when invoking batch files and commands internal to the shell. On MS-DOS this is also used to make a default value for the `SHELL` environment variable.

`NAME` On MS-DOS, this variable defaults to the value of the `USER` variable.

`TEMP`
`TMP` On MS-DOS and MS-Windows, these specify the name of the directory for storing temporary files in.

`EMACSTEST`
On MS-DOS, this specifies a file to use to log the operation of the internal terminal emulator. This feature is useful for submitting bug reports.

`EMACSCOLORS`
On MS-DOS, this specifies the screen colors. It is useful to set them this way, since otherwise Emacs would display the default colors momentarily when it starts up.

The value of this variable should be the two-character encoding of the foreground (the first character) and the background (the second character) colors of the default face. Each character should be the hexadecimal code for the desired color on a standard PC text-mode display. For example, to get blue text on a

lightgray backgraound, specify '`EMACSCOLORS=17`', since 1 is the code of the blue color and 7 is the code of the lightgray color.

The PC display usually supports only eight background colors. However, Emacs switches the DOS display to a mode where all 16 colors can be used for the background, so all four bits of the background color are actually used.

`WINDOW_GFX`
: Used when initializing the Sun windows system.

B.6 Specifying the Display Name

The environment variable `DISPLAY` tells all X clients, including Emacs, where to display their windows. Its value is set up by default in ordinary circumstances, when you start an X server and run jobs locally. Occasionally you may need to specify the display yourself; for example, if you do a remote login and want to run a client program remotely, displaying on your local screen.

With Emacs, the main reason people change the default display is to let them log into another system, run Emacs on that system, but have the window displayed at their local terminal. You might need to log in to another system because the files you want to edit are there, or because the Emacs executable file you want to run is there.

The syntax of the `DISPLAY` environment variable is '*host*`:`*display*`.`*screen*', where *host* is the host name of the X Window System server machine, *display* is an arbitrarily-assigned number that distinguishes your server (X terminal) from other servers on the same machine, and *screen* is a rarely-used field that allows an X server to control multiple terminal screens. The period and the *screen* field are optional. If included, *screen* is usually zero.

For example, if your host is named '`glasperle`' and your server is the first (or perhaps the only) server listed in the configuration, your `DISPLAY` is '`glasperle:0.0`'.

You can specify the display name explicitly when you run Emacs, either by changing the `DISPLAY` variable, or with the option '`-d` *display*' or '`--display=`*display*'. Here is an example:

```
emacs --display=glasperle:0 &
```

You can inhibit the direct use of the window system and GUI with the '`-nw`' option. It tells Emacs to display using ordinary ASCII on its controlling terminal. This is also an initial option.

Sometimes, security arrangements prevent a program on a remote system from displaying on your local system. In this case, trying to run Emacs produces messages like this:

```
Xlib:  connection to "glasperle:0.0" refused by server
```

You might be able to overcome this problem by using the `xhost` command on the local system to give permission for access from your remote machine.

B.7 Font Specification Options

By default, Emacs displays text in the font named '`9x15`', which makes each character nine pixels wide and fifteen pixels high. You can specify a different font on your command line through the option '`-fn` *name*' (or '`--font`', which is an alias for '`-fn`').

'`-fn` *name*'
'`--font=`*name*'
: Use font *name* as the default font.

Under X, each font has a long name which consists of eleven words or numbers, separated by dashes. Some fonts also have shorter nicknames—'`9x15`' is such a nickname. You can use either kind of name. You can use wildcard patterns for the font name; then Emacs lets X choose one of the fonts that match the pattern. Here is an example, which happens to specify the font whose nickname is '`6x13`':

```
emacs -fn "-misc-fixed-medium-r-semicondensed--13-*-*-*-c-60-iso8859-1" &
```

You can also specify the font in your '`.Xdefaults`' file:

```
emacs.font: -misc-fixed-medium-r-semicondensed--13-*-*-*-c-60-iso8859-1
```

A long font name has the following form:

-maker-family-weight-slant-widthtype-style...
...-pixels-height-horiz-vert-spacing-width-charset

maker This is the name of the font manufacturer.

family This is the name of the font family—for example, '`courier`'.

weight This is normally '`bold`', '`medium`' or '`light`'. Other words may appear here in some font names.

slant This is '`r`' (roman), '`i`' (italic), '`o`' (oblique), '`ri`' (reverse italic), or '`ot`' (other).

widthtype This is normally '`condensed`', '`extended`', '`semicondensed`' or '`normal`'. Other words may appear here in some font names.

style This is an optional additional style name. Usually it is empty—most long font names have two hyphens in a row at this point.

pixels This is the font height, in pixels.

height This is the font height on the screen, measured in tenths of a printer's point—approximately 1/720 of an inch. In other words, it is the point size of the font, times ten. For a given vertical resolution, *height* and *pixels* are proportional; therefore, it is common to specify just one of them and use '`*`' for the other.

horiz This is the horizontal resolution, in pixels per inch, of the screen for which the font is intended.

vert This is the vertical resolution, in pixels per inch, of the screen for which the font is intended. Normally the resolution of the fonts on your system is the right value for your screen; therefore, you normally specify '`*`' for this and *horiz*.

spacing This is '`m`' (monospace), '`p`' (proportional) or '`c`' (character cell).

width This is the average character width, in pixels, multiplied by ten.

charset This is the character set that the font depicts. Normally you should use '`iso8859-1`'.

You will probably want to use a fixed-width default font—that is, a font in which all characters have the same width. Any font with '`m`' or '`c`' in the *spacing* field of the long name is a fixed-width font. Here's how to use the `xlsfonts` program to list all the fixed-width fonts available on your system:

```
xlsfonts -fn '*x*' | egrep "^[0-9]+x[0-9]+"
xlsfonts -fn '*-*-*-*-*-*-*-*-*-*-*-m*'
xlsfonts -fn '*-*-*-*-*-*-*-*-*-*-*-c*'
```

To see what a particular font looks like, use the `xfd` command. For example:

```
xfd -fn 6x13
```

displays the entire font '`6x13`'.

While running Emacs, you can set the font of the current frame (see Section 17.12 [Frame Parameters], page 213) or for a specific kind of text (see Section 11.1 [Faces], page 103).

B.8 Window Color Options

On a color display, you can specify which color to use for various parts of the Emacs display. To find out what colors are available on your system, type `M-x list-colors-display`, or press `C-Mouse-2` and select '`Display Colors`' from the pop-up menu. If you do not specify colors, on windowed displays the default for the background is white and the default for all other colors is black. On a monochrome display, the foreground is black, the background is white, and the border is gray if the display supports that. On terminals, the background is usually black and the foreground is white.

Here is a list of the command-line options for specifying colors:

'`-fg` *color*'
'`--foreground-color=`*color*'
: Specify the foreground color. *color* should be a standard color name, or a numeric specification of the color's red, green, and blue components as in '`#4682B4`' or '`RGB:46/82/B4`'.

‘`-bg` *color*’
‘`--background-color=`*color*’
: Specify the background color.

‘`-bd` *color*’
‘`--border-color=`*color*’
: Specify the color of the border of the X window.

‘`-cr` *color*’
‘`--cursor-color=`*color*’
: Specify the color of the Emacs cursor which indicates where point is.

‘`-ms` *color*’
‘`--mouse-color=`*color*’
: Specify the color for the mouse cursor when the mouse is in the Emacs window.

‘`-r`’
‘`--reverse-video`’
: Reverse video—swap the foreground and background colors.

For example, to use a coral mouse cursor and a slate blue text cursor, enter:

```
emacs -ms coral -cr 'slate blue' &
```

You can reverse the foreground and background colors through the ‘`-r`’ option or with the X resource ‘`reverseVideo`’.

The ‘`-fg`’, ‘`-bg`’, and ‘`-rv`’ options function on character terminals as well as on window systems.

B.9 Options for Window Geometry

The ‘`-geometry`’ option controls the size and position of the initial Emacs frame. Here is the format for specifying the window geometry:

‘`-g` *width*x*height*`{+-}`*xoffset*`{+-}`*yoffset*’
: Specify window size *width* and *height* (measured in character columns and lines), and positions *xoffset* and *yoffset* (measured in pixels).

‘`--geometry=`*width*x*height*`{+-}`*xoffset*`{+-}`*yoffset*’
: This is another way of writing the same thing.

`{+-}` means either a plus sign or a minus sign. A plus sign before *xoffset* means it is the distance from the left side of the screen; a minus sign means it counts from the right side. A plus sign before *yoffset* means it is the distance from the top of the screen, and a minus sign there indicates the distance from the bottom. The values *xoffset* and *yoffset* may themselves

be positive or negative, but that doesn't change their meaning, only their direction.

Emacs uses the same units as `xterm` does to interpret the geometry. The *width* and *height* are measured in characters, so a large font creates a larger frame than a small font. (If you specify a proportional font, Emacs uses its maximum bounds width as the width unit.) The *xoffset* and *yoffset* are measured in pixels.

Since the mode line and the echo area occupy the last 2 lines of the frame, the height of the initial text window is 2 less than the height specified in your geometry. In non-X-toolkit versions of Emacs, the menu bar also takes one line of the specified number. But in the X toolkit version, the menu bar is additional and does not count against the specified height. The tool bar, if present, is also additional.

You do not have to specify all of the fields in the geometry specification.

If you omit both *xoffset* and *yoffset*, the window manager decides where to put the Emacs frame, possibly by letting you place it with the mouse. For example, '`164x55`' specifies a window 164 columns wide, enough for two ordinary width windows side by side, and 55 lines tall.

The default width for Emacs is 80 characters and the default height is 40 lines. You can omit either the width or the height or both. If you start the geometry with an integer, Emacs interprets it as the width. If you start with an '`x`' followed by an integer, Emacs interprets it as the height. Thus, '`81`' specifies just the width; '`x45`' specifies just the height.

If you start with '`+`' or '`-`', that introduces an offset, which means both sizes are omitted. Thus, '`-3`' specifies the *xoffset* only. (If you give just one offset, it is always *xoffset*.) '`+3-3`' specifies both the *xoffset* and the *yoffset*, placing the frame near the bottom left of the screen.

You can specify a default for any or all of the fields in '`.Xdefaults`' file, and then override selected fields with a '`--geometry`' option.

B.10 Internal and External Borders

An Emacs frame has an internal border and an external border. The internal border is an extra strip of the background color around all four edges of the frame. Emacs itself adds the internal border. The external border is added by the window manager outside the internal border; it may contain various boxes you can click on to move or iconify the window.

'`-ib` *width*'
'`--internal-border=`*width*'
: Specify *width* as the width of the internal border, in pixels.

'`-bw` *width*'
'`--border-width=`*width*'
: Specify *width* as the width of the main border, in pixels.

When you specify the size of the frame, that does not count the borders. The frame's position is measured from the outside edge of the external border.

Use the '`-ib` *n*' option to specify an internal border *n* pixels wide. The default is 1. Use '`-bw` *n*' to specify the width of the external border (though the window manager may not pay attention to what you specify). The default width of the external border is 2.

B.11 Frame Titles

An Emacs frame may or may not have a specified title. The frame title, if specified, appears in window decorations and icons as the name of the frame. If an Emacs frame has no specified title, the default title is the name of the executable program (if there is only one frame) or the selected window's buffer name (if there is more than one frame).

You can specify a title for the initial Emacs frame with a command line option:

'`-title` *title*'
'`--title=`*title*'
'`-T` *title*' Specify *title* as the title for the initial Emacs frame.

The '`--name`' option (see Section B.13 [Resources X], page 521) also specifies the title for the initial Emacs frame.

B.12 Icons

Most window managers allow the user to "iconify" a frame, removing it from sight, and leaving a small, distinctive "icon" window in its place. Clicking on the icon window makes the frame itself appear again. If you have many clients running at once, you can avoid cluttering up the screen by iconifying most of the clients.

'`-i`'
'`--icon-type`'
Use a picture of a gnu as the Emacs icon.

'`-iconic`'
'`--iconic`'
Start Emacs in iconified state.

The '`-i`' or '`--icon-type`' option tells Emacs to use an icon window containing a picture of the GNU gnu. If omitted, Emacs lets the window manager choose what sort of icon to use—usually just a small rectangle containing the frame's title.

The '`-iconic`' option tells Emacs to begin running as an icon, rather than showing a frame right away. In this situation, the icon is the only indication that Emacs has started; the text frame doesn't appear until you deiconify it.

B.13 X Resources

Programs running under the X Window System organize their user options under a hierarchy of classes and resources. You can specify default values for these options in your X resources file, usually named '`~/.Xdefaults`'.

Each line in the file specifies a value for one option or for a collection of related options, for one program or for several programs (optionally even for all programs).

Programs define named resources with particular meanings. They also define how to group resources into named classes. For instance, in Emacs, the '`internalBorder`' resource controls the width of the internal border, and the '`borderWidth`' resource controls the width of the external border. Both of these resources are part of the '`BorderWidth`' class. Case distinctions are significant in these names.

In '`~/.Xdefaults`', you can specify a value for a single resource on one line, like this:

```
emacs.borderWidth: 2
```

Or you can use a class name to specify the same value for all resources in that class. Here's an example:

```
emacs.BorderWidth: 2
```

If you specify a value for a class, it becomes the default for all resources in that class. You can specify values for individual resources as well; these override the class value, for those particular resources. Thus, this example specifies 2 as the default width for all borders, but overrides this value with 4 for the external border:

```
emacs.Borderwidth: 2
emacs.borderwidth: 4
```

The order in which the lines appear in the file does not matter. Also, command-line options always override the X resources file.

The string '`emacs`' in the examples above is also a resource name. It actually represents the name of the executable file that you invoke to run Emacs. If Emacs is installed under a different name, it looks for resources under that name instead of '`emacs`'.

'`-name` *name*'
'`--name=`*name*'
: Use *name* as the resource name (and the title) for the initial Emacs frame. This option does not affect subsequent frames,

but Lisp programs can specify frame names when they create frames.

If you don't specify this option, the default is to use the Emacs executable's name as the resource name.

'`-xrm` *resource-values*'
'`--xrm=`*resource-values*'
Specify X resource values for this Emacs job (see below).

For consistency, '`-name`' also specifies the name to use for other resource values that do not belong to any particular frame.

The resources that name Emacs invocations also belong to a class; its name is '`Emacs`'. If you write '`Emacs`' instead of '`emacs`', the resource applies to all frames in all Emacs jobs, regardless of frame titles and regardless of the name of the executable file. Here is an example:

```
Emacs.BorderWidth: 2
Emacs.borderWidth: 4
```

You can specify a string of additional resource values for Emacs to use with the command line option '`-xrm` *resources*'. The text *resources* should have the same format that you would use inside a file of X resources. To include multiple resource specifications in *resources*, put a newline between them, just as you would in a file. You can also use '`#include "`*filename*`"`' to include a file full of resource specifications. Resource values specified with '`-xrm`' take precedence over all other resource specifications.

The following table lists the resource names that designate options for Emacs, each with the class that it belongs to:

`background` (class `Background`)
Background color name.

`bitmapIcon` (class `BitmapIcon`)
Use a bitmap icon (a picture of a gnu) if '`on`', let the window manager choose an icon if '`off`'.

`borderColor` (class `BorderColor`)
Color name for the external border.

`borderWidth` (class `BorderWidth`)
Width in pixels of the external border.

`cursorColor` (class `Foreground`)
Color name for text cursor (point).

`font` (class `Font`)
Font name for text (or fontset name, see Section 18.9 [Fontsets], page 229).

`foreground` (class `Foreground`)
Color name for text.

`geometry` (class `Geometry`)
: Window size and position. Be careful not to specify this resource as '`emacs*geometry`', because that may affect individual menus as well as the Emacs frame itself.

  If this resource specifies a position, that position applies only to the initial Emacs frame (or, in the case of a resource for a specific frame name, only that frame). However, the size if specified here applies to all frames.

`iconName` (class `Title`)
: Name to display in the icon.

`internalBorder` (class `BorderWidth`)
: Width in pixels of the internal border.

`lineSpacing` (class LineSpacing)
: Additional space (*leading*) between lines, in pixels.

`menuBar` (class `MenuBar`)
: Give frames menu bars if '`on`'; don't have menu bars if '`off`'.

`toolBar` (class `ToolBar`)
: Number of lines to reserve for the tool bar. A zero value suppresses the tool bar. If the value is non-zero and `auto-resize-tool-bars` is non-`nil`, the tool bar's size will be changed automatically so that all tool bar items are visible.

`minibuffer` (class `Minibuffer`)
: If '`none`', don't make a minibuffer in this frame. It will use a separate minibuffer frame instead.

`paneFont` (class `Font`)
: Font name for menu pane titles, in non-toolkit versions of Emacs.

`pointerColor` (class `Foreground`)
: Color of the mouse cursor.

`reverseVideo` (class `ReverseVideo`)
: Switch foreground and background default colors if '`on`', use colors as specified if '`off`'.

`screenGamma` (class `ScreenGamma`)
: Gamma correction for colors, equivalent to the frame parameter `screen-gamma`.

`selectionFont` (class `Font`)
: Font name for pop-up menu items, in non-toolkit versions of Emacs. (For toolkit versions, see Section B.14 [Lucid Resources], page 524, also see Section B.15 [LessTif Resources], page 525.)

`synchronous` (class `Synchronous`)
: Run Emacs in synchronous mode if '`on`'. Synchronous mode is useful for debugging X problems.

`title` (class `Title`)
: Name to display in the title bar of the initial Emacs frame.

`verticalScrollBars` (class `ScrollBars`)
: Give frames scroll bars if '`on`'; don't have scroll bars if '`off`'.

Here are resources for controlling the appearance of particular faces (see Section 11.1 [Faces], page 103):

face`.attributeFont`
: Font for face *face*.

face`.attributeForeground`
: Foreground color for face *face*.

face`.attributeBackground`
: Background color for face *face*.

face`.attributeUnderline`
: Underline flag for face *face*. Use '`on`' or '`true`' for yes.

B.14 Lucid Menu X Resources

If the Emacs installed at your site was built to use the X toolkit with the Lucid menu widgets, then the menu bar is a separate widget and has its own resources. The resource names contain '`pane.menubar`' (following, as always, the name of the Emacs invocation or '`Emacs`' which stands for all Emacs invocations). Specify them like this:

```
Emacs.pane.menubar.resource:  value
```

For example, to specify the font '`8x16`' for the menu-bar items, write this:

```
Emacs.pane.menubar.font:  8x16
```

Resources for *non-menubar* toolkit pop-up menus have '`menu*`', in like fashion. For example, to specify the font '`8x16`' for the pop-up menu items, write this:

```
Emacs.menu*.font:  8x16
```

For dialog boxes, use '`dialog`' instead of '`menu`':

```
Emacs.dialog*.font:  8x16
```

Experience shows that on some systems you may need to add '`shell.`' before the '`pane.menubar`' or '`menu*`'. On some other systems, you must not add '`shell.`'.

Here is a list of the specific resources for menu bars and pop-up menus:

`font`
: Font for menu item text.

`foreground`
: Color of the foreground.

`background`
: Color of the background.

`buttonForeground`
: In the menu bar, the color of the foreground for a selected item.

`horizontalSpacing`
: Horizontal spacing in pixels between items. Default is 3.

`verticalSpacing`
: Vertical spacing in pixels between items. Default is 1.

`arrowSpacing`
: Horizontal spacing between the arrow (which indicates a submenu) and the associated text. Default is 10.

`shadowThickness`
: Thickness of shadow line around the widget.

`margin`
: The margin of the menu bar, in characters. The default of 4 makes the menu bar appear like the LessTif/Motif one.

B.15 LessTif Menu X Resources

If the Emacs installed at your site was built to use the X toolkit with the LessTif or Motif widgets, then the menu bar is a separate widget and has its own resources. The resource names contain '`pane.menubar`' (following, as always, the name of the Emacs invocation or '`Emacs`' which stands for all Emacs invocations). Specify them like this:

`Emacs.pane.menubar.`*subwidget*`.`*resource*`:` *value*

Each individual string in the menu bar is a subwidget; the subwidget's name is the same as the menu item string. For example, the word '`File`' in the menu bar is part of a subwidget named '`emacs.pane.menubar.File`'. Most likely, you want to specify the same resources for the whole menu bar. To do this, use '`*`' instead of a specific subwidget name. For example, to specify the font '`8x16`' for the menu-bar items, write this:

```
Emacs.pane.menubar.*.fontList:  8x16
```

This also specifies the resource value for submenus.

Each item in a submenu in the menu bar also has its own name for X resources; for example, the '`File`' submenu has an item named '`Save (current buffer)`'. A resource specification for a submenu item looks like this:

`Emacs.pane.menubar.popup_*.`*menu*`.`*item*`.`*resource*`:` *value*

For example, here's how to specify the font for the '`Save (current buffer)`' item:

```
Emacs.pane.menubar.popup_*.File.Save (current buffer).fontList: 8x16
```

For an item in a second-level submenu, such as '`Spell-Check Message`' under '`Spell Checking`' under '`Tools`', the resource fits this template:

```
Emacs.pane.menubar.popup_*.popup_*.menu.resource: value
```

For example,

```
Emacs.pane.menubar.popup_*.popup_*.Spell Checking.Spell-Check Message: value
```

It's impossible to specify a resource for all the menu-bar items without also specifying it for the submenus as well. So if you want the submenu items to look different from the menu bar itself, you must ask for that in two steps. First, specify the resource for all of them; then, override the value for submenus alone. Here is an example:

```
Emacs.pane.menubar.*.fontList:  8x16
Emacs.pane.menubar.popup_*.fontList: 8x16
```

For toolkit pop-up menus, use '`menu*`' instead of '`pane.menubar`'. For example, to specify the font '`8x16`' for the pop-up menu items, write this:

```
Emacs.menu*.fontList:  8x16
```

Here is a list of the specific resources for menu bars and pop-up menus:

`armColor` The color to show in an armed button.

`fontList` The font to use.

`marginBottom`
`marginHeight`
`marginLeft`
`marginRight`
`marginTop`
`marginWidth`
: Amount of space to leave around the item, within the border.

`borderWidth`
: The width of border around the menu item, on all sides.

`shadowThickness`
: The width of the border shadow.

`bottomShadowColor`
: The color for the border shadow, on the bottom and the right.

`topShadowColor`
: The color for the border shadow, on the top and the left.

Appendix C Emacs 20 Antinews

For those users who live backwards in time, here is information about downgrading to Emacs version 20. We hope you will enjoy the greater simplicity that results from the absence of many Emacs 21 features.

- The display engine has been greatly simplified by eliminating support for variable-size characters and other non-text display features. This avoids the complexity of display layout in Emacs 21. To wit:
  - Variable-size characters are not supported in Emacs 20. You cannot use fonts which contain oversized characters, and using italics fonts can result in illegible display. However, text which uses variable-size fonts is unreadable anyway. With all characters in a frame layed out on a regular grid, each character having the same height and width, text is much easier to read.
  - Emacs does not display images, or play sounds. It just displays text, as you would expect from a **text** editor.
  - Specification of the font for a face now uses an XLFD font name, for compatibility with other X applications. This means that font attributes cannot be merged when combining faces; however, experience shows that mergers are bad economics. Face inheritance has also been removed, so no one can accumulate "too much face."
  - Several face appearance attributes such as 3D appearence, strike-through, and overline, have been eliminated.
  - Emacs now provides its own "lean and mean" scroll bars instead of using those from the X toolkit. Toggle buttons and radio buttons in menus now look just like any other menu item, which simplifies them, and prevents them from standing out and distracting your attention from the other menu items.
  - There are no toolbars and no tooltips; in particular, GUD mode cannot display variable values in a tooltip when you click on that variable's name. Instead, Emacs 20 provides a direct interface to the debugger, so that you can type appropriate debugger commands, such as `display foo` and `print bar`. As these commands use explicit words, their meaning is more self-evident.
  - Colors are not available on character terminals. If you *must* have colors, but cannot afford running X, you can now use the MS-DOG version of Emacs inside a DOS emulator.
  - The mode line is not mouse-sensitive, since it is meant only to display information. Use keyboard commands to switch between buffers, toggle read-only and modified status, switch minor modes on and off, etc.
  - The support for "wheeled" mice under X has been removed, because of their slow scroll rate, and because you will find less and less of

these mice as you go back in time. Instead Emacs 20 provides the `C-v` and `M-v` keys for scrolling. (You can also use the scroll bar, but be advised that it, too, may be absent in yet earlier Emacs versions.)

- Busy-cursor display is gone, as it was found to be too hard to draw on displays whose resolution is getting lower and lower. This means that you get the standard kind of cursor blinking that your terminal provides.
- Some aspects of Emacs appearance, such as the colors of the scroll bar and the menus, can only be controlled via X resources. Since colors aren't supported except on X, it doesn't make any sense doing this in any way but the X way. For those users who aren't privy to X arcana, we've provided good default colors that should make everybody happy.
- The variable `show-trailing-whitespace` has no special meaning, so trailing whitespace on a line is now always displayed correctly: as empty space. To see if a line ends with spaces or TABs, type `C-e` on that line. Likewise, empty lines at the end of the buffer are not marked in any way; use `M->` to see where the end of the buffer is.
- The spacing between text lines on the display now always follows the font design and the rules of your window manager. This provides for predictable appearance of the displayed text.

- Emacs 20 has simpler support for multi-lingual editing. While not as radical a simplification as Emacs 19 was, it goes a long way toward eliminating some of the annoying features:
  - Translations of the Emacs reference cards to other languages are no longer part of the distribution, because in the past we expect computer users to speak English.
  - To avoid extra confusion, many language environments have been eliminated. For example, '`Polish`' and '`Celtic`' (Latin-8) environments are not supported. The Latin-9 environment is gone, too, because you won't need the Euro sign in the past.
  - Emacs 20 always asks you which coding system to use when saving a buffer, unless it can use the same one that it used to read the buffer. It does not try to see if the preferred coding system is suitable.
  - Commands which provide detailed information about character sets and coding systems, such as `list-charset-chars`, `describe-character-set`, and the `C-u C-x =` key-sequence, no longer exist. The less said about non-ASCII characters, the better.
  - The terminal coding system cannot be set to something CCL-based, so keyboards which produce `KOI8` and DOS/Windows codepage

codes cannot be supported directly. Instead, you should use one of the input methods provided in the Leim package.

- As you move back through time, some systems will become unimportant or enter the vaporware phase, so Emacs 20 does not support them:
  - Emacs 20 cannot be built on GNU/Linux systems running on IA64 machines, and you cannot build a 64-bit Emacs on Solaris or Irix even though there are still 64-bit versions of those OSes.
  - LynxOS is also not supported, and neither is the Macintosh, though they still exist.
- The arrangement of menu bar items differs from most other GUI programs. We think that uniformity of look-and-feel is boring, and that Emacs' unique features require its unique menu-bar configuration.
- You cannot save the options that you set from the '`Options`' menu-bar menu; instead, you need to set all the options again each time you start a new session. However, if you follow the recommended practice and keep a single Emacs session running until you log out, you won't have to set the options very often.
- Emacs 20 does not pop up a buffer with error messages when an error is signaled during loading of the user's init file. Instead, it simply announces the fact that an error happened. To know where in the init file was that, insert `(message "foo")` lines judiciously into the file and look for those messages in the '`*Messages*`' buffer.
- Some commands no longer treat Transient Mark mode specially. For example, `ispell` doesn't spell-check the region when Transient Mark mode is in effect and the mark is active; instead, it checks the current buffer. (Transient Mark mode is alien to the spirit of Emacs, so we are planning to remove it altogether in an earlier version.)
- `C-Down-Mouse-3` does not show what would be in the menu bar when the menu bar is not displayed.
- For uniformity, the ⟨delete⟩ function key in Emacs 20 works exactly like the ⟨DEL⟩ key, on both text-only terminals and window systems—it always deletes backward. This eliminates the inconsistency of Emacs 21, where the key labeled ⟨delete⟩ deletes forward when you are using a window system, and backward on a text-only terminals.
- The ability to place backup files in special subdirectories (controlled by `backup-directory-alist`) has been eliminated. This makes finding your backup files much easier: they are always in the same directory as the original files.
- Emacs no longer refuses to load Lisp files compiled by incompatible versions of Emacs, which may contain invalid byte-code. Instead, Emacs now dumps core when it encounters such byte-code. However, this is a rare occurrence, and it won't happen at all when all Emacs versions merge together, in the distant past.

- The `C-x 5 1` command has been eliminated. If you want to delete all the frames but the current one, delete them one by one instead.
- CC Mode now enforces identical values for some customizable options, such as indentation style, for better consistency. In particular, if you select an indentation style for Java, the same style is used for C and C++ buffers as well.
- Isearch does not highlight other possible matches; it shows only the current match, to avoid distracting your attention. `Mouse-2` in the echo area during incremental search now signals an error, instead of inserting the current selection into the search string. But you can accomplish more or less the same job by typing `M-y`.
- The ability to specify a port number when editing remote files with `ange-ftp` was removed. Instead, Emacs 20 provides undocumented features in the function `ange-ftp-normal-login` (*Use the source, Luke!*) to specify the port.
- Emacs 20 does not check for changing time stamps of remote files, since the old FTP programs you will encounter in the past could not provide the time stamp anyway. Windows-style FTP clients which output the '`^M`' character at the end of each line get special handling from `ange-ftp` in Emacs 20, with unexpected results that should make your life more interesting.
- Many complicated display features, including highlighting of mouse-sensitive text regions and popping up help strings for menu items, don't work in the MS-DOS version. Spelling doesn't work on MS-DOS, and Eshell doesn't exist, so there's no workable shell-mode, either. This fits the spirit of MS-DOS, which resembles a dumb character terminal.
- The `woman` package has been removed, so Emacs users on non-Posix systems will need *a real man* to read manual pages. (Users who are not macho can read the Info documentation instead.)
- `recentf` has been removed, because we figure that you can remember the names of the files you edit frequently. With decreasing disk size, you should have fewer files anyway, so you won't notice the absence of this feature.
- The `field` property does not exist in Emacs 20, so various packages that run subsidiary programs in Emacs buffers cannot in general distinguish which text was user input and which was output from the subprocess. If you need to try to do this nonetheless, Emacs 20 provides a variable `comint-prompt-regexp`, which lets you try to distinguish input by recognizing prompt strings.
- We have eliminated the special major modes for Delphi sources, PostScript files, context diffs, and '`TODO`' files. Use Fundamental Mode instead.
- Many additional packages that unnecessarily complicate your life in Emacs 21 are absent in Emacs 20. You cannot browse C++ classes

with Ebrowse, access SQL data bases, access LDAP and other directory servers, or mix shell commands and Lisp functions using Eshell.

- To keep up with decreasing computer memory capacity and disk space, many other functions and files have been eliminated in Emacs 20.

Appendix D Emacs and the Mac OS

Emacs built on the Mac OS supports many of its major features: multiple frames, colors, scroll bars, menu bars, use of the mouse, fontsets, international characters, input methods, coding systems, and synchronous subprocesses (`call-process`). Much of this works in the same way as on other platforms and is therefore documented in the rest of this manual. This section describes the peculiarities of using Emacs under the Mac OS.

The following features of Emacs are not yet supported on the Mac: unexec (`dump-emacs`), asynchronous subprocesses (`start-process`), and networking (`open-network-connection`). As a result, packages such as Gnus, Ispell, and Comint do not work.

Since external Unix programs to handle commands such as `print-buffer` and `diff` are not available on the Mac OS, they are not supported in the Mac OS version.

D.1 Keyboard Input on the Mac

On the Mac, Emacs can use either the ⟨option⟩ key or the ⟨command⟩ key as the ⟨META⟩ key. If the value of the variable `mac-command-key-is-meta` is non-`nil` (its default value), Emacs uses the ⟨command⟩ key as the ⟨META⟩ key. Otherwise it uses the ⟨option⟩ key as the ⟨META⟩ key.

Most people should want to use the ⟨command⟩ key as the ⟨META⟩ key, so that dead-key processing with the ⟨option⟩ key will still work. This is useful for entering non-ASCII Latin characters directly from the Mac keyboard, for example.

Emacs recognizes the setting in the Keyboard control panel and supports international and alternative keyboard layouts (e.g., Dvorak). Selecting one of the layouts from the keyboard layout pull-down menu will affect how the keys typed on the keyboard are interpreted.

The Mac OS intercepts and handles certain key combinations (e.g., ⟨command⟩-⟨SPC⟩ for switching input languages). These will not be passed to Emacs.

The Mac keyboard ordinarily generates characters in the Mac Roman encoding. To use it for entering ISO Latin-1 characters directly, set the value of the variable `mac-keyboard-text-encoding` to `kTextEncodingISOLatin1`. Note that not all Mac Roman characters that can be entered at the keyboard can be converted to ISO Latin-1 characters.

To enter ISO Latin-2 characters directly from the Mac keyboard, set the value of `mac-keyboard-text-encoding` to `kTextEncodingISOLatin2`. Then let Emacs know that the keyboard generates Latin-2 codes, by typing

`C-x` ⟨RET⟩ `k iso-latin-2` ⟨RET⟩. To make this setting permanent, put this in your '`.emacs`' init file:

```
(set-keyboard-coding-system 'iso-latin-2)
```

D.2 International Character Set Support on the Mac

The Mac uses a non-standard encoding for the upper 128 single-byte characters. It also deviates from the ISO 2022 standard by using character codes in the range 128-159. The coding system `mac-roman` is used to represent this Mac encoding. It is used for editing files stored in this native encoding, and for displaying file names in Dired mode.

Any native (non-symbol) Mac font can be used to correctly display characters in the `mac-roman` coding system.

The fontset `fontset-mac` is created automatically when Emacs is run on the Mac. It displays characters in the `mac-roman` coding system using 12-point Monaco.

To insert characters directly in the `mac-roman` coding system, type `C-x` ⟨RET⟩ `k mac-roman` ⟨RET⟩, or put this in your '`.emacs`' init file:

```
(set-keyboard-coding-system 'mac-roman)
```

This is useful for editing documents in native Mac encoding.

You can use input methods provided either by LEIM (see Section 18.4 [Input Methods], page 221) or the Mac OS to enter international characters.

To use the former, see the International Character Set Support section of the manual.

To use input methods provided by the Mac OS, set the keyboard coding system accordingly using the `C-x` ⟨RET⟩ `k` command (`set-keyboard-coding-system`). For example, for Traditional Chinese, use '`chinese-big5`' as keyboard coding system; for Japanese, use '`sjis`', etc. Then select the desired input method in the keyboard layout pull-down menu.

The Mac clipboard and the Emacs kill ring (see Section 9.1 [Killing], page 85) are connected as follows: the most recent kill is copied to the clipboard when Emacs is suspended and the contents of the clipboard is inserted into the kill ring when Emacs resumes. The result is that you can yank a piece of text and paste it into another Mac application, or cut or copy one in another Mac application and yank it into a Emacs buffer.

The encoding of text selections must be specified using the commands `C-x` ⟨RET⟩ `x` (`set-selection-coding-system`) or `C-x` ⟨RET⟩ `X` (`set-next-selection-coding-system`) (e.g., for Traditional Chinese, use '`chinese-big5-mac`' and for Japanese, '`sjis-mac`'). See Section 18.8 [Specify Coding], page 227, for more details.

D.3 Environment Variables and Command Line Arguments.

Environment variables and command line arguments for Emacs can be set by modifying the '`STR#`' resources 128 and 129, respectively. A common environment variable that one may want to set is '`HOME`'.

The way to set an environment variable is by adding a string of the form

```
ENV_VAR=VALUE
```

to resource '`STR#`' number 128 using `ResEdit`. To set up the program to use unibyte characters exclusively, for example, add the string

```
EMACS_UNIBYTE=1
```

D.4 Volumes and Directories on the Mac

The directory structure in the Mac OS is seen by Emacs as

/volumename/filename

So when Emacs requests a file name, doing file name completion on '`/`' will display all volumes on the system. As in Unix, '`..`' can be used to go up a directory level.

To access files and folders on the desktop, look in the folder '`Desktop Folder`' in your boot volume (this folder is usually invisible in the Mac `Finder`).

Emacs creates the Mac folder '`:Preferences:Emacs:`' in the '`System Folder`' and uses it as the temporary directory. The Unix emulation code maps the Unix directory '`/tmp`' to it. Therefore it is best to avoid naming a volume '`tmp`'. If everything works correctly, the program should leave no files in it when it exits. You should be able to set the environment variable `TMPDIR` to use another directory but this folder will still be created.

D.5 Specifying Fonts on the Mac

It is rare that you need to specify a font name in Emacs; usually you specify face attributes instead. But when you do need to specify a font name in Emacs on the Mac, use a standard X font name:

-maker-family-weight-slant-widthtype-style...

...-pixels-height-horiz-vert-spacing-width-charset

See Section B.7 [Font X], page 516. Wildcards are supported as they are on X.

Native Apple fonts in Mac Roman encoding has maker name `apple` and charset `mac-roman`. For example 12-point Monaco can be specified by the name '`-apple-monaco-*-12-*-mac-roman`'.

Native Apple Traditional Chinese, Simplified Chinese, Japanese, and Korean fonts have charsets '`big5-0`', '`gb2312.1980-0`', '`jisx0208.1983-sjis`', and '`ksc5601.1989-0`', respectively.

Single-byte fonts converted from GNU fonts in BDF format, which are not in the Mac Roman encoding, have foundry, family, and character sets encoded in the names of their font suitcases. E.g., the font suitcase '`ETL-Fixed-ISO8859-1`' contains fonts which can be referred to by the name '`-ETL-fixed-*-iso8859-1`'.

D.6 Mac-Specific Lisp Functions

The function `do-applescript` takes a string argument, executes it as an AppleScript command, and returns the result as a string.

The function `mac-filename-to-unix` takes a Mac file name and returns the Unix equivalent. The function `unix-filename-to-mac` performs the opposite conversion. They are useful for constructing AppleScript commands to be passed to `do-applescript`.

Appendix E Emacs and MS-DOS

This section briefly describes the peculiarities of using Emacs under the MS-DOS "operating system" (also known as "MS-DOG"). If you build Emacs for MS-DOS, the binary will also run on Windows 3.X, Windows NT, Windows 9X, or OS/2 as a DOS application; the information in this chapter applies for all of those systems, if you use an Emacs that was built for MS-DOS.

Note that it is possible to build Emacs specifically for Windows NT or Windows 9X. If you do that, most of this chapter does not apply; instead, you get behavior much closer to what is documented in the rest of the manual, including support for long file names, multiple frames, scroll bars, mouse menus, and subprocesses. However, the section on text files and binary files does still apply. There are also two sections at the end of this chapter which apply specifically for Windows NT and 9X.

E.1 Keyboard and Mouse on MS-DOS

The PC keyboard maps use the left ⟨ALT⟩ key as the ⟨META⟩ key. You have two choices for emulating the ⟨SUPER⟩ and ⟨HYPER⟩ keys: choose either the right ⟨CTRL⟩ key or the right ⟨ALT⟩ key by setting the variables `dos-hyper-key` and `dos-super-key` to 1 or 2 respectively. If neither `dos-super-key` nor `dos-hyper-key` is 1, then by default the right ⟨ALT⟩ key is also mapped to the ⟨META⟩ key. However, if the MS-DOS international keyboard support program '`KEYB.COM`' is installed, Emacs will *not* map the right ⟨ALT⟩ to ⟨META⟩, since it is used for accessing characters like `~` and `@` on non-US keyboard layouts; in this case, you may only use the left ⟨ALT⟩ as ⟨META⟩ key.

The variable `dos-keypad-mode` is a flag variable that controls what key codes are returned by keys in the numeric keypad. You can also define the keypad ⟨ENTER⟩ key to act like `C-j`, by putting the following line into your '`_emacs`' file:

```
;; Make the Enter key from the Numeric keypad act as C-j.
(define-key function-key-map [kp-enter] [?\C-j])
```

The key that is called ⟨DEL⟩ in Emacs (because that's how it is designated on most workstations) is known as ⟨BS⟩ (backspace) on a PC. That is why the PC-specific terminal initialization remaps the ⟨BS⟩ key to act as ⟨DEL⟩; the ⟨DEL⟩ key is remapped to act as `C-d` for the same reasons.

Emacs built for MS-DOS recognizes `C-`⟨BREAK⟩ as a quit character, just like `C-g`. This is because Emacs cannot detect that you have typed `C-g` until it is ready for more input. As a consequence, you cannot use `C-g` to stop a running command (see Section 32.1 [Quitting], page 491). By contrast, `C-`⟨BREAK⟩ *is* detected as soon as you type it (as `C-g` is on other systems),

so it can be used to stop a running command and for emergency escape (see Section 32.2.8 [Emergency Escape], page 496).

Emacs on MS-DOS supports a mouse (on the default terminal only). The mouse commands work as documented, including those that use menus and the menu bar (see Section 1.4 [Menu Bar], page 28). Scroll bars don't work in MS-DOS Emacs. PC mice usually have only two buttons; these act as `Mouse-1` and `Mouse-2`, but if you press both of them together, that has the effect of `Mouse-3`. If the mouse does have 3 buttons, Emacs detects that at startup, and all the 3 buttons function normally, as on X.

Help strings for menu-bar and pop-up menus are displayed in the echo area when the mouse pointer moves across the menu items. Highlighting of mouse-sensitive text (see Section 17.4 [Mouse References], page 207) is also supported.

Some versions of mouse drivers don't report the number of mouse buttons correctly. For example, mice with a wheel report that they have 3 buttons, but only 2 of them are passed to Emacs; the clicks on the wheel, which serves as the middle button, are not passed. In these cases, you can use the `M-x msdos-set-mouse-buttons` command to tell Emacs how many mouse buttons to expect. You could make such a setting permanent by adding this fragment to your '`_emacs`' init file:

```
;; Treat the mouse like a 2-button mouse.
(msdos-set-mouse-buttons 2)
```

Emacs built for MS-DOS supports clipboard operations when it runs on Windows. Commands that put text on the kill ring, or yank text from the ring, check the Windows clipboard first, just as Emacs does on the X Window System (see Section 17.1 [Mouse Commands], page 203). Only the primary selection and the cut buffer are supported by MS-DOS Emacs on Windows; the secondary selection always appears as empty.

Due to the way clipboard access is implemented by Windows, the length of text you can put into the clipboard is limited by the amount of free DOS memory that is available to Emacs. Usually, up to 620KB of text can be put into the clipboard, but this limit depends on the system configuration and is lower if you run Emacs as a subprocess of another program. If the killed text does not fit, Emacs prints a message saying so, and does not put the text into the clipboard.

Null characters also cannot be put into the Windows clipboard. If the killed text includes null characters, Emacs does not put such text into the clipboard, and prints in the echo area a message to that effect.

The variable `dos-display-scancodes`, when non-`nil`, directs Emacs to display the ASCII value and the keyboard scan code of each keystroke; this feature serves as a complement to the `view-lossage` command, for debugging.

E.2 Display on MS-DOS

Display on MS-DOS cannot use font variants, like bold or italic, but it does support multiple faces, each of which can specify a foreground and a background color. Therefore, you can get the full functionality of Emacs packages that use fonts (such as `font-lock`, Enriched Text mode, and others) by defining the relevant faces to use different colors. Use the `list-colors-display` command (see Section 17.12 [Frame Parameters], page 213) and the `list-faces-display` command (see Section 11.1 [Faces], page 103) to see what colors and faces are available and what they look like.

The section Section E.6 [MS-DOS and MULE], page 547, later in this chapter, describes how Emacs displays glyphs and characters which aren't supported by the native font built into the DOS display.

When Emacs starts, it changes the cursor shape to a solid box. This is for compatibility with other systems, where the box cursor is the default in Emacs. This default shape can be changed to a bar by specifying the `cursor-type` parameter in the variable `default-frame-alist` (see Section 17.7 [Creating Frames], page 209). The MS-DOS terminal doesn't support a vertical-bar cursor, so the bar cursor is horizontal, and the *width* parameter, if specified by the frame parameters, actually determines its height. As an extension, the bar cursor specification can include the starting scan line of the cursor as well as its width, like this:

```
'(cursor-type bar width . start)
```

In addition, if the *width* parameter is negative, the cursor bar begins at the top of the character cell.

The MS-DOS terminal can only display a single frame at a time. The Emacs frame facilities work on MS-DOS much as they do on text-only terminals (see Chapter 17 [Frames], page 203). When you run Emacs from a DOS window on MS-Windows, you can make the visible frame smaller than the full screen, but Emacs still cannot display more than a single frame at a time.

The `mode4350` command switches the display to 43 or 50 lines, depending on your hardware; the `mode25` command switches to the default 80x25 screen size.

By default, Emacs only knows how to set screen sizes of 80 columns by 25, 28, 35, 40, 43 or 50 rows. However, if your video adapter has special video modes that will switch the display to other sizes, you can have Emacs support those too. When you ask Emacs to switch the frame to *n* rows by *m* columns dimensions, it checks if there is a variable called `screen-dimensions-`*n*`x`*m*, and if so, uses its value (which must be an integer) as the video mode to switch to. (Emacs switches to that video mode by calling the BIOS `Set Video Mode` function with the value of `screen-dimensions-`*n*`x`*m* in the `AL` register.) For example, suppose your adapter will switch to

66x80 dimensions when put into video mode 85. Then you can make Emacs support this screen size by putting the following into your '`_emacs`' file:

```
(setq screen-dimensions-66x80 85)
```

Since Emacs on MS-DOS can only set the frame size to specific supported dimensions, it cannot honor every possible frame resizing request. When an unsupported size is requested, Emacs chooses the next larger supported size beyond the specified size. For example, if you ask for 36x80 frame, you will get 40x80 instead.

The variables `screen-dimensions-`*nxm* are used only when they exactly match the specified size; the search for the next larger supported size ignores them. In the above example, even if your VGA supports 38x80 dimensions and you define a variable `screen-dimensions-38x80` with a suitable value, you will still get 40x80 screen when you ask for a 36x80 frame. If you want to get the 38x80 size in this case, you can do it by setting the variable named `screen-dimensions-36x80` with the same video mode value as `screen-dimensions-38x80`.

Changing frame dimensions on MS-DOS has the effect of changing all the other frames to the new dimensions.

E.3 File Names on MS-DOS

MS-DOS normally uses a backslash, '\', to separate name units within a file name, instead of the slash used on other systems. Emacs on MS-DOS permits use of either slash or backslash, and also knows about drive letters in file names.

On MS-DOS, file names are case-insensitive and limited to eight characters, plus optionally a period and three more characters. Emacs knows enough about these limitations to handle file names that were meant for other operating systems. For instance, leading dots '`.`' in file names are invalid in MS-DOS, so Emacs transparently converts them to underscores '`_`'; thus your default init file (see Section 31.7 [Init File], page 486) is called '`_emacs`' on MS-DOS. Excess characters before or after the period are generally ignored by MS-DOS itself; thus, if you visit the file '`LongFileName.EvenLongerExtension`', you will silently get '`longfile.eve`', but Emacs will still display the long file name on the mode line. Other than that, it's up to you to specify file names which are valid under MS-DOS; the transparent conversion as described above only works on file names built into Emacs.

The above restrictions on the file names on MS-DOS make it almost impossible to construct the name of a backup file (see Section 14.3.1.1 [Backup Names], page 151) without losing some of the original file name characters. For example, the name of a backup file for '`docs.txt`' is '`docs.tx~`' even if single backup is used.

If you run Emacs as a DOS application under Windows 9X, you can turn on support for long file names. If you do that, Emacs doesn't truncate file names or convert them to lower case; instead, it uses the file names that you specify, verbatim. To enable long file name support, set the environment variable `LFN` to '`y`' before starting Emacs. Unfortunately, Windows NT doesn't allow DOS programs to access long file names, so Emacs built for MS-DOS will only see their short 8+3 aliases.

MS-DOS has no notion of home directory, so Emacs on MS-DOS pretends that the directory where it is installed is the value of `HOME` environment variable. That is, if your Emacs binary, '`emacs.exe`', is in the directory '`c:/utils/emacs/bin`', then Emacs acts as if `HOME` were set to '`c:/utils/emacs`'. In particular, that is where Emacs looks for the init file '`_emacs`'. With this in mind, you can use '`~`' in file names as an alias for the home directory, as you would on GNU or Unix. You can also set `HOME` variable in the environment before starting Emacs; its value will then override the above default behavior.

Emacs on MS-DOS handles the directory name '`/dev`' specially, because of a feature in the emulator libraries of DJGPP that pretends I/O devices have names in that directory. We recommend that you avoid using an actual directory named '`/dev`' on any disk.

E.4 Text Files and Binary Files

GNU Emacs uses newline characters to separate text lines. This is the convention used on GNU and Unix.

MS-DOS and MS-Windows normally use carriage-return linefeed, a two-character sequence, to separate text lines. (Linefeed is the same character as newline.) Therefore, convenient editing of typical files with Emacs requires conversion of these end-of-line (EOL) sequences. And that is what Emacs normally does: it converts carriage-return linefeed into newline when reading files, and converts newline into carriage-return linefeed when writing files. The same mechanism that handles conversion of international character codes does this conversion also (see Section 18.6 [Coding Systems], page 223).

One consequence of this special format-conversion of most files is that character positions as reported by Emacs (see Section 4.9 [Position Info], page 50) do not agree with the file size information known to the operating system.

In addition, if Emacs recognizes from a file's contents that it uses newline rather than carriage-return linefeed as its line separator, it does not perform EOL conversion when reading or writing that file. Thus, you can read and edit files from GNU and Unix systems on MS-DOS with no special effort,

and they will retain their Unix-style end-of-line convention after you edit them.

The mode line indicates whether end-of-line translation was used for the current buffer. If MS-DOS end-of-line translation is in use for the buffer, a backslash '\' is displayed after the coding system mnemonic near the beginning of the mode line (see Section 1.3 [Mode Line], page 26). If no EOL translation was performed, the string '(Unix)' is displayed instead of the backslash, to alert you that the file's EOL format is not the usual carriage-return linefeed.

To visit a file and specify whether it uses DOS-style or Unix-style end-of-line, specify a coding system (see Section 18.8 [Specify Coding], page 227). For example, C-x ⟨RET⟩ c unix ⟨RET⟩ C-x C-f foobar.txt visits the file 'foobar.txt' without converting the EOLs; if some line ends with a carriage-return linefeed pair, Emacs will display '^M' at the end of that line. Similarly, you can direct Emacs to save a buffer in a specified EOL format with the C-x ⟨RET⟩ f command. For example, to save a buffer with Unix EOL format, type C-x ⟨RET⟩ f unix ⟨RET⟩ C-x C-s. If you visit a file with DOS EOL conversion, then save it with Unix EOL format, that effectively converts the file to Unix EOL style, like dos2unix.

When you use NFS or Samba to access file systems that reside on computers using GNU or Unix systems, Emacs should not perform end-of-line translation on any files in these file systems–not even when you create a new file. To request this, designate these file systems as *untranslated* file systems by calling the function add-untranslated-filesystem. It takes one argument: the file system name, including a drive letter and optionally a directory. For example,

```
(add-untranslated-filesystem "Z:")
```

designates drive Z as an untranslated file system, and

```
(add-untranslated-filesystem "Z:\\foo")
```

designates directory '\foo' on drive Z as an untranslated file system.

Most often you would use add-untranslated-filesystem in your '_emacs' file, or in 'site-start.el' so that all the users at your site get the benefit of it.

To countermand the effect of add-untranslated-filesystem, use the function remove-untranslated-filesystem. This function takes one argument, which should be a string just like the one that was used previously with add-untranslated-filesystem.

Designating a file system as untranslated does not affect character set conversion, only end-of-line conversion. Essentially, it directs Emacs to create new files with the Unix-style convention of using newline at the end of a line. See Section 18.6 [Coding Systems], page 223.

Some kinds of files should not be converted at all, because their contents are not really text. Therefore, Emacs on MS-DOS distinguishes certain files as *binary files*. (This distinction is not part of MS-DOS; it is

made by Emacs only.) Binary files include executable programs, compressed archives, etc. Emacs uses the file name to decide whether to treat a file as binary: the variable `file-name-buffer-file-type-alist` defines the file-name patterns that indicate binary files. If a file name matches one of the patterns for binary files (those whose associations are of the type `(`*pattern* `. t)`, Emacs reads and writes that file using the `no-conversion` coding system (see Section 18.6 [Coding Systems], page 223) which turns off *all* coding-system conversions, not only the EOL conversion. `file-name-buffer-file-type-alist` also includes file-name patterns for files which are known to be DOS-style text files with carriage-return linefeed EOL format, such as '`CONFIG.SYS`'; Emacs always writes those files with DOS-style EOLs.

If a file which belongs to an untranslated file system matches one of the file-name patterns in `file-name-buffer-file-type-alist`, the EOL conversion is determined by `file-name-buffer-file-type-alist`.

E.5 Printing and MS-DOS

Printing commands, such as `lpr-buffer` (see Section 30.5 [Hardcopy], page 438) and `ps-print-buffer` (see Section 30.6 [PostScript], page 439) can work in MS-DOS and MS-Windows by sending the output to one of the printer ports, if a Unix-style `lpr` program is unavailable. The same Emacs variables control printing on all systems (see Section 30.5 [Hardcopy], page 438), but in some cases they have different default values on MS-DOS and MS-Windows.

If you want to use your local printer, printing on it in the usual DOS manner, then set the Lisp variable `lpr-command` to `""` (its default value) and `printer-name` to the name of the printer port—for example, `"PRN"`, the usual local printer port (that's the default), or `"LPT2"`, or `"COM1"` for a serial printer. You can also set `printer-name` to a file name, in which case "printed" output is actually appended to that file. If you set `printer-name` to `"NUL"`, printed output is silently discarded (sent to the system null device).

On MS-Windows, when the Windows network software is installed, you can also use a printer shared by another machine by setting `printer-name` to the UNC share name for that printer–for example, `"//joes_pc/hp4si"`. (It doesn't matter whether you use forward slashes or backslashes here.) To find out the names of shared printers, run the command '`net view`' at a DOS command prompt to obtain a list of servers, and '`net view` *server-name*' to see the names of printers (and directories) shared by that server. Alternatively, click the '`Network Neighborhood`' icon on your desktop, and look for machines which share their printers via the network.

Some printers expect DOS codepage encoding of non-ASCII text, even though they are connected to a Windows machine which uses a different encoding for the same locale. For example, in the Latin-1 locale, DOS uses

codepage 850 whereas Windows uses codepage 1252. See Section E.6 [MS-DOS and MULE], page 547. When you print to such printers from Windows, you can use the `C-x RET c` (`universal-coding-system-argument`) command before `M-x lpr-buffer`; Emacs will then convert the text to the DOS codepage that you specify. For example, `C-x RET c cp850-dos RET M-x lpr-region RET` will print the region while converting it to the codepage 850 encoding. You may need to create the `cp`*nnn* coding system with `M-x codepage-setup`.

If you set `printer-name` to a file name, it's best to use an absolute file name. Emacs changes the working directory according to the default directory of the current buffer, so if the file name in `printer-name` is relative, you will end up with several such files, each one in the directory of the buffer from which the printing was done.

The commands `print-buffer` and `print-region` call the `pr` program, or use special switches to the `lpr` program, to produce headers on each printed page. MS-DOS and MS-Windows don't normally have these programs, so by default, the variable `lpr-headers-switches` is set so that the requests to print page headers are silently ignored. Thus, `print-buffer` and `print-region` produce the same output as `lpr-buffer` and `lpr-region`, respectively. If you do have a suitable `pr` program (for example, from GNU Textutils), set `lpr-headers-switches` to `nil`; Emacs will then call `pr` to produce the page headers, and print the resulting output as specified by `printer-name`.

Finally, if you do have an `lpr` work-alike, you can set the variable `lpr-command` to `"lpr"`. Then Emacs will use `lpr` for printing, as on other systems. (If the name of the program isn't `lpr`, set `lpr-command` to specify where to find it.) The variable `lpr-switches` has its standard meaning when `lpr-command` is not `""`. If the variable `printer-name` has a string value, it is used as the value for the `-P` option to `lpr`, as on Unix.

A parallel set of variables, `ps-lpr-command`, `ps-lpr-switches`, and `ps-printer-name` (see Section 30.7 [PostScript Variables], page 440), defines how PostScript files should be printed. These variables are used in the same way as the corresponding variables described above for non-PostScript printing. Thus, the value of `ps-printer-name` is used as the name of the device (or file) to which PostScript output is sent, just as `printer-name` is used for non-PostScript printing. (There are two distinct sets of variables in case you have two printers attached to two different ports, and only one of them is a PostScript printer.)

The default value of the variable `ps-lpr-command` is `""`, which causes PostScript output to be sent to the printer port specified by `ps-printer-name`, but `ps-lpr-command` can also be set to the name of a program which will accept PostScript files. Thus, if you have a non-PostScript printer, you can set this variable to the name of a PostScript interpreter program (such as Ghostscript). Any switches that need to be passed to the interpreter program are specified using `ps-lpr-switches`. (If the value of `ps-printer-`

`name` is a string, it will be added to the list of switches as the value for the `-P` option. This is probably only useful if you are using `lpr`, so when using an interpreter typically you would set `ps-printer-name` to something other than a string so it is ignored.)

For example, to use Ghostscript for printing on an Epson printer connected to the '`LPT2`' port, put this in your '`_emacs`' file:

```
(setq ps-printer-name t)  ; Ghostscript doesn't understand -P
(setq ps-lpr-command "c:/gs/gs386")
(setq ps-lpr-switches '("-q" "-dNOPAUSE"
"-sDEVICE=epson"
"-r240x72"
"-sOutputFile=LPT2"
"-Ic:/gs"))
```

(This assumes that Ghostscript is installed in the '`"c:/gs"`' directory.)

For backwards compatibility, the value of `dos-printer` (`dos-ps-printer`), if it has a value, overrides the value of `printer-name` (`ps-printer-name`), on MS-DOS and MS-Windows only.

E.6 International Support on MS-DOS

Emacs on MS-DOS supports the same international character sets as it does on Unix and other platforms (see Chapter 18 [International], page 219), including coding systems for converting between the different character sets. However, due to incompatibilities between MS-DOS/MS-Windows and Unix, there are several DOS-specific aspects of this support that users should be aware of. This section describes these aspects.

`M-x dos-codepage-setup`
: Set up Emacs display and coding systems as appropriate for the current DOS codepage.

`M-x codepage-setup`
: Create a coding system for a certain DOS codepage.

MS-DOS is designed to support one character set of 256 characters at any given time, but gives you a variety of character sets to choose from. The alternative character sets are known as *DOS codepages*. Each codepage includes all 128 ASCII characters, but the other 128 characters (codes 128 through 255) vary from one codepage to another. Each DOS codepage is identified by a 3-digit number, such as 850, 862, etc.

In contrast to X, which lets you use several fonts at the same time, MS-DOS doesn't allow use of several codepages in a single session. Instead, MS-DOS loads a single codepage at system startup, and you must reboot

MS-DOS to change it[1]. Much the same limitation applies when you run DOS executables on other systems such as MS-Windows.

If you invoke Emacs on MS-DOS with the '`--unibyte`' option (see Section B.2 [Initial Options], page 508), Emacs does not perform any conversion of non-ASCII characters. Instead, it reads and writes any non-ASCII characters verbatim, and sends their 8-bit codes to the display verbatim. Thus, unibyte Emacs on MS-DOS supports the current codepage, whatever it may be, but cannot even represent any other characters.

For multibyte operation on MS-DOS, Emacs needs to know which characters the chosen DOS codepage can display. So it queries the system shortly after startup to get the chosen codepage number, and stores the number in the variable `dos-codepage`. Some systems return the default value 437 for the current codepage, even though the actual codepage is different. (This typically happens when you use the codepage built into the display hardware.) You can specify a different codepage for Emacs to use by setting the variable `dos-codepage` in your init file.

Multibyte Emacs supports only certain DOS codepages: those which can display Far-Eastern scripts, like the Japanese codepage 932, and those that encode a single ISO 8859 character set.

The Far-Eastern codepages can directly display one of the MULE character sets for these countries, so Emacs simply sets up to use the appropriate terminal coding system that is supported by the codepage. The special features described in the rest of this section mostly pertain to codepages that encode ISO 8859 character sets.

For the codepages which correspond to one of the ISO character sets, Emacs knows the character set name based on the codepage number. Emacs automatically creates a coding system to support reading and writing files that use the current codepage, and uses this coding system by default. The name of this coding system is `cp`*nnn*, where *nnn* is the codepage number.[2]

All the `cp`*nnn* coding systems use the letter '`D`' (for "DOS") as their mode-line mnemonic. Since both the terminal coding system and the default coding system for file I/O are set to the proper `cp`*nnn* coding system at startup, it is normal for the mode line on MS-DOS to begin with '`-DD\-`'. See Section 1.3 [Mode Line], page 26. Far-Eastern DOS terminals do not use the `cp`*nnn* coding systems, and thus their initial mode line looks like on Unix.

---

[1] Normally, one particular codepage is burnt into the display memory, while other codepages can be installed by modifying system configuration files, such as '`CONFIG.SYS`', and rebooting.

[2] The standard Emacs coding systems for ISO 8859 are not quite right for the purpose, because typically the DOS codepage does not match the standard ISO character codes. For example, the letter '`ç`' ('`c`' with cedilla) has code 231 in the standard Latin-1 character set, but the corresponding DOS codepage 850 uses code 135 for this glyph.

Since the codepage number also indicates which script you are using, Emacs automatically runs `set-language-environment` to select the language environment for that script (see Section 18.3 [Language Environments], page 220).

If a buffer contains a character belonging to some other ISO 8859 character set, not the one that the chosen DOS codepage supports, Emacs displays it using a sequence of ASCII characters. For example, if the current codepage doesn't have a glyph for the letter 'ò' (small 'o' with a grave accent), it is displayed as '{`o}', where the braces serve as a visual indication that this is a single character. (This may look awkward for some non-Latin characters, such as those from Greek or Hebrew alphabets, but it is still readable by a person who knows the language.) Even though the character may occupy several columns on the screen, it is really still just a single character, and all Emacs commands treat it as one.

Not all characters in DOS codepages correspond to ISO 8859 characters—some are used for other purposes, such as box-drawing characters and other graphics. Emacs cannot represent these characters internally, so when you read a file that uses these characters, they are converted into a particular character code, specified by the variable `dos-unsupported-character-glyph`.

Emacs supports many other characters sets aside from ISO 8859, but it cannot display them on MS-DOS. So if one of these multibyte characters appears in a buffer, Emacs on MS-DOS displays them as specified by the `dos-unsupported-character-glyph` variable; by default, this glyph is an empty triangle. Use the `C-u C-x =` command to display the actual code and character set of such characters. See Section 4.9 [Position Info], page 50.

By default, Emacs defines a coding system to support the current codepage. To define a coding system for some other codepage (e.g., to visit a file written on a DOS machine in another country), use the `M-x codepage-setup` command. It prompts for the 3-digit code of the codepage, with completion, then creates the coding system for the specified codepage. You can then use the new coding system to read and write files, but you must specify it explicitly for the file command when you want to use it (see Section 18.8 [Specify Coding], page 227).

These coding systems are also useful for visiting a file encoded using a DOS codepage, using Emacs running on some other operating system.

MS-Windows provides its own codepages, which are different from the DOS codepages for the same locale. For example, DOS codepage 850 supports the same character set as Windows codepage 1252; DOS codepage 855 supports the same character set as Windows codepage 1251, etc. The MS-Windows version of Emacs uses the current codepage for display when invoked with the '`-nw`' option.

E.7 Subprocesses on MS-DOS

Because MS-DOS is a single-process "operating system," asynchronous subprocesses are not available. In particular, Shell mode and its variants do not work. Most Emacs features that use asynchronous subprocesses also don't work on MS-DOS, including Shell mode and GUD. When in doubt, try and see; commands that don't work print an error message saying that asynchronous processes aren't supported.

Compilation under Emacs with `M-x compile`, searching files with `M-x grep` and displaying differences between files with `M-x diff` do work, by running the inferior processes synchronously. This means you cannot do any more editing until the inferior process finishes.

Spell checking also works, by means of special support for synchronous invocation of the `ispell` program. This is slower than the asynchronous invocation on Unix.

Instead of the Shell mode, which doesn't work on MS-DOS, you can use the `M-x eshell` command. This invokes the Eshell package that implements a Unix-like shell entirely in Emacs Lisp.

By contrast, Emacs compiled as native Windows application **does** support asynchronous subprocesses. See Section E.8 [Windows Processes], page 551.

Printing commands, such as `lpr-buffer` (see Section 30.5 [Hardcopy], page 438) and `ps-print-buffer` (see Section 30.6 [PostScript], page 439), work in MS-DOS by sending the output to one of the printer ports. See Section E.5 [MS-DOS Printing], page 545.

When you run a subprocess synchronously on MS-DOS, make sure the program terminates and does not try to read keyboard input. If the program does not terminate on its own, you will be unable to terminate it, because MS-DOS provides no general way to terminate a process. Pressing `C-c` or `C-`⟨BREAK⟩ might sometimes help in these cases.

Accessing files on other machines is not supported on MS-DOS. Other network-oriented commands such as sending mail, Web browsing, remote login, etc., don't work either, unless network access is built into MS-DOS with some network redirector.

Dired on MS-DOS uses the `ls-lisp` package where other platforms use the system `ls` command. Therefore, Dired on MS-DOS supports only some of the possible options you can mention in the `dired-listing-switches` variable. The options that work are '`-A`', '`-a`', '`-c`', '`-i`', '`-r`', '`-S`', '`-s`', '`-t`', and '`-u`'.

E.8 Subprocesses on Windows 95 and NT

Emacs compiled as a native Windows application (as opposed to the DOS version) includes full support for asynchronous subprocesses. In the Windows version, synchronous and asynchronous subprocesses work fine on both Windows 95 and Windows NT as long as you run only 32-bit Windows applications. However, when you run a DOS application in a subprocess, you may encounter problems or be unable to run the application at all; and if you run two DOS applications at the same time in two subprocesses, you may have to reboot your system.

Since the standard command interpreter (and most command line utilities) on Windows 95 are DOS applications, these problems are significant when using that system. But there's nothing we can do about them; only Microsoft can fix them.

If you run just one DOS application subprocess, the subprocess should work as expected as long as it is "well-behaved" and does not perform direct screen access or other unusual actions. If you have a CPU monitor application, your machine will appear to be 100% busy even when the DOS application is idle, but this is only an artifact of the way CPU monitors measure processor load.

You must terminate the DOS application before you start any other DOS application in a different subprocess. Emacs is unable to interrupt or terminate a DOS subprocess. The only way you can terminate such a subprocess is by giving it a command that tells its program to exit.

If you attempt to run two DOS applications at the same time in separate subprocesses, the second one that is started will be suspended until the first one finishes, even if either or both of them are asynchronous.

If you can go to the first subprocess, and tell it to exit, the second subprocess should continue normally. However, if the second subprocess is synchronous, Emacs itself will be hung until the first subprocess finishes. If it will not finish without user input, then you have no choice but to reboot if you are running on Windows 95. If you are running on Windows NT, you can use a process viewer application to kill the appropriate instance of ntvdm instead (this will terminate both DOS subprocesses).

If you have to reboot Windows 95 in this situation, do not use the `Shutdown` command on the `Start` menu; that usually hangs the system. Instead, type `CTL-ALT-`⟨DEL⟩ and then choose `Shutdown`. That usually works, although it may take a few minutes to do its job.

E.9 Using the System Menu on Windows

Emacs compiled as a native Windows application normally turns off the Windows feature that tapping the ⟨ALT⟩ key invokes the Windows menu. The reason is that the ⟨ALT⟩ also serves as ⟨META⟩ in Emacs. When using Emacs,

users often press the ⟨META⟩ key temporarily and then change their minds; if this has the effect of bringing up the Windows menu, it alters the meaning of subsequent commands. Many users find this frustrating.

You can reenable Windows's default handling of tapping the ⟨ALT⟩ key by setting `w32-pass-alt-to-system` to a non-`nil` value.

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 the available GNU software, please see the latest issue of the GNU's Bulletin. The list is much too long to include here.

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

[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.

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.

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.

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.

---

[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.

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 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

[3] Several such companies now exist.

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

---

[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.

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.

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 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 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, 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 pro-

gramming, 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.

Glossary

Abbrev
: An abbrev is a text string which 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 24 [Abbrevs], page 345.

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 32.1 [Quitting], page 491.

Alt
: Alt is the name of a modifier bit which 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⟩ which is really a ⟨META⟩ key.) See Section 2.1 [User Input], page 31.

ASCII character
: An ASCII character is either an ASCII control character or an ASCII printing character. See Section 2.1 [User Input], page 31.

ASCII control character
: An ASCII control character is the Control version of an uppercase letter, or the Control version of one of the characters '`@[\]^_?`'.

ASCII printing character
: ASCII printing characters include letters, digits, space, and these punctuation characters: '`!@#$%^& *()_-+=|\~‘ {}[]:;"’ <>,.?/`'.

Auto Fill Mode
: Auto Fill mode is a minor mode in which text that you insert is automatically broken into lines of fixed width. See Section 21.5 [Filling], page 248.

Auto Saving
: Auto saving is the practice of saving the contents of an Emacs buffer in a specially-named file, so that the information will not be lost if the buffer is lost due to a system error or user error. See Section 14.5 [Auto Save], page 158.

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 14.3.1 [Backup], page 150.

Balance Parentheses
: Emacs can balance parentheses manually or automatically. Manual balancing is done by the commands to move over balanced expressions (see Section 22.2 [Lists], page 274). Automatic balancing is done by blinking or highlighting the parenthesis that matches one just inserted (see Section 22.6 [Matching Parens], page 290).

Bind
: To bind a key sequence means to give it a binding (q.v.). See Section 31.4.5 [Rebinding], page 477.

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 the user types that sequence. See Section 2.3 [Commands], page 34. 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 31.4.1 [Keymaps], page 474.

Blank Lines
: Blank lines are lines that contain only whitespace. Emacs has several commands for operating on the blank lines in the buffer.

Buffer
: The buffer is the basic editing unit; one buffer corresponds to one text being edited. You can 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 (q.v.). Most buffers are visiting (q.v.) some file. See Chapter 15 [Buffers], page 185.

Buffer Selection History
: Emacs keeps a buffer selection history which records how recently each Emacs buffer has been selected. This is used for choosing a buffer to select. See Chapter 15 [Buffers], page 185.

Button Down Event
: A button down event is the kind of input event generated right away when you press a mouse button. See Section 31.4.10 [Mouse Buttons], page 482.

`C-`
: `C-` in the name of a character is an abbreviation for Control. See Section 2.1 [User Input], page 31.

`C-M-`
: `C-M-` in the name of a character is an abbreviation for Control-Meta. See Section 2.1 [User Input], page 31.

Case Conversion
: Case conversion means changing text from upper case to lower case or vice versa. See Section 21.6 [Case], page 254, for the commands for case conversion.

Character
: Characters form the contents of an Emacs buffer; see Section 2.4 [Text Characters], page 35. 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 31.

Character Set
: Emacs supports a number of character sets, each of which represents a particular alphabet or script. See Chapter 18 [International], page 219.

Click Event
: A click event is the kind of input event generated when you press a mouse button and release it without moving the mouse. See Section 31.4.10 [Mouse Buttons], page 482.

Coding System
: A coding system is an encoding for representing 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 18.6 [Coding Systems], page 223.

Command
: A command is a Lisp function specially defined to be able to serve as a key binding in Emacs. 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 2.3 [Commands], page 34.

Command Name
: A command name is the name of a Lisp symbol which is a command (see Section 2.3 [Commands], page 34). You can invoke any command by its name using `M-x` (see Chapter 6 [M-x], page 65).

Comment
: A comment is text in a program which is intended only for humans reading the program, and which is marked specially 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 22.7 [Comments], page 291.

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 23.1 [Compilation], page 331).

Complete Key
: A complete key is a key sequence which 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 (q.v.) 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 33.

Completion
: Completion is what Emacs does when it automatically fills out 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 occurs when ⟨TAB⟩, ⟨SPC⟩ or ⟨RET⟩ is typed. See Section 5.3 [Completion], page 57.

Continuation Line
: When a line of text is longer than the width of the window, it takes up more than one screen line when displayed. We say that the text line is continued, and all screen lines used for it after the first are called continuation lines. See Chapter 4 [Basic Editing], page 41.

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 31.

Copyleft
: A copyleft is a notice giving the public legal permission to redistribute a program or other work of art. Copylefts are used by left-wing programmers to promote freedom and cooperation, just as copyrights are used by right-wing programmers to gain power over other people.

  The particular form of copyleft used by the GNU project is called the GNU General Public License. See [Copying], page 5.

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 15 [Buffers], page 185.

Current Line
: The line point is on (see Section 1.1 [Point], page 23).

Current Paragraph
: The paragraph that point is in. If point is between paragraphs, the current paragraph is the one that follows point. See Section 21.3 [Paragraphs], page 246.

Current Defun
: The defun (q.v.) that point is in. If point is between defuns, the current defun is the one that follows point. See Section 22.4 [Defuns], page 277.

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 Chapter 4 [Basic Editing], page 41.

Customization
: Customization is making minor changes in the way Emacs works. It is often done by setting variables (see Section 31.2 [Variables], page 457) or by rebinding key sequences (see Section 31.4.1 [Keymaps], page 474).

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 55.

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. See Section 5.1 [Minibuffer File], page 56.

Defun
: A defun is a list at the top level of parenthesis or bracket structure in a program. It is so named because most such lists in Lisp programs are calls to the Lisp function `defun`. See Section 22.4 [Defuns], page 277.

⟨DEL⟩
: ⟨DEL⟩ is a character that runs the command to delete one character of text. See Chapter 4 [Basic Editing], page 41.

Deletion
: Deletion means erasing text without copying it into the kill ring (q.v.). The alternative is killing (q.v.). See Section 9.1 [Killing], page 85.

Deletion of Files
: Deleting a file means erasing it from the file system. See Section 14.10 [Misc File Ops], page 181.

Deletion of Messages
: Deleting a message 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 27.4 [Rmail Deletion], page 369.

Deletion of Windows
: Deleting a window means eliminating it from the screen. Other windows expand to use up the space. The deleted window can never come back, but no actual text is thereby lost. See Chapter 16 [Windows], page 195.

Directory
: File directories are named collections in the file system, within which you can place individual files or subdirectories. See Section 14.8 [Directories], page 180.

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 28 [Dired], page 387.

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 31.4.11 [Disabling], page 484.

Down Event
: Short for 'button down event'.

Drag Event
: A drag event is the kind of input event generated when you press a mouse button, move the mouse, and then release the button. See Section 31.4.10 [Mouse Buttons], page 482.

Dribble File
: A file into which Emacs writes all the characters that the user types on the keyboard. Dribble files are used to make a record for debugging Emacs bugs. Emacs does not make a dribble file unless you tell it to. See Section 32.3 [Bugs], page 497.

Echo Area
: The echo area is the bottom line of the screen, used for echoing the arguments to commands, for asking questions, and printing 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 24.

Echoing
: Echoing is acknowledging the receipt of commands 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.

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 printing an error message (q.v.). Type-ahead is discarded. Then Emacs is ready to read another editing command.

Error Message
: An error message is a single line of output displayed by Emacs when the user asks for something impossible to do (such as, killing text forward when point is at the end of the buffer). They 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.

Expunging
: Expunging an Rmail file or Dired buffer is an operation that truly discards the messages or files you have previously flagged for deletion.

File Locking
: Emacs used file locking to notice when two different users start to edit one file at the same time. See Section 14.3.2 [Interlocking], page 154.

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).

  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.

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 21.5 [Filling], page 248.

Filling
: Filling text means shifting text between consecutive lines so that all the lines are approximately the same length. See Section 21.5 [Filling], page 248.

Formatted Text
: Formatted text is text that displays with formatting information while you edit. Formatting information includes fonts, colors, and specified margins. See Section 21.11 [Formatted Text], page 265.

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 windowing system, all the frames can be visible at the same time. See Chapter 17 [Frames], page 203.

Function Key
: A function key is a key on the keyboard that sends input but does not correspond to any character. See Section 31.4.7 [Function Keys], page 479.

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 24 [Abbrevs], page 345.

Global Keymap
: The global keymap (q.v.) contains key bindings that are in effect except when overridden by local key bindings in a major mode's local keymap (q.v.). See Section 31.4.1 [Keymaps], page 474.

Global Mark Ring
: The global mark ring records the series of buffers you have recently set a mark in. In many cases you can use this to backtrack through buffers you have been editing in, or in which you have found tags. See Section 8.6 [Global Mark Ring], page 82.

Global Substitution
: Global substitution means replacing each occurrence of one string by another string through a large amount of text. See Section 12.7 [Replace], page 132.

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 31.2 [Variables], page 457.

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 Chapter 4 [Basic Editing], page 41.

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.

Hardcopy
: Hardcopy means printed output. Emacs has commands for making printed listings of text in Emacs buffers. See Section 30.5 [Hardcopy], page 438.

⟨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 any command does. See Chapter 7 [Help], page 67.

Hyper
: Hyper is the name of a modifier bit which 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 2.1 [User Input], page 31.

Inbox
: An inbox is a file in which mail is delivered by the operating system. Rmail transfers mail from inboxes to Rmail files (q.v.) in which the mail is then stored permanently or until explicitly deleted. See Section 27.5 [Rmail Inbox], page 370.

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 20 [Indentation], page 239.

Indirect Buffer
: An indirect buffer is a buffer that shares the text of another buffer, called its base buffer. See Section 15.6 [Indirect Buffers], page 191.

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 31.

Input Method
: An input method is a system for entering non-ASCII text characters by typing sequences of ASCII characters (q.v.). See Section 18.4 [Input Methods], page 221.

Insertion
: Insertion means copying text into the buffer, either from the keyboard or from some other place in Emacs.

Interlocking
: Interlocking is a feature for warning when you start to alter a file that someone else is already editing. See Section 14.3.2 [Simultaneous Editing], page 154.

Justification
: Justification means adding extra spaces to lines of text to make them come exactly to a specified width. See Section 21.5 [Filling], page 248.

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. See Section 31.3 [Keyboard Macros], page 470.

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 33.

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 31.4.1 [Keymaps], page 474.

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. See Section 31.5 [Keyboard Translations], page 484.

Kill Ring
: The kill ring is where all text you have killed 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 89.

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 to erase text do killing, as opposed to deletion (q.v.). See Section 9.1 [Killing], page 85.

Killing Jobs
: 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 3.1 [Exiting], page 38.

Language Environment
: Your choice of language environment specifies defaults for the input method (q.v.) and coding system (q.v.). See Section 18.3 [Language Environments], page 220. These defaults are relevant if you edit non-ASCII text (see Chapter 18 [International], page 219).

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 22.2 [Lists], page 274.

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 24 [Abbrevs], page 345.

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 31.4.1 [Keymaps], page 474.

Local Variable
: A local value of a variable (q.v.) applies to only one buffer. See Section 31.2.4 [Locals], page 466.

`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 31.

`M-C-`
: `M-C-` in the name of a character is an abbreviation for Control-Meta; it means the same thing as `C-M-`. 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 31.

`M-x` `M-x` is the key sequence which 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 [M-x], page 65.

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 26 [Sending Mail], page 357. See Chapter 27 [Rmail], page 367, for how to read mail.

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 26.6 [Mail Methods], page 366.

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 Chapter 19 [Major Modes], page 235.

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 77.

Mark Ring
The mark ring is used to hold several recent previous locations of the mark, just 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 8.5 [Mark Ring], page 81.

Menu Bar The menu bar is the 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 17.15 [Menu Bars], page 215.

Message See 'mail.'

Meta Meta is the name of a modifier bit which a command character may have. It is present in a character if the character is typed with the ⟨META⟩ key held down. Such characters are given 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 31.

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 55.

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 5.4 [Minibuffer History], page 61.

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. See Section 31.1 [Minor Modes], page 455.

Minor Mode Keymap
: 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 31.4.1 [Keymaps], page 474.

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 26.

Modified Buffer
: A buffer (q.v.) is modified if its text has been changed since the last time the buffer was saved (or since when it was created, if it has never been saved). See Section 14.3 [Saving], page 148.

Moving Text
: Moving text means erasing it from one place and inserting it in another. The usual way to move text by killing (q.v.) and then yanking (q.v.). See Section 9.1 [Killing], page 85.

MULE
: MULE refers to the Emacs features for editing non-ASCII text using multibyte characters (q.v.). See Chapter 18 [International], page 219.

Multibyte Character
: A multibyte character is a character that takes up several buffer positions. Emacs uses multibyte characters to represent non-ASCII text, since the number of non-ASCII characters is much more than 256. See Section 18.1 [International Intro], page 219.

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 97.

Narrowing
: Narrowing means creating a restriction (q.v.) that limits editing in the current buffer to only a part of the text in the buffer. Text outside that part is inaccessible to the user until the boundaries are widened again, but it is still there, and saving the file saves it all. See Section 30.9 [Narrowing], page 443.

Newline
: Control-J characters in the buffer terminate lines of text and are therefore also called newlines. See Section 2.4 [Text Characters], page 35.

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 52.

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 the right. See Section 31.1 [Minor Modes], page 455.

Page
: A page is a unit of text, delimited by formfeed characters (ASCII control-L, code 014) coming at the beginning of a line. Some Emacs commands are provided for moving over and operating on pages. See Section 21.4 [Pages], page 247.

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 21.3 [Paragraphs], page 246.

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. See Section 31.6 [Syntax], page 485.

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 Chapter 4 [Basic], page 41.

Prefix Argument
: See 'numeric argument.'

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 33.

Primary Rmail File
: Your primary Rmail file is the file named '`RMAIL`' in your home directory. That's where Rmail stores your incoming mail, unless you specify a different file name. See Chapter 27 [Rmail], page 367.

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 kill commands set the primary selection and the yank command uses the primary selection when appropriate. See Section 9.1 [Killing], page 85.

Prompt
: A prompt is text printed to ask the user 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 55); the echoing which 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 24).

Quitting
: Quitting means canceling a partially typed command or a running command, using `C-g` (or `C-`⟨BREAK⟩ on MS-DOS). See Section 32.1 [Quitting], page 491.

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 Chapter 4 [Basic Editing], page 41.

Quoting File Names
: Quoting a file name turns off the special significance of constructs such as '`$`', '`~`' and '`:`'. See Section 14.13 [Quoted File Names], page 183.

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 15 [Buffers], page 185.

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 opposite corner. See Section 9.4 [Rectangles], page 93.

Recursive Editing Level
: A recursive editing level is a state in which part of the execution of a command involves asking the user 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 indicates recursive editing levels with square brackets (‘[’ and ‘]’). See Section 30.13 [Recursive Edit], page 447.

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 23.

Regexp
: See ‘regular expression.’

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 77.

Registers
: Registers are named slots in which text or buffer positions or rectangles can be saved for later use. See Chapter 10 [Registers], page 97.

Regular Expression
: A regular expression is a pattern that can match various text strings; for example, ‘`l[0-9]+`’ matches ‘`l`’ followed by one or more digits. See Section 12.5 [Regexps], page 125.

Repeat Count
: See ‘numeric argument.’

Replacement
: See ‘global substitution.’

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.). See Section 30.9 [Narrowing], page 443.

⟨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 31.

Rmail File
: An Rmail file is a file containing text in a special format used by Rmail for storing mail. See Chapter 27 [Rmail], page 367.

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 14.3 [Saving], page 148.

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 17.13 [Scroll Bars], page 214.

Scrolling
: Scrolling means shifting the text in the Emacs window so as to see a different part of the buffer. See Chapter 11 [Display], page 103.

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 119.

Search Path
: A search path is a list of directory names, 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 23.7 [Lisp Libraries], page 339.

Secondary Selection
: The secondary selection is one particular X selection; 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 17.2 [Secondary Selection], page 206.

Selecting
: Selecting a buffer means making it the current (q.v.) buffer. See Chapter 15 [Buffers], page 185.

Selection
: The X window system allows an application program to specify named 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.

Self-Documentation
: Self-documentation is the feature of Emacs which 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 67.

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 21.2 [Sentences], page 245.

Sexp
: A sexp (short for "s-expression") is the basic syntactic unit of Lisp in its textual form: either a list, or Lisp atom. Many Emacs commands operate on sexps. The term 'sexp' is generalized to languages other than Lisp, to mean a syntactically recognizable expression. See Section 22.2 [Lists], page 274.

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 work. Emacs detects all cases of simultaneous editing and warns one of the users to investigate. See Section 14.3.2 [Simultaneous Editing], page 154.

String
: A string is a kind of Lisp data object which 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 'global substitution'.

Syntax Table
: The syntax table tells Emacs which characters are part of a word, which characters balance each other like parentheses, etc. See Section 31.6 [Syntax], page 485.

Super
: Super is the name of a modifier bit which 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 2.1 [User Input], page 31.

Tags Table
: A tags table is a file that serves as an index to the function definitions in one or more other files. See Section 22.16 [Tags], page 301.

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 32.3 [Bugs], page 497.

Text
: Two meanings (see Chapter 21 [Text], page 243):
  - Data consisting of a sequence of characters, as opposed to binary numbers, images, graphics commands, executable programs, and the like. The contents of an Emacs buffer are always text in this sense.
  - Data consisting of written human language, as opposed to programs, or following the stylistic conventions of human language.

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 17.16 [Tool Bars], page 216.

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 32.1 [Quitting], page 491.

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, sexps (q.v.) or lines (see Section 13.2 [Transpose], page 137).

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 also 'continuation line.' See Chapter 4 [Basic Editing], page 41.

Undoing
: Undoing means making your previous editing go in reverse, bringing back the text that existed earlier in the editing session. See Section 4.4 [Undo], page 45.

User Option
: A user option is a variable (q.v.) that exists so that you can customize Emacs by setting it to a new value. See Section 31.2 [Variables], page 457.

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 Section 31.2 [Variables], page 457, 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 14.7 [Version Control], page 161.

Visiting
: Visiting a file means loading its contents into a buffer (q.v.) where they can be edited. See Section 14.2 [Visiting], page 145.

Whitespace
: Whitespace is any run of consecutive formatting characters (space, tab, newline, and backspace).

Widening
: Widening is removing any restriction (q.v.) on the current buffer; it is the opposite of narrowing (q.v.). See Section 30.9 [Narrowing], page 443.

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 23, for basic information on how Emacs uses the screen. See Chapter 16 [Windows], page 195, for commands to control the use of windows.

Word Abbrev
: See ‘abbrev.’

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 123.

WYSIWYG
: WYSIWYG stands for “What you see is what you get.” Emacs generally provides WYSIWYG editing for files of characters; in Enriched mode (see Section 21.11 [Formatted Text], page 265), it provides WYSIWYG editing for files that include text formatting information.

Yanking
: Yanking means reinserting text previously killed. 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 89.

Key (Character) 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

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

R

S

T

U

V

W

X

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

Short Contents

Table of Contents