# Руководство по GNU Emacs

# Руководство по GNU Emacs

Тринадцатая редакция, обновлено для Emacs версии 20.7

Ричард Столмен

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided also that the sections entitled “The GNU Manifesto”, “Distribution” and “GNU General Public License” are included exactly as in the original, and provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that the sections entitled “The GNU Manifesto”, “Distribution” and “GNU General Public License” may be included in a translation approved by the Free Software Foundation instead of in the original English.

Тринадцатая редакция
обновлено для Emacs версии 20.7, июль 1999

© Free Software Foundation, Inc., 1985-1987, 1993-1999
© Перевод. Н. Ю. Иванова, 1993
© Перевод. А. Я. Отт, О. С. Тихонов, 1999

ISBN 5-85593-133-1

Ричард Столмен
"Руководство по GNU Emacs"

Издано АНО"ИЛКиРЛ"
Лицензия ЛР N 066121 от 22.09.1998г.

Подписано к печати 25.11.1999г.
Формат 70x100 1/16
Тираж 1000 экз. Тип.зак.

ОАО "АСТРА СЕМЬ"
Москва, Филипповский пер.,13

# Предисловие

Это руководство описывает использование и простую настройку редактора Emacs. От читателя не требуется быть программистом; для простой настройки не надо обладать навыками в программировании. Те пользователи, которые не интересуются настройкой, могут пропускать рассыпанные по тексту краткие советы по ней.

Это, главным образом, справочное руководство, но его также можно использовать в качестве учебника. Для тех, кто еще совсем не знаком с Emacs, неплохой идеей будет начать с обучающей диалоговой программы. Чтобы вызвать ее, запустите Emacs и наберите `C-h t`. Тогда вы сможете изучить Emacs, используя его в специально разработанном файле, который описывает команды, говорит, когда их использовать, а затем объясняет полученные с помощью этих команд результаты.

При первом чтении просто бегло просмотрите первую и вторую главы, в которых описана принятая в этом руководстве система обозначений и общий вид экрана в Emacs. Обратите внимание, на какие вопросы есть ответы в этих главах, чтобы вы могли вернуться к ним позже. После прочтения четвертой главы вы должны попрактиковаться с рассмотренными в ней командами. Следующие несколько глав описывают часто используемые фундаментальные методы и понятия. Вам нужно понять их до конца, экспериментируя с ними, если это необходимо.

Главы с четырнадцатой по девятнадцатую описывают средства среднего уровня, полезные в любых видах редактирования. Глава 20 и последующие описывают возможности, которые вы можете захотеть использовать, а можете и не захотеть; причитайте эти главы, когда эти возможности вам понадобятся.

Если вам кажется, что Emacs работает неправильно, прочтите главу о решении проблем. Она объясняет, как справиться с некоторыми часто возникающими трудностями (см. Раздел 32.2 [Неполадки], с. 372), а также когда и как нужно сообщать об ошибках в Emacs (см. Раздел 32.3 [Ошибки в Emacs], с. 375).

Чтобы отыскать документацию на отдельную команду, загляните в именной указатель. Ключи (литеральные команды) и имена команд перечислены в отдельных именных указателях. Также имеется словарь с перекрестными ссылками на каждый термин.

Данное руководство доступно в печатной форме и в виде Info-файла. Info-файл служит для диалогового чтения с программой Info; это будет первичным способом просмотра интерактивной документации в системе GNU. И этот Info-файл, и сама программа Info распространяются вместе с GNU Emacs. Печатная книга и Info-файл содержат по сути один тот же текст и создаются из общего исходного файла, который также распространяется вместе с GNU Emacs.

GNU Emacs — член семейства Emacs-редакторов, насчитывающего большое число редакторов, построенных на общих организационных принципах. Чтобы почитать о философии, лежащей в основе Emacs, и уроках, полученных из его развития, выпишите копию AI memo 519a "Emacs, the Extensible, customizable Self-Documenting Display Editor", в Publication Department, Artificial Intelligence Lab, 545 Tech Square, Cambridge, MA 02139, USA. По последним сообщениям они стоили $2.25 за копию. Другая полезная публикация — это LCS TM-165, "A Cookbook for an Emacs", Craig Finseth, доступная из Publication Department, Laboratory for Computer Seince, 543 Tech Square, Cambridge, MA 02139, USA. Сегодняшняя цена $3.

Эта редакция руководства предназначена для использования с GNU Emacs, установленном в системах GNU и Unix. GNU Emacs также можно использовать в системах VMS, MS-DOS (иначе называемой MS-DOG), Windows NT и Windows 95. Эти системы придерживаются другого синтаксиса имен файлов; кроме того, VMS и MS-DOS не поддерживают всех возможностей GNU Emacs. В этом руководстве мы не пытаемся описать применение Emacs в системе VMS. Для получения информации об использовании Emacs в MS-DOS смотрите Приложение C [MS-DOS], с. 403.

# Распространение

GNU Emacs — *свободная программа*; это значит, что всякий может свободно использовать и повторно распространять его при соблюдении определенных условий. GNU Emacs не является общественным достоянием; на него действуют авторские права, и на его распространение есть ограничения, но эти ограничения разработаны так, чтобы позволить все, что может захотеть хороший благонамеренный гражданин. Что не разрешается, так это пытаться мешать другим и далее совместно пользоваться любой версией GNU Emacs, которую они могли бы от вас получить. Точные условия находятся в Универсальной Общественной Лицензии GNU, которая поставляется с Emacs и также идет в следующем разделе.

Один из способов получить копию GNU Emacs — взять ее кого-то, у кого Emacs уже есть. Вам не нужно просить нашего разрешения либо что-то еще, просто скопируйте ее. Если вы имеете доступ к Internet, вы можете получить последнюю версию дистрибутива GNU Emacs по анонимному FTP; подробности смотрите в файле ‘`etc/FTP`’ в поставке Emacs.

Вы можете также получить GNU Emacs, когда покупаете компьютер. Производители компьютеров могут распространять копии на тех же самых условиях, что и все остальные. Эти условия требуют от них дать вам полные исходные тексты, включая любые изменения, сделанные ими, и разрешить вам распространять полученный от них GNU Emacs согласно обычным условиям Универсальной Общественной Лицензии. Другими словами, эта программа должна быть свободной для вас, когда вы ее получаете, а не только свободной для производителей.

Вы также можете заказать копии GNU Emacs у Фонда Свободного ПО на компакт-диске. Это удобный и надежный способ получить копию; это также хороший способ помочь средствами нашей работе. (Фонд всегда получал большую часть средств таким способом.) Форма заказа включена в файл ‘`etc/ORDERS`’ в поставке Emacs, она есть и на нашем Web-сайте http://www.gnu.org/order/order.html. Для получения дополнительной информации пишите по адресу

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

Доход от платы за распространение идет на поддержку целей фонда: разработки нового свободного программного обеспечения и развития наших существующих программ, включая GNU Emacs.

Если вы находите GNU Emacs полезным, пожалуйста **пришлите взнос** в Фонд Свободного программного обеспечения, чтобы поддержать нашу работу. Пожертвования в Фонд Свободного ПО в США обладают налоговой скидкой. Если вы используете GNU Emacs на работе, пожалуйста, предложите вашей компании сделать взнос. Если политика компании недружелюбно относится к идее пожертвования благотворительной организации, вы можете вместо этого предложить время от времени заказывать у Фонда Свободного ПО компакт-диски или подписаться на регулярные обновления.

В разработке GNU Emacs принимали участие Пер Абрахамсен, Джей К. Адамс, Джо Арсено, Боаз Бен-Цви, Джим Блэнди, Дэвид М. Браун, Теренс М. Бренон, Питер Бретон, Фрэнк Брец, Кевин Броди, Винсент Броман, Нейл В. Ван Дайк, Мортен Велиндер, Ульрик Виет, Майк Вильямс, Родни Витби, Феликс С. Т. Ву, Стивен А. Вуд, Барри Ворсо, Джоан Вроманс, Том Вурглер, Кейт Габриэльски, Кевин Галахер, Кевин Галло, Дуг Гвин, Говард Гейл, Анри Гийом, Дэвид Гиллеспи, Стивен Гильди, Боб Гликстайн, Борис Голдовски, Микеланджело Григни, Майкл Гшвинд, Матье Девин, Майкл ДеКорте, Гари Делп, Кайл Джонс, Майкл К. Джонсон, Эрик Динг, Карстен Доминик, Скотт Дрейвс, Виктор

Духовни, Джеми Завински, Эли Зарецкий, Нил Зиринг, Вильям Зоммерфельд, Ларс Ингебригтсен, Эндрю Иннес, Джон Итон, Томодзи Кагатани, Говард Кайе, Майкл Кайфер, Брюстер Кале, Билл Карпентер, Дуг Катинг, Давид Кауфман, Генри Кауц, Джеф Кёнинг, Ричард Кинг, Джеймс Кларк, Майк Кларксон, Глин Клементс, Дэвид Когедал, Ларри К. Колодни, Роберт Кравиц, Себастиан Кремер, Эндрю Ксилаг, Даниель ЛаЛиберт, Аарон Ларсон, Джеймс Р. Ларус, Фредерик Лепье, Ларс Линдберг, Эрик Лудлам, Роланд МакГрат, Билл Манн, Кен Манхаймер, Брайан Марик, Бенгт Мартенсон, Чарли Мартин, Саймон Маршалл, Дэвид Меггинсон, Томас Мей, Нейл М. Мейгер, Вейн Месард, Ричард Млинарик, Кейт Мур, Эрик Наггум, Юрген Никельсен, Томас Нойман, Джефф Норден, Эндрю Норман, Майк Ньютон, Джефф Пек, Дамон Антон Пермезель, Том Перрин, Йенс Петерсен, Кристиан Плонт, Франческо Поторти, Майкл Д. Прандж, Дэниел Пфайфер, Фред Пьерестегай, Эшвин Рам, Пол Рейли, Эрик С. Реймонд, Эдвард М. Рейнгольд, Роб Рипел, Роланд Б. Робертс, Джон Робинсон, Гилермо Х. Розас, Вильям Розенблат, Денни Розендаль, Ивар Руммелхоф, Вольфганг Рупрехт, Масахико Сато, Мануэль Серрано, Эспен Скоглунд, Рик Сладки, Линн Слейтер, Дэвид Смит, Крис Смит, Пол Д. Смит, Майкл Статс, Аке Стенхофф, Питер Стефенсон, Джонатан Стигельман, Стив Страссман, Джеймс Б. Сэлем, Йенс Т. Бергер Тиэльман, Спенсер Томас, Джим Томпсон, Эд Уилкинсон, Масанобу Умеда, Дейл Р. Уорли, Джозеф Брайан Уэллс, Фредерик Фернбак, Фред Фиш, Карл Фогель, Джефри Фолькер, Гари Фостер, Ной Фридмен, Джон Хайдеман, Кеничи Ханда, К. Шейн Хартман, Маркус Херич, Манабу Хигасида, Карл Хойер, Андерс Холст, Курт Хорник, Том Хулдер, Крис Хэнсон, Рето Циммерман, Ян Т. Циммерман, Станислав Шалунов, Марк Шапиро, Рэндел Шварц, Вильям Шелтер, Ричард Шерман, Олин Шиверс, Ральф Шлайхер, Грегор Шмид, Майкл Шмидт, Рональд С. Шнель, Филипп Шнобелен, Стефан Шоэф, Сема Штейнгольд, Ганс Чалупски, Боб Чассел, Рольф Эберт, Стивен Эглен, Торбьорн Эйнарсон, Цугутомо Энами, Ганс Генрик Эриксен, Майкл Эрнст и Ата Этемади.

# GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright © 1989, 1991 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

## 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 individually 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 interactive 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

## How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> *one line to give the program's name and an idea of what it does.*
> Copyright (C) 19*yy* *name of author*
>
> This program is free software; you can redistribute it and/or
> modify it under the terms of the GNU General Public License
> as published by the Free Software Foundation; either version 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.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright (C) 19*yy* *name of author*
> 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.

The hypothetical commands '`show w`' and '`show c`' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than '`show w`' and '`show c`'; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright
> interest in the program 'Gnomovision'
> (which makes passes at compilers) written
> by James Hacker.
>
> *signature of Ty Coon*, 1 April 1989
> Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

# УНИВЕРСАЛЬНАЯ ОБЩЕСТВЕННАЯ ЛИЦЕНЗИЯ GNU

Версия 2, июнь 1991

Copyright © 1989, 1991 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
© Перевод. О.В. Кузина, В.М. Юфа, 1993
© Перевод. О.С. Тихонов, 1998

Этот документ можно копировать, а также распространять его дословные копии, однако вносить в него изменения запрещено.

## Преамбула

Лицензии на большую часть программного обеспечения (ПО) составлены так, чтобы лишить вас свободы совместно использовать и изменять его. В противоположность этому, предназначение Универсальной Общественной Лицензии GNU состоит в том, чтобы гарантировать вашу свободу совместно использовать и изменять свободное ПО, т.е. обеспечить свободу ПО для всех его пользователей. Данная Универсальная Общественная Лицензия применима к большей части ПО Фонда Свободного ПО и ко всем другим программам, чьи авторы принимают на себя обязательство ее использовать. (Для некоторых программ Фонда Свободного ПО вместо нее применяется Универсальная Общественная Лицензия GNU для библиотек.) Вы тоже можете применить ее к своим программам.

Когда мы говорим о свободном ПО, мы имеем в виду свободу, а не бесплатность. Наши Универсальные Общественные Лицензии разрабатывались для того, чтобы гарантировать, что вы пользуетесь свободой распространять копии свободного ПО (и при желании получать за это вознаграждение); что вы получаете исходный код или можете получить его, если захотите; что вы можете изменять ПО или использовать его части в новых свободных программах; и что вы знаете обо всех этих правах.

Чтобы защитить ваши права, нам нужно ввести некоторые ограничения, которые запретят кому бы то ни было отказывать вам в этих правах или потребовать от вас отказаться от этих прав. Эти ограничения накладывают на вас некоторые обязательства, если вы распространяете копии ПО или изменяете его.

Например, если вы распространяете копии такой программы бесплатно или за вознаграждение, вы должны предоставить получателям все права, которыми обладаете вы сами. Вы должны гарантировать, что они тоже получат или смогут получить исходный код. Наконец, вы должны показать им текст данных условий, чтобы они знали о своих правах.

Мы защищаем ваши права в два этапа: (1) сохраняем авторские права на ПО и (2) предлагаем вам эту лицензию, которая дает вам законное право копировать, распространять и/или модифицировать ПО.

Кроме того, в целях защиты как каждого автора, так и нас, мы хотим удостовериться, что каждый понимает, что гарантий на это свободное ПО нет. Если ПО модифицируется и передается кем-то еще, мы хотим, чтобы получатели ПО знали, что то, что у них есть, — это не оригинал, чтобы любые проблемы, созданные другими, не отразились на репутации первоначальных авторов.

И наконец, каждой свободной программе постоянно угрожают патенты на ПО. Мы хотим избежать той опасности, что повторные распространители свободной программы самостоятельно получат патенты, делая программу таким образом частной собственностью. Чтобы предотвратить это, мы со всей определенностью заявляем, что любой патент должен быть либо предоставлен всем для свободного использования, либо не предоставлен никому.

Ниже следуют точные определения и условия для копирования, распространения и модификации.

## ОПРЕДЕЛЕНИЯ И УСЛОВИЯ ДЛЯ КОПИРОВАНИЯ, РАСПРОСТРАНЕНИЯ И МОДИФИКАЦИИ

0. Эта Лицензия применима к любой программе или другому произведению, содержащему уведомление, помещенное держателем авторских прав и сообщающее о том, что оно может распространяться при условиях, оговоренных в данной Универсальной Общественной Лицензии. В дальнейшем термин “Программа” относится к любой такой программе или произведению, а термин “произведение, основанное на Программе” означает Программу или любое произведение, содержащее Программу или ее часть, дословную, или модифицированную, и/или переведенную на другой язык. (Здесь и далее перевод включается без ограничений в понятие “модификация”.) Каждый обладатель лицензии адресуется как “вы”.

   Виды деятельности, не являющиеся копированием, распространением или модификацией, не охватываются данной Лицензией; они лежат за пределами ее влияния. Использование Программы по ее функциональному назначению не ограничено, а выходные данные Программы охватываются этой Лицензией, только если их содержание является произведением, основанным на Программе (вне зависимости от того, были ли они получены в процессе использования Программы). Являются ли они таковыми, зависит от того, что именно делает Программа.

1. Вы можете копировать и распространять дословные копии исходного кода Программы по его получении на любом носителе, при условии что вы соответствующим образом помещаете на видном месте в каждой копии соответствующее уведомление об авторских правах и отказ от предоставления гарантий; оставляете нетронутыми все уведомления, относящиеся к данной Лицензии и к отсутствию каких-либо гарантий; и передаете всем другим получателям Программы копию данной Лицензии вместе с Программой.

   Вы можете назначить плату за физический акт передачи копии и можете по своему усмотрению предоставлять гарантии за вознаграждение.

2. Вы можете изменять свою копию или копии Программы или любой ее части, создавая таким образом произведение, основанное на Программе, и копировать и распространять эти модификации или произведение в соответствии с Разделом 1, приведенным выше, при условии, что вы выполните все нижеследующие условия:

   a. Вы обязаны снабдить модифицированные файлы заметными уведомлениями, содержащими указания на то, что вы изменили файлы, и дату каждого изменения.

   b. Вы обязаны предоставить всем третьим лицам лицензию на бесплатное использование каждого произведения, которое вы распространяете или публикуете, целиком, и которое полностью или частично содержит Программу или какую-либо ее часть, на условиях, оговоренных в данной Лицензии.

   c. Если модифицированная программа обычно читает команды в интерактивном режиме работы, вы должны сделать так, чтобы при запуске для работы в таком интерактивном режиме обычным для нее способом она печатала или выводила на экран объявление, содержащее соответствующее уведомление об авторских правах и уведомление о том, что гарантий нет (или, наоборот, сообщающее о том, что вы обеспечиваете гарантии), и что пользователи могут повторно распространять программу при этих условиях, и указывающее пользователю, как просмотреть копию данной Лицензии. (Исключение: если сама Программа работает в интерактивном режиме, но обычно не выводит подобных сообщений, то ваше произведение, основанное на Программе, не обязано выводить объявление.)

Эти требования применяются к модифицированному произведению в целом. Если известные части этого произведения не были основаны на Программе и могут обоснованно считаться независимыми и самостоятельными произведениями, то эта Лицензия и ее условия не распространяются на эти части, если вы распространяете их как отдельные произведения. Но если вы распространяете эти части как часть целого произведения, основанного на Программе, то вы обязаны делать это в соответствии с условиями данной Лицензии, распространяя права получателей лицензии на все произведение и, таким образом, на каждую часть, вне зависимости от того, кто ее написал.

Таким образом, содержание этого раздела не имеет цели претендовать на ваши права на произведение, написанное полностью вами, или оспаривать их; цель скорее в том, чтобы реализовать право управлять распространением производных или коллективных произведений, основанных на Программе.

Кроме того, простое нахождение другого произведения, не основанного на этой Программе, совместно с Программой (или с произведением, основанным на этой Программе) на одном носителе для постоянного хранения или распространяемом носителе не распространяет действие этой Лицензии на другое произведение.

3. Вы можете копировать и распространять Программу (или произведение, основанное на ней) согласно Разделу 2) в объектном коде или в выполнимом виде в соответствии с Разделами 1 и 2, приведенными выше, при условии, что вы также выполните одно из следующих требований:
   a. Сопроводите ее полным соответствующим машиночитаемым исходным кодом, который должен распространяться в соответствии с Разделами 1 и 2, приведенными выше, на носителе, который обычно используется для обмена ПО; или,
   b. Сопроводите ее письменным предложением, действительным по крайней мере в течение трех лет, предоставить любому третьему лицу за вознаграждение, не превышающее стоимость физического акта изготовления копии, полную машиночитаемую копию соответствующего исходного кода, подлежащую распространению в соответствии с Разделами 1 и 2, приведенными выше; или
   c. Сопроводите ее информацией, полученной вами в качестве предложения распространить соответствующий исходный код. (Эта возможность допустима только для некоммерческого распространения, и только если вы получили программу в объектном коде или в исполняемом виде с предложением в соответствии с Пунктом b) выше.)

   Исходный код для произведения означает его вид, предпочтительный для выполнения в нем модификаций. Для исполняемого произведения полный исходный код означает все исходные коды для всех модулей, которые он содержит, плюс любые связанные с произведением файлы определения интерфейса, плюс сценарии, используемые для управления компиляцией и установкой исполняемого произведения. Однако, в виде особого исключения распространяемый исходный код не обязан включать то, что обычно предоставляется (как в объектных, так и в исходных кодах) с основными компонентами (компилятор, ядро и так далее) операционной системы, под управлением которой работает исполняемое произведение, за исключением случая, когда сам компонент сопровождает исполняемое произведение.

   Если распространение исполняемого произведения или объектного кода происходит путем предоставления доступа для копирования с обозначенного места, то предоставление доступа для копирования исходного кода с того же места считается распространением исходного кода, даже если третьи лица не принуждаются к копированию исходного кода вместе с объектным кодом.

4. Вы не можете копировать, изменять, повторно лицензировать, или распространять Программу никаким иным способом, кроме явно предусмотренных данной Лицензией. Любая попытка копировать, изменять или распространять Программу каким-либо

другим способом или с измененной лицензией неправомерна и автоматически прекращает ваши права, данные вам этой Лицензией. Однако лицензии лиц, получивших от вас копии или права согласно данной Универсальной Общественной Лицензии, не прекращают своего действия, если эти лица полностью соблюдают условия.

5. Вы не обязаны соглашаться с этой Лицензией, так как вы не подписывали ее. Однако, ничто, кроме этой Лицензии, не дает вам право изменять или распространять эту Программу или основанные на ней произведения. Эти действия запрещены законом, если вы не принимаете к соблюдению эту Лицензию. А значит, изменяя или распространяя Программу (или произведение, основанное на Программе), вы изъявляете свое согласие с этой Лицензией и всеми ее условиями о копировании, распространении или модификации Программы или основанных на ней произведений.

6. Каждый раз, когда вы повторно распространяете Программу (или любое произведение, основанное на Программе), получатель этого произведения автоматически получает от первоначального выдавшего лицензию лица свою лицензию на копирование, распространение или модификацию Программы, обсуждаемую в этих определениях и условиях. Вы не можете налагать каких-либо дополнительных ограничений на осуществление получателем прав, предоставленных данным документом. Вы не несете ответственности за соблюдение третьими лицами условий этой Лицензии.

7. Если в результате судебного разбирательства, или обвинения в нарушении патента или по любой другой причине (не обязательно связанной с патентами), вам навязаны условия, противоречащие данной Лицензии (по постановлению суда, по соглашению или иным способом), это не освобождает вас от соблюдения Лицензии. Если вы не можете заниматься распространением так, чтобы одновременно удовлетворить требованиям и этой Лицензии, и всем другим требованиям, то вы не должны заниматься распространением Программы. Например, если патент не позволяет безвозмездное повторное распространение Программы всем, кто получил копии от вас непосредственно или через посредников, то единственным способом удовлетворить и патенту, и этой Лицензии будет ваш полный отказ от распространения Программы.

   Если какая-либо часть этого раздела не имеет силы или не может быть исполнена при некоторых конкретных обстоятельствах, то подразумевается, что имеет силу остальная часть раздела, а при других обстоятельствах имеет силу весь Раздел.

   Цель этого раздела — не побудить вас делать заявления о нарушениях прав на патент, или заявлять о других претензиях на право собственности или оспаривать правильность подобных претензий; единственная цель этого раздела — защита целостности системы распространения свободного ПО, которая реализуется использованием общественных лицензий. Многие люди внесли щедрый вклад в широкий спектр ПО, распространяемого по этой системе, полагаясь на ее согласованное применение; только автору принадлежит право решать, хочет ли он или она распространять ПО в этой системе или в какой-то другой, и получатель лицензии не может влиять на принятие этого решения.

   Этот раздел предназначен для того, чтобы тщательно прояснить, что полагается следствием из остальной части данной Лицензии.

8. Если распространение и/или применение Программы ограничено в ряде стран либо патентами, либо авторскими правами на интерфейсы, первоначальный обладатель авторских прав, выпускающий Программу с этой Лицензией, может добавить явное ограничение на географическое распространение, исключив такие страны, так что распространение разрешается только в тех странах, которые не были исключены. В этом случае данная Лицензия включает в себя это ограничение, как если бы оно было написано в тексте данной Лицензии.

9. Фонд Свободного ПО может время от времени публиковать пересмотренные и/или новые версии Универсальной Общественной Лицензии. Такие новые версии будут

сходны по духу с настоящей версией, но могут отличаться в деталях, направленных на новые проблемы или обстоятельства.

Каждой версии придается отличительный номер. Если в Программе указывается, что к ней относится некоторый номер версии данной Лицензии и “любая последующая версия”, вы можете по выбору следовать определениям и условиям либо данной версии, либо любой последующей версии, опубликованной Фондом Свободного ПО. Если в Программе не указан номер версии данной Лицензии, вы можете выбрать любую версию, когда-либо опубликованную Фондом Свободного ПО.

10. Если вы хотите встроить части Программы в другие свободные программы с иными условиями распространения, напишите автору с просьбой о разрешении. Для ПО, которое охраняется авторскими правами Фонда Свободного ПО, напишите в Фонд Свободного ПО; мы иногда делаем такие исключения. Наше решение будет руководствоваться двумя целями: сохранения свободного статуса всех производных нашего свободного ПО и содействия совместному и повторному использованию ПО вообще.

## НИКАКИХ ГАРАНТИЙ

11. ПОСКОЛЬКУ ПРОГРАММА ПРЕДОСТАВЛЯЕТСЯ БЕСПЛАТНО, НА ПРОГРАММУ НЕТ ГАРАНТИЙ В ТОЙ МЕРЕ, КАКАЯ ДОПУСТИМА ПРИМЕНИМЫМ ЗАКОНОМ. ЗА ИСКЛЮЧЕНИЕМ ТЕХ СЛУЧАЕВ, КОГДА ОБРАТНОЕ ЗАЯВЛЕНО В ПИСЬМЕННОЙ ФОРМЕ, ДЕРЖАТЕЛИ АВТОРСКИХ ПРАВ И/ИЛИ ДРУГИЕ СТОРОНЫ ПОСТАВЛЯЮТ ПРОГРАММУ "КАК ОНА ЕСТЬ" БЕЗ КАКОГО-ЛИБО ВИДА ГАРАНТИЙ, ВЫРАЖЕННЫХ ЯВНО ИЛИ ПОДРАЗУМЕВАЕМЫХ, ВКЛЮЧАЯ, НО НЕ ОГРАНИЧИВАЯСЬ ИМИ, ПОДРАЗУМЕВАЕМЫЕ ГАРАНТИИ КОММЕРЧЕСКОЙ ЦЕННОСТИ И ПРИГОДНОСТИ ДЛЯ КОНКРЕТНОЙ ЦЕЛИ. ВЕСЬ РИСК В ОТНОШЕНИИ КАЧЕСТВА И ПРОИЗВОДИТЕЛЬНОСТИ ПРОГРАММЫ ОСТАЕТСЯ ПРИ ВАС. ЕСЛИ ПРОГРАММА ОКАЖЕТСЯ ДЕФЕКТНОЙ, ВЫ ПРИНИМАЕТЕ НА СЕБЯ СТОИМОСТЬ ВСЕГО НЕОБХОДИМОГО ОБСЛУЖИВАНИЯ, ВОССТАНОВЛЕНИЯ ИЛИ ИСПРАВЛЕНИЯ.

12. НИ В КОЕМ СЛУЧАЕ, ЕСЛИ НЕ ТРЕБУЕТСЯ СООТВЕТСТВУЮЩИМ ЗАКОНОМ, ИЛИ НЕ УСЛОВЛЕНО В ПИСЬМЕННОЙ ФОРМЕ, НИ ОДИН ДЕРЖАТЕЛЬ АВТОРСКИХ ПРАВ И НИ ОДНО ДРУГОЕ ЛИЦО, КОТОРОЕ МОЖЕТ ИЗМЕНЯТЬ И/ИЛИ ПОВТОРНО РАСПРОСТРАНЯТЬ ПРОГРАММУ, КАК БЫЛО РАЗРЕШЕНО ВЫШЕ, НЕ ОТВЕТСТВЕННЫ ПЕРЕД ВАМИ ЗА УБЫТКИ, ВКЛЮЧАЯ ЛЮБЫЕ ОБЩИЕ, СПЕЦИАЛЬНЫЕ, СЛУЧАЙНЫЕ ИЛИ ПОСЛЕДОВАВШИЕ УБЫТКИ, ПРОИСТЕКАЮЩИЕ ИЗ ИСПОЛЬЗОВАНИЯ ИЛИ НЕВОЗМОЖНОСТИ ИСПОЛЬЗОВАНИЯ ПРОГРАММЫ (ВКЛЮЧАЯ, НО НЕ ОГРАНИЧИВАЯСЬ ПОТЕРЕЙ ДАННЫХ, ИЛИ ДАННЫМИ, СТАВШИМИ НЕПРАВИЛЬНЫМИ, ИЛИ ПОТЕРЯМИ, ПОНЕСЕННЫМИ ИЗ-ЗА ВАС ИЛИ ТРЕТЬИХ ЛИЦ, ИЛИ ОТКАЗОМ ПРОГРАММЫ РАБОТАТЬ СОВМЕСТНО С ЛЮБЫМИ ДРУГИМИ ПРОГРАММАМИ), ДАЖЕ ЕСЛИ ТАКОЙ ДЕРЖАТЕЛЬ ИЛИ ДРУГОЕ ЛИЦО БЫЛИ ИЗВЕЩЕНЫ О ВОЗМОЖНОСТИ ТАКИХ УБЫТКОВ.

## КОНЕЦ ОПРЕДЕЛЕНИЙ И УСЛОВИЙ

## Как применять эти условия к вашим новым программам

Если вы разрабатываете новую программу и хотите, чтобы она принесла максимально возможную пользу обществу, лучший способ достичь этого — включить ее в свободное ПО, которое каждый может повторно распространять и изменять согласно данным условиям.

Чтобы сделать это, добавьте в программу следующие уведомления. Надежнее всего будет добавить их в начало каждого исходного файла, чтобы наиболее эффективно передать сообщение об отсутствии гарантий; каждый файл должен содержать по меньшей мере строку, содержащую знак охраны авторского права и указание на то, где находится полное уведомление.

> *одна строка, содержащая название программы и краткое*
> *описание того, что она делает.*
> (C) *наименование (имя) автора* 19*гг*
>
> Это свободная программа; вы можете повторно распространять ее и/или
> модифицировать ее в соответствии с Универсальной Общественной
> Лицензией GNU, опубликованной Фондом Свободного ПО; либо версии 2,
> либо (по вашему выбору) любой более поздней версии.
>
> Эта программа распространяется в надежде, что она будет полезной,
> но БЕЗ КАКИХ-ЛИБО ГАРАНТИЙ; даже без подразумеваемых гарантий
> КОММЕРЧЕСКОЙ ЦЕННОСТИ или ПРИГОДНОСТИ ДЛЯ КОНКРЕТНОЙ ЦЕЛИ. Для
> получения подробных сведений смотрите Универсальную Общественную
> Лицензию GNU.
>
> Вы должны были получить копию Универсальной Общественной Лицензии
> GNU вместе с этой программой; если нет, напишите по адресу: Free
> Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
> 02111-1307 USA

Добавьте также сведения о том, как связаться с вами по электронной и обычной почте.

Если программа интерактивная, сделайте так, чтобы при запуске в интерактивном режиме она выдавала краткое уведомление вроде следующего:

> Гномовизор, версия 69, (C) *имя автора* 19*гг*
> Гномовизор поставляется АБСОЛЮТНО БЕЗ КАКИХ-ЛИБО ГАРАНТИЙ;
> для получения подробностей напечатайте 'show w'. Это свободная
> программа, и вы приглашаетесь повторно распространять ее при
> определенных условиях; для получения подробностей введите 'show c'.

Гипотетические команды ‘show w’ и ‘show c’ должны показывать соответствующие части Универсальной Общественной Лицензии. Конечно, используемые вами команды могут называться как-нибудь иначе, нежели ‘show w’ и ‘show c’; они даже могут выбираться с помощью мыши или быть пунктами меню — как больше подходит для вашей программы.

Вы также должны добиться того, чтобы ваш работодатель (если вы работаете программистом) или ваше учебное заведение, если таковое имеется, подписали в случае необходимости “отказ от имущественных прав” на эту программу. Вот образец; замените фамилии:

> Компания ■Братья Ёдины■ настоящим отказывается от всех
> имущественных прав на программу 'Гномовизор' (которая делает
> пассы в сторону компиляторов), написанную Абстрактным К.И.
>
> *подпись Мага Ната*, 1 апреля 1989 г
> Маг Нат, Президент фирмы Вице.

Данная Универсальная Общественная Лицензия не позволяет вам включать вашу программу в программы, являющиеся частной собственностью. Если ваша программа — это библиотека процедур, вам стоит подумать, не будет ли лучше разрешить программам, являющимся частной собственностью, связываться с вашей библиотекой. Если это то, чего вы хотите, используйте вместо этой Лицензии Универсальную Общественную Лицензию GNU для библиотек.

# Введение

Вы читаете о GNU Emacs, GNU-инкарнации развитого, самодокументированного, настраиваемого, расширяемого экранного редактора реального времени Emacs. (‘G’ в ‘GNU’ читается.)

Мы говорим, что Emacs — это *экранный* редактор, так как редактируемый текст обычно виден на экране и автоматически обновляется, когда вы вводите команды. См. Глава 1 [Экран], с. 23.

Мы называем его редактором реального времени, так как экран обновляется очень часто, обычно после каждого набранного вами знака или пары знаков. Это уменьшает количество информации, которую вы должны удерживать в памяти, когда редактируете. См. Глава 4 [Основы редактирования], с. 35.

Мы называем Emacs развитым, так как он предоставляет средства, которые дают нечто большее, чем простая вставка или удаление: управление подпроцессами, автоматическое создание отступов в программах, просмотр двух или более файлов сразу, редактирование форматированного текста и действия в терминах символов, слов, строк, предложений, абзацев и страниц, а также выражений и комментариев в нескольких разных языках программирования.

*Самодокументированность* означает, что в любое время вы можете набрать специальный символ, `Control-h`, чтобы узнать, что вы можете сделать. Вы также можете использовать его, чтобы выяснить, что делает произвольная команда, или найти все команды, имеющие отношение к какой-либо теме. См. Глава 7 [Справка], с. 55.

*Настраиваемость* означает, что вы можете легко изменить определения команд Emacs. Например, если вы используете язык программирования, в котором комментарии начинаются с ‘`<**`’ и кончаются на ‘`**>`’, вы можете велеть командам Emacs для работы с комментариями использовать именно такие строки (см. Раздел 22.7 [Комментарии], с. 219). Другой вид настройки — это перерасположение установленных команд. Например, если вы предпочитаете, чтобы четыре основные команды движения курсора (вверх, вниз, вправо, влево) находились на клавишах, расположенных крестообразно на клавиатуре, вы можете перепривязать эти клавиши. См. Глава 31 [Настройка], с. 341.

*Расширяемость* означает, что вы можете пойти дальше простой настройки и написать совершенно новые команды, программы на языке Лисп, предназначенные для запуска в собственном Лисп-интерпретаторе Emacs. Emacs является “интерактивно расширяемой” системой, что значит, что он делится на много функций, которые вызывают друг друга; каждая из них может быть переопределена во время сеанса редактирования. Почти любая часть Emacs может быть заменена без копирования всего Emacs. Большинство команд редактирования Emacs уже написаны на Лиспе; несколько исключений могли бы быть написаны на Лиспе, но написаны на Си для эффективности. Хотя только программист может написать расширение, любой может использовать его после. Если вы хотите научиться программированию на Emacs Lisp, мы рекомендуем вам книгу *Introduction to Emacs Lisp*, написанную Робертом Дж. Часселом, также опубликованную Фондом Свободного Программного Обеспечения.

Когда Emacs запущен под системой X Windows, он предоставляет собственные меню и удобные привязки для кнопок мыши. Но Emacs может дать многие преимущества оконной системы и на текстовом терминале. Например, вы можете видеть одновременно несколько файлов, перемещать текст между файлами и редактировать во время работы команд оболочки.

# 1 Организация экрана

На текстовом терминале Emacs занимает весь экран. В системе X Windows Emacs создает для себя свои X-окна. Мы используем термин *фрейм* для обозначения всего текстового экрана или всего X-окна, используемого Emacs. Оба вида фреймов используются в Emacs одинаково для отображения вашего текста. Обычно Emacs запускается только с одним фреймом, но при желании вы можете создать дополнительные. См. Глава 17 [Фреймы], с. 147.

Когда вы запускаете Emacs, весь фрейм, за исключением первой и последней строки, отводится для редактируемого текста. Эта область называется *окном*. Первая строка называется *полоской меню*, а последняя — это особая *эхо-область* или *окно минибуфера*, где появляются запросы, и куда вы можете ввести ответы. Дальнейшие сведения об этих специальных строках смотрите ниже.

Текстовое окно может быть поделено по горизонтали или вертикали на несколько других окон, каждое из которых может быть использовано для отдельного файла (см. Глава 16 [Окна], с. 141). В данном руководстве слово “окно” всегда обозначает часть фрейма внутри Emacs.

То окно, в котором находится курсор, называется *выбранным окном*; в нем идет редактирование. Большинство команд Emacs неявно относятся в тексту в выбранном окне (хотя команды мыши как правило действуют в том окне, в котором вы щелкнули, вне зависимости от того, выбрано оно или нет). Остальные окна существуют просто для справки, пока вы не выберете одно из них. Если вы используете несколько фреймов в системе X Windows, то передача фокуса ввода некоторому фрейму выбирает окно в этом фрейме.

Последняя строка каждого текстового окна называется *строкой режима*, которая описывает происходящее в этом окне. Она выводится в инверсном виде, если терминал поддерживает это, и содержит текст, который начинается при старте Emacs со строки ‘`-:-  *scratch*`’. Строка режима показывает сведения о текущем состоянии, например, какой буфер показан в окне над ней, какие основные и второстепенные режимы в нем используются, и содержит ли этот буфер несохраненные изменения.

## 1.1 Точка

Курсор терминала внутри Emacs показывает позицию, в которой будет выполняться команда редактирования. Это позиция именуется *точкой*. Многие команды Emacs перемещают точку по тексту, так что вы имеете возможность редактировать в любом месте. Вы также можете установить точку, щелкнув первой кнопкой мыши.

Когда курсор указывает *на* знак, то следует думать, что точка находится *между* двумя знаками; она расположена *перед* знаком, над которым появляется курсор. Например, если ваш текст выглядит как ‘`frob`’, а курсор находится над ‘`b`’, то точка расположена между ‘`o`’ и ‘`b`’. Если вы вставите в этой позиции знак ‘`!`’, то получите ‘`fro!b`’, с точкой между ‘`!`’ и ‘`b`’. Таким образом, курсор останется над ‘`b`’, как и раньше.

Иногда говорят “курсор”, имея в виду точку, или говорят о командах движения точки как о командах “движения курсора”.

Терминалы имеют только один курсор, и когда происходит процесс вывода, курсор должен находиться там, где идет набор. Это не означает, что точка перемещается. Это значит лишь то, что Emacs может показать вам позицию точки, только когда терминал не работает.

Если вы редактируете в Emacs несколько файлов, каждый в своем буфере, то всякий буфер имеет свою собственную позицию точки. Буфер, который не показывается в данное время, запоминает, где находится точка, на случай, если вы снова выведете его на экран.

Когда фрейм разбит на несколько окон, каждое из них имеет свою собственную позицию точки. Курсор же показывает позицию точки в выбранном окне. Этим способом вы также можете узнать, которое из окон выбрано. Если один и тот же буфер появляется более чем в одном окне, то каждое из них имеет свою позицию точки.

Если есть несколько фреймов, каждый из них может отобразить один курсор. Курсор в выбранном фрейме заполнен цветом; курсор в других фреймах — это пустая клетка, которая появляется в том окне, которое было бы выбранным, если бы вы передали фокус ввода этому фрейму.

Термин ‘точка’ происходит от названия знака ‘.’, который был командой TECO (язык, на котором был написан первоначальный Emacs) для получения значения величины, называемой теперь ‘точкой’.

## 1.2 Эхо-область

Строка внизу фрейма (под строкой режима) называется *эхо-областью*. Она используется для показа небольших фрагментов текста в нескольких целях.

*Эхо* обозначает отображение набираемых вами знаков. Вне Emacs, операционная система обычно отображает весь ваш ввод. Emacs управляет эхо иначе.

Emacs никогда не повторяет команды, состоящие из одиночного знака, а команды, состоящие из нескольких, повторяются, только если вы останавливаетесь в процессе их набора. Как только вы останавливаетесь более чем на секунду в середине команды, все знаки этой команды сразу же отображаются. Это служит *подсказкой* для окончания команды. Как только включается эхо, окончание команды отображается немедленно в процессе набора. Такое поведение предназначено для того, чтобы уверенный пользователь получал быстрый результат, в то же время это позволяет неуверенным пользователям получить максимум обратной связи. Вы можете изменить это поведение, установив особую переменную (см. Раздел 11.7 [Переменные изображения], с. 84).

Если команда не может быть выполнена, она в печатает эхо-области *сообщение об ошибке*. Сообщение об ошибке сопровождается звуковым сигналом или миганием экрана. Кроме того, когда произошла ошибка, любой набранный перед этим ввод сбрасывается.

Некоторые команды печатают в эхо-области информационные сообщения. Они похожи на сообщения об ошибках, но без звукового сигнала, и набранная ранее информация не сбрасывается. Иногда эти сообщения говорят вам, что команда выполнена, в случае, если это явно не наблюдается при просмотре редактируемого текста. Иногда единственной целью команды является печать сообщения, дающего вам специфическую информацию. Например, команда `C-x =` используется, чтобы напечатать сообщение, описывающее позицию точки в тексте и ее текущий столбец в окне. Команды, требующие для своего выполнения длительное время, часто выводят во время работы сообщения, заканчивающиеся на ‘...’, а в конце, когда они закончились, слово ‘`done`’.

Информативные сообщения из эхо-области сохраняются в буфере, называемом ‘`*Messages*`’. (Мы еще не объяснили, что такое буфер; для получения большей информации о них смотрите Глава 15 [Буферы], с. 135.) Если вы пропустили сообщение, которое недолго отображалось на экране, вы можете переключиться в буфер ‘`*Messages*`’ и снова посмотреть его. (Последовательные сообщения о промежуточных результатах часто сворачиваются в этом буфере в одно.)

Размер буфера ‘`*Messages*`’ ограничен определенным числом строк, это число задает переменная `message-log-max`. Как только буфер дорастает до этого размера, каждая новая строка удаляет одну строку из начала. См. Раздел 31.2 [Переменные], с. 343, чтобы узнать, как установить переменные, такие как `message-log-max`.

Эхо-область используется также для отображения *минибуфера*: окна, которое используется для считывания аргументов для команд, например, имени файла для редактирования. Когда используется минибуфер, эхо-область начинается со строки подсказки, которая

обычно кончается двоеточием; кроме того, в этой строке появляется курсор, так как она становится выбранным окном. Вы всегда можете выйти из минибуфера, набрав `C-g`. См. Глава 5 [Минибуфер], с. 45.

## 1.3 Строка режима

Последняя строка в каждом окне называется *строкой режима*, она описывает происходящее в этом окне. Когда имеется только одно текстовое окно, строка режима находится непосредственно над эхо-областью; это предпоследняя строка фрейма. Строка режима показывается в инверсном изображении, если терминал поддерживает это, и начинается и кончается дефисами.

Обычно строка режима выглядит так:

*-ск:из буф (основной второстепенные)-стр-поз----*

Это дает информацию о буфере, отображенном в окне: его имя, используемые основной и второстепенный режимы, изменялся ли текст буфера, и как далеко вниз по буферу вы сейчас находитесь.

*из* содержит две звездочки '`**`', если текст в буфере редактировался (буфер "изменен"), или '`-`', если буфер не редактировался. Для буферов, доступных только для чтения, это '`%*`', если буфер изменен, и '`%%`' в противном случае.

*буф* — это имя *буфера* данного окна. Чаще всего оно совпадает с именем файла, который вы редактируете. См. Глава 15 [Буферы], с. 135.

Буфер, находящийся в выбранном окне (в том окне, в котором находится курсор), является также выбранным буфером Emacs — тем буфером, где происходит редактирование. Когда мы говорим, что некоторые команды выполняются "для буфера", мы говорим о текущем выбранном буфере.

*стр* — это '`L`', за которой следует текущей номер строки, где находится точка. Этот номер выводится, когда включен режим Line Number (обычно это так). Вы можете по желанию показать также и номер текущего столбца, включив режим Column Number (он не включается по умолчанию, потому что работает несколько медленнее). См. Раздел 11.5 [Возможности строки режима], с. 83.

*поз* говорит вам, есть ли еще текст сверху от начала либо ниже конца окна. Если буфер небольшой и умещается в окне целиком, то *поз* имеет значение '`All`'. Иначе он имеет значение '`Top`', если вы смотрите начало файла, '`Bot`', если конец, либо '*nn*`%`', где *nn* означает количество процентов файла, находящееся над верхом окна.

*основной* — имя *основного режима*, действующего в этом буфере. В любой момент каждый буфер находится в одном и только одном из возможных основных режимов. Существующие основные режимы включают режим Fundamental (наименее специализированный), режим Text, режим Lisp, режим C, режим Texinfo и многие другие. См. Глава 19 [Основные режимы], с. 175, чтобы подробнее узнать о том, чем режимы отличаются, и как выбрать один из них.

Некоторые основные режимы показывают после имени дополнительные сведения. Например, буферы Rmail показывают номер текущего сообщения и общее число сообщений. Буферы Compilation и Shell отображают состояние подпроцесса.

*второстепенные* — это список некоторых *второстепенных режимов*, которые включены в данный момент в буфере выбранного окна. Например, '`Fill`' значит, что включен режим Auto fill. '`Abbrev`' означает, что включен режим Word Abbrev. '`Ovwrt`' означает, что включен режим Overwrite. См. Раздел 31.1 [Второстепенные режимы], с. 341, для более подробной информации. '`Narrow`' означает, что область редактирования отображаемого буфера ограничена только частью его текста. На самом деле это не второстепенный режим, но действует похоже. См. Раздел 30.8 [Сужение], с. 335. '`Def`' означает, что сейчас определяется макрос клавиатуры. См. Раздел 31.3 [Макросы клавиатуры], с. 353.

Кроме того, если в данный момент Emacs находится внутри уровня рекурсивного редактирования, то внутри круглых скобок, которые окружают имена режимов, появляются квадратные скобки (‘[...]’). Если Emacs находится на некотором уровне рекурсивного редактирования внутри другого, то появляются двойные квадратные скобки, и так далее. Так как уровень рекурсивного редактирования относится ко всему Emacs, а не к каждому буферу в отдельности, то квадратные скобки появляются или в строках режима всех окон или ни в одной из них. См. Раздел 30.12 [Рекурсивное редактирование], с. 338.

Неоконные терминалы могут показать только один фрейм Emacs в одно время (см. Глава 17 [Фреймы], с. 147). На таких терминалах строка режима отображает имя выбранного фрейма, после *из*. Начальный фрейм называется ‘F1’.

*ск* обозначает систему кодирования, которая используется в редактируемом файле. Дефис указывает на состояние, устанавливаемое по умолчанию: отсутствие преобразования, за исключением преобразования конца строки, если содержимое файла требует этого. ‘=’ обозначает полное отсутствие преобразований. Не столь простые преобразования представляются различными буквами — например, ‘1’ относится к ISO Latin-1. См. Раздел 18.7 [Системы кодирования], с. 165, для подробной информации. Если вы используете метод ввода, к началу *ск* добавляется строка в форме ‘*в*>’; *в* идентифицирует метод ввода. (Некоторые методы ввода показывают вместо ‘>’ знак ‘+’ или ‘@’.) См. Раздел 18.4 [Методы ввода], с. 163.

Когда вы пользуетесь текстовым терминалом (не оконной системой), *ск* использует три знака для описания, соответственно, системы кодирования для ввода с клавиатуры, системы кодирования для терминального вывода и системы кодирования, применяемой в редактируемом файле.

Когда выключена поддержка многобайтных знаков, *ск* не появляется совсем. См. Раздел 18.2 [Включение многобайтных знаков], с. 161.

Двоеточие после *ск* может меняться в некоторых обстоятельствах на другой знак. Emacs использует для разделения строк в буфере перевод строки. Некоторые файлы используют другое соглашение о разделении строк: возврат каретки и перевод строки (соглашение MS-DOS) или просто возврат каретки (соглашение Macintosh). Если файл в буфере использует возврат каретки и перевод строки, двоеточие заменяется на обратную косую черту (‘\’) или ‘(DOS)’, в зависимости от вашей операционной системы. Если в этом файле применяется просто возврат каретки, то двоеточие заменяется на косую черту (‘/’) или ‘(Mac)’. На некоторых системах Emacs позывает вместо двоеточия строку ‘(Unix)’, даже для файлов, использующих для разделения строк перевод строки.

Вы можете настроить способ отображения в строке режима каждого из форматов разделения строк, устанавливая переменные `eol-mnemonic-unix`, `eol-mnemonic-dos`, `eol-mnemonic-mac` и `eol-mnemonic-undecided` равными строкам, которые вы находите подходящими. См. Раздел 31.2 [Переменные], с. 343, где объясняется, как устанавливать переменные.

См. Раздел 11.5 [Возможности строки режима], с. 83, подробности о возможностях добавления в строку режима другой полезной информации, например номера текущего столбца в точке, текущего времени и сообщения о приходе почты.

## 1.4 Полоска меню

Каждый фрейм Emacs обычно имеет сверху *полоску меню*, которую вы можете использовать для произведения определенных действий. Нет необходимости перечислять их здесь, потому что вам проще посмотреть самим.

Когда вы используете оконную систему, вы можете выбирать команды из полоски меню при помощи мыши. Направленная вправо стрелка после пункта меню указывает, что этот пункт ведет ко вторичному меню; ‘...’ в конце означает, что до начала работы эта команда считает аргументы с клавиатуры.

Чтобы просмотреть полное имя команды и документацию к пункту меню, напечатайте `C-h k` и затем выберите нужный пункт с помощью мыши обычным способом. (см. Раздел 7.1 [Справка о ключах], с. 56).

На текстовых терминалах, где нет мыши, вы можете использовать меню, напечатав `M-‘` или ⟨F10⟩ (они запускают команду `tmm-menubar`). Эта команда входит в режим, где вы можете выбрать пункт меню с помощью клавиатуры. Предварительный выбор показывается в эхо-области. Вы можете использовать правую и левую курсорные стрелки для движения по меню к разным пунктам. Когда вы нашли нужный вам пункт, напечатайте ⟨RET⟩, чтобы выбрать его.

К каждому пункту меню также приписывается буква или цифра; обычно это первая первая буква какого-то слова из имени пункта. Эта буква или цифра отделяется от имени значком ‘=>’. Для выбора пункта вы можете напечатать его букву или цифру.

Некоторые команды в меню имеют также обычные привязки к ключу; в таком случае после самого пункта перечисляется один эквивалентный ключ в круглых скобках.

# 2 Знаки, ключи и команды

Эта глава рассказывает о наборах знаков, используемых в Emacs для ввода команд и внутри файлов, а также объясняет концепции *ключей* и *команд*, которые необходимы для понимания того, как Emacs воспринимает ввод с клавиатуры и мыши.

## 2.1 Виды пользовательского ввода

GNU Emacs использует для ввода с клавиатуры расширение набора знаков ASCII; он также воспринимает незнаковые события, включая функциональные клавиши и действия с клавишами мыши.

ASCII состоит из 128 знаковых кодов. Некоторым из этих кодов приписаны графические обозначения, такие как 'a' и '='; остальные являются управляющими знаками, например `Control-a` (также именуется `C-a` для краткости). `C-a` получил свое имя из-за того, что вы набираете его, зажав клавишу ⟨CTRL⟩ и затем нажав `a`.

У некоторых управляющих знаков ASCII есть специальные названия, и большинство терминалов имеют специальные клавиши, с помощью которых их можно набрать, например: ⟨RET⟩, ⟨TAB⟩, ⟨DEL⟩ и ⟨ESC⟩. Знак пробела обычно обозначается ниже как ⟨SPC⟩, хотя строго говоря, он является графическим знаком, чье изображение должно быть пустым. На некоторых клавиатурах есть клавиша "linefeed"; это другое название для `C-j`.

Emacs расширяет множество знаков ASCII тысячами других печатных знаков (см. Глава 18 [MULE], с. 161), дополнительными управляющими знаками и несколькими модификаторами, которые можно комбинировать с любым другим знаком.

На ASCII-терминалах существует только 32 управляющих знака. Это варианты с модификатором control для букв и знаков '`@[]\^_`'. Кроме того, клавиша shift не имеет смысла с управляющими знаками: `C-a` и `C-A` — это одно и то же, и Emacs не может их различить.

Но во множестве знаков Emacs есть место для вариантов с control для всех печатных знаков и для различения между `C-a` и `C-A`. X Windows позволяет ввести все эти знаки. Например, `C-` (то есть Control-минус) и `C-5` — допустимые команды Emacs в системе X.

Еще одно расширение множества знаков Emacs — это дополнительные биты модификаторов. Чаще всего используется только один бит-модификатор, он называется Meta. Для любого знака есть его Meta-вариант; примеры включают `Meta-a` (для краткости `M-a`), `M-A` (это не то же самое, что `M-a`, но эти два знака имеют обычно одно и то же значение в Emacs), `M-`⟨RET⟩ и `M-C-a`. По традиции мы обычно пишем `C-M-a`, а не `M-C-a`; но логично говоря, порядок набора клавиш ⟨CTRL⟩ и ⟨META⟩ не имеет значения.

Некоторые терминалы имеют клавишу ⟨META⟩ и позволяют набирать Meta-знаки, нажав эту клавишу. Таким образом, `Meta-a` набирается путем одновременного нажатия клавиш ⟨META⟩ и `a`. Клавиша ⟨META⟩ работает очень похоже на ⟨SHIFT⟩. Подобные клавиши не всегда метятся ⟨META⟩, так как эти функции часто присваиваются специальными клавишам с каким-то другим первоначальным значением.

Если у вас нет клавиши ⟨META⟩, вы все же можете набрать Meta-знаки, используя последовательность двух знаков, начинающуюся с ⟨ESC⟩. Таким образом, чтобы ввести `M-a`, вы можете набрать ⟨ESC⟩ `a`. Чтобы ввести `C-M-a`, вы должны набрать ⟨ESC⟩ `C-a`. ⟨ESC⟩ разрешена также и на терминалах с клавишами ⟨META⟩, на случай, если вы привыкли использовать ее.

X Windows предоставляет несколько других клавиш-модификаторов, которые могут быть применены к любому вводимому знаку. Эти клавиши называются ⟨SUPER⟩, ⟨HYPER⟩ и ⟨ALT⟩. Мы пишем '`s-`', '`H-`' и '`A-`', чтобы сказать, что знак использует эти модификаторы. Таким образом, `s-H-C-x` — это сокращение для `Super-Hyper-Control-x`. Не все X-терминалы в действительности предоставляют клавиши для этих модификаторов — фактически многие терминалы имеют клавишу, помеченную ⟨ALT⟩, которая на самом деле

является клавишей ⟨META⟩. Стандартные привязки ключей Emacs не содержат каких-либо знаков с этими модификаторами. Но вы можете придать им свои значения, настраивая Emacs.

Ввод с клавиатуры включает клавиши, не являющие знаками: например функциональные клавиши и курсорные стрелки. Кнопки мыши также выпадают из гаммы знаков. Вы можете модифицировать эти события с помощью клавиш-модификаторов ⟨CTRL⟩, ⟨META⟩, ⟨SUPER⟩, ⟨HYPER⟩ и ⟨ALT⟩ точно так же, как и знаки клавиатуры.

Вводимые знаки и незнаковый ввод вместе называются *событиями*. См. раздел “Input Events” в *The Emacs Lisp Reference Manual*, для дальнейшей информации. Если вы не программируете на Лиспе, а просто хотите переопределить значения некоторых знаков и незнаковых событий, смотрите Глава 31 [Настройка], с. 341.

ASCII-терминалы в действительности не могут посылать ничего, кроме знаков ASCII. Такие терминалы используют для представления каждой функциональной клавиши последовательность знаков. Но это невидимо для пользователя Emacs, потому что процедуры обработки ввода с клавиатуры распознают эти особые последовательности и преобразуют их в события функциональных клавиш до того, как любые другие части Emacs могут их увидеть.

## 2.2 Ключи

*Последовательность ключей*, (или коротко, *ключ*) есть последовательность вводимых событий, которая воспринимается как целое, как “единая команда”. Некоторые командные последовательности Emacs — это просто одиночный знак или одно событие; например, просто `C-f` достаточно для перемещения вперед на один знак. Но в Emacs также есть команды, которые вызываются двумя или более событиями.

Если последовательность событий достаточна для вызова команды, она является *законченным ключом*. Примером законченных ключей могут быть `C-a`, `X`, ⟨RET⟩, ⟨NEXT⟩ (функциональная клавиша), ⟨DOWN⟩ (стрелка), `C-x C-f` и `C-x 4 C-f`. Если последовательность недостаточна, чтобы составить законченный ключ, она называется *префиксным ключом*. В примерах выше `C-x` и `C-x 4` являются префиксными ключами. Любая последовательность ключей — это либо законченный, либо префиксный ключ.

Большинство одиночных знаков являются законченными ключами в стандартных привязках команд Emacs. Немногие из них являются префиксными ключами. Префиксный ключ объединяется со следующим событием в более длинную последовательность ключей, которая сама может быть законченной или префиксной. Например, `C-x` — это префиксный ключ, поэтому `C-x` и следующее событие объединяются в двухзнаковую последовательность ключей. Большинство этих последовательностей являются законченными ключами, в том числе `C-x C-f` и `C-x b`. Некоторые, например `C-x 4` и `C-x r`, сами являются префиксными ключами, приводящими к трехзнаковым последовательностям ключей. Нет ограничения на длину последовательности ключей, но на практике люди редко используют последовательности длиннее четырех событий.

Напротив, добавить события к законченному ключу вы не можете. Например, двухзнаковая последовательность `C-f C-k` не является ключом, так как `C-f` является законченным ключом сама по себе. Невозможно дать `C-f C-k` независимое значение как команде. `C-f C-k` — это две команды, а не одна.

Из всего сказанного следует, что префиксными ключами в Emacs являются `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` и `M-j`. Но этот список не встроен; это просто стандартная привязка ключей в Emacs. В процессе настройки Emacs вы можете сделать новые префиксные ключи или уничтожить эти. См. Раздел 31.4 [Привязки ключей], с. 356.

Если вы в самом деле создаете или удаляете префиксные ключи, это изменяет набор возможных последовательностей ключей. Например, если вы переопределяете `C-f` как

префикс, `C-f C-k` автоматически становятся ключом (законченным, если вы не определите его тоже как префикс). Наоборот, если вы уничтожите определение префикса `C-x 4`, то `C-x 4 f` (или `C-x 4` *что-нибудь*) не будет в дальнейшем ключом.

Напечатав знак вызова справки (`C-h` или ⟨F1⟩) после префиксного знака, вы получите список команд, начинающихся с этого префикса. Есть несколько префиксных знаков, для которых `C-h` не работает — по историческим причинам у них есть другое значение для `C-h`, которое непросто изменить. Но ⟨F1⟩ должна работать для всех префиксных знаков.

## 2.3 Ключи и команды

В этом руководстве много отрывков, которые говорят вам, что делают отдельные ключи. Но Emacs не определяет значение ключей непосредственно. Вместо этого Emacs присваивает значение именованным *командам* и затем придает ключам значения путем *привязки* их к командам.

Каждая команда имеет имя, выбранное для нее программистом. Обычно это имя составляется из нескольких английских слов, разделенных дефисами; например, `next-line` или `forward-word`. У команды также есть *определение*, являющееся программой на Лиспе; она заставляет функцию делать то, что она делает. На самом деле, команда в языке Emacs Lisp — это особая разновидность лисповской функции; это такая функция, которая указывает, как читать аргументы при интерактивном вызове. Для получения большей информации о командах и функциях смотрите раздел “What Is a Function” в *The Emacs Lisp Reference Manual*. (Определение, которое мы использовали в данном руководстве, слегка упрощено.)

Связь между ключами и функциями записывается в различных таблицах, называемых *таблицами ключей*. См. Раздел 31.4.1 [Таблицы ключей], с. 356.

Когда мы говорим, что “`C-n` сдвигает вертикально вниз на одну строку”, мы замалчиваем отличие, которое не относится к делу при обычном использовании, но существенно для понимания того, как настраивается Emacs. Это команда `next-line`, которая запрограммирована на сдвиг вниз. Ключ `C-n` имеет такое действие, *так как* он привязан к этой команде. Если вы перепривяжете `C-n` к команде `forward-word`, то он будет перемещать на одно слово вперед. Перепривязка клавиш — обычный метод настройки.

В остальной части этого руководства мы обычно будем игнорировать эту тонкость, чтобы сохранить простоту. Чтобы дать необходимую для настройки информацию, мы ставим имя команды, которая в действительности выполняет работу, в круглых скобках после ссылки на ключ, который ее запускает. Например, мы будем говорить: “Команда `C-n` (`next-line`) передвигает точку вниз”, имея в виду, что `next-line` — это команда, которая двигает вниз, а `C-n` — это ключ, который стандартно привязан к ней.

Раз уж мы говорим сейчас только об информации для настройки, то это подходящий момент, чтобы рассказать вам о *переменных*. Часто описание команд будет говорить: “Чтобы изменить это, установите переменную `mumble-foo`”. Переменная — это имя, используемое для запоминания значения. Многие описанные в этом руководстве переменные существуют просто для облегчения настройки: некоторая команда или другая часть Emacs проверяет переменную и ведет себя по-разному в зависимости от ее значения. Пока вы не интересуетесь настройкой, вы можете пропустить информацию о переменных. Когда вы будете готовы, чтобы заинтересоваться, прочитайте базовые сведения о переменных, и тогда информация об отдельных переменных будет иметь смысл. См. Раздел 31.2 [Переменные], с. 343.

## 2.4 Наборы знаков для текста

Текст в буферах Emacs — это последовательность восьмибитных байт. Каждый байт может содержать один знак ASCII. Допустимы как управляющие знаки ASCII (с восьме-

ричными кодами от 000 до 037, и 0177), так и печатные ASCII-знаки (с кодами от 040 до 0176); однако, управляющие знаки не из ASCII не могут появиться в буфере. Другие флаги-модификаторы, используемые при вводе с клавиатуры, такие как Meta, также недопустимы в буферах.

Некоторые управляющие знаки ASCII служат в тексте для особых целей и имеют особые названия. Например, знак новой строки (восьмеричный код 012) используется в буфере для завершения строки, а символ табуляции (восьмеричный код 011) используется для создания отступа до следующей позиции табуляции (обычно через каждые 8 столбцов). См. Раздел 11.6 [Отображение текста], с. 84.

Печатные знаки, не входящие в ASCII, также могут появляться в буферах. Когда включены многобайтные знаки, вы можете использовать любые печатные не-ASCII-знаки, которые Emacs поддерживает. Они имеют коды, начинающиеся от 256, или восьмеричного 0400, и каждый из них представляется как последовательность двух или более байт. См. Глава 18 [MULE], с. 161.

Если вы выключите поддержку многобайтных знаков, то сможете использовать только один алфавит не-ASCII-знаков, каждый из которых вмещается в один байт. Для них используются коды от 0200 до 0377. См. Раздел 18.12 [Однобайтные европейские знаки], с. 172.

# 3 Вход и выход из Emacs

Обычно для вызова Emacs достаточно просто набрать '`emacs`' в оболочке. Emacs очищает экран и отображает начальные справочные сведения и уведомление об авторских правах. Некоторые операционные системы сбрасывают все набранное перед тем, как Emacs стартует; они не дают Emacs возможности предотвратить это. Поэтому рекомендуется подождать, пока Emacs очистит экран, и только потом набрать вашу первую команду редактирования.

Если вы запускаете Emacs из окна с оболочкой в системе X Windows, запускайте его в фоновом режиме с помощью '`emacs&`'. Тогда Emacs не свяжет окно оболочки, и вы сможете выполнять другие команды, пока Emacs работает в своих X-окнах. Вы можете начинать печатать команды, как только направите ввод с клавиатуры во фрейм Emacs.

Когда Emacs начинает работу, он создает буфер, называемый '`*scratch*`'. Это буфер, который предоставляется вам первоначально. Буфер '`*scratch*`' использует режим Lisp Interaction; вы можете набирать в нем Лисп-выражения и вычислять их, либо вы можете проигнорировать такую возможность и просто писать в нем заметки. (Вы можете задать для этого буфера другой основной режим, установив в вашем файле инициализации переменную `initial-major-mode`. См. Раздел 31.7 [Файл инициализации], с. 366.)

Из аргументов в командной строке оболочки можно указать файлы, к которым вы хотите обратиться, Лисп-файлы для загрузки и функции, которые будут вызваны. См. Приложение A [Аргументы командной строки], с. 385. Но мы не рекомендуем так делать. Эта возможность существует преимущественно для совместимости с другими редакторами.

Многие редакторы спроектированы так, что запускаются снова каждый раз, когда вы хотите редактировать. Вы редактируете один файл и затем выходите из редактора. В следующий раз, когда вы хотите редактировать другой файл или тот же самый, вы должны запустить редактор снова. С такими редакторами имеет смысл использовать аргумент командной строки, чтобы сообщить, какой файл будет редактироваться.

Но не имеет смысла запускать новый Emacs каждый раз, когда вы хотите редактировать другой файл. С одной стороны, это было бы раздражающе медленно. С другой стороны, при этом не использовалась бы способность Emacs обращаться к нескольким файлам за один сеанс редактирования. И при этом терялись бы накопленные сведения о контексте: регистры, история отмены изменений, список пометок и другие.

Рекомендуемый способ использования GNU Emacs — запускать его только один раз сразу после входа в систему и делать все ваши редакции в одном и том же процессе Emacs. Каждый раз, когда вы хотите редактировать другой файл, вы вызываете его в уже существующий Emacs, который в конце концов предназначен, чтобы хранить много файлов, готовых для редактирования. Обычно вы не уничтожаете Emacs до тех пор, пока не решите выйти из системы. См. Глава 14 [Файлы], с. 105, для получения информации о редактировании нескольких файлов одновременно.

## 3.1 Выход из Emacs

Существует две команды выхода из Emacs, так как существует два вида выхода: *приостановка* Emacs и *уничтожение* Emacs.

*Приостановка* означает временную остановку Emacs и возврат управления его родительскому процессу (обычно это оболочка), она позволяет вам позднее возобновить редактирование в том же задании Emacs, с теми же буферами, с тем же списком уничтожений, с той же историей отмены и так далее. Это обычный способ выхода.

*Уничтожение* Emacs подразумевает уничтожение задания Emacs. Вы снова можете запустить Emacs позднее, но вы получите новый Emacs. Нет способа возобновить тот же сеанс редактирования после его уничтожения.

`C-z` Приостановить Emacs (`suspend-emacs`) или минимизировать текущий фрейм (`iconify-or-deiconify-frame`).

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

Чтобы приостановить Emacs, наберите `C-z` (`suspend-emacs`). Это вернет вас обратно в оболочку, из которой вы вызывали Emacs. Вы можете возобновить Emacs с помощью команды '`%emacs`' в большинстве обычных оболочек.

В системах, которые не позволяют программам приостанавливаться, `C-z` запускает подчиненную оболочку, которая связана прямо с терминалом, и Emacs ждет, пока вы не покинете подоболочку. (Скорее всего, вы можете сделать это с помощью `C-d` или '`exit`', но это зависит от используемой оболочки.) В этих системах единственный способ вернуться в оболочку, из которой Emacs был запущен (чтобы выйти из системы, например), — это уничтожить Emacs.

Приостановка также невозможна, если вы запустили Emacs из оболочки, не поддерживающей приостановку программ, даже если система ее поддеривает. В таком случае вы можете установить переменную `cannot-suspend` в отличное от `nil` значение, чтобы принудить `C-z` запускать подчиненную оболочку. (Можно было бы назвать родительскую оболочку Emacs тоже "подчиненной" за неумение правильно управлять заданиями, но это вопрос вкуса.)

Когда Emacs напрямую общается с X-сервером и создает собственные выделенные X-окна, `C-z` имеет другой смысл. Приостановка приложения, использующего собственные X-окна, бессмысленна и бесполезна. Вместо этого, `C-z` запускает команду `iconify-or-deiconify-frame`, которая временно закрывает выбранный фрейм Emacs (см. Глава 17 [Фреймы], с. 147). Снова получить доступ к этому фрейму можно с помощью программы управления окнами.

Чтобы уничтожить Emacs, наберите `C-x C-c` (`save-buffer-kill-emacs`). Двухзнаковый ключ используется для того, чтобы его было труднее набрать. Эта команда сначала предлагает сохранить все модифицированные обращающиеся к файлам буферы. Если вы не сохраняете их все, она запрашивает новое подтверждение с помощью `yes` перед уничтожением Emacs, так как любые незаписанные изменения будут потеряны навсегда. Также, если остаются запущенными какие-либо подпроцессы, то `C-x C-c` запрашивает подтверждение и для них, так как уничтожение Emacs немедленно уничтожит подпроцессы.

Невозможно перезапустить сеанс Emacs, когда вы его уже уничтожили. Однако, вы можете указать Emacs записывать определенные сведения о сеансе, например, к каким файлам обращались, когда Emacs был уничтожен; тогда в следующий раз, когда вы перезапустите Emacs, он попытается обратиться к тем же файлам. См. Раздел 30.11 [Сохранение сеансов Emacs], с. 337.

Операционная система обычно улавливает определенные специальные символы, которые могут мгновенно приостанавливать или уничтожать программу. **Это свойство операционной системы отключается в то время, когда вы находитесь в Emacs.** Значения `C-z` и `C-c` как ключей в Emacs были вдохновлены стандартными значениями `C-z` и `C-c` в некоторых операционных системах как символов останова и уничтожения программы, но это единственная их связь с этими операционными системами. Вы можете настроить эти ключи для запуска любых команд по вашему выбору (см. Раздел 31.4.1 [Таблицы ключей], с. 356).

# 4 Основные команды редактирования

Здесь мы рассказываем основы о том, как ввести текст, сделать изменения и записать текст в файл. Если эти сведения новы для вас, вы можете изучить это более легким способом, запустив самоучитель по Emacs. Чтобы сделать так, запустите Emacs и наберите `Control-h t` (`help-with-tutorial`).

Чтобы очистить экран и перерисовать изображение, напечатайте `C-l` (`recenter`).

## 4.1 Вставка текста

Чтобы вставить печатные знаки в редактируемый вами текст, достаточно просто набрать их. Таким образом, вводимые вами знаки вставляются в буфер в позиции курсора (то есть в *точке*; см. Раздел 1.1 [Точка], с. 23). Курсор двигается вперед, и весь текст после курсора тоже сдвигается вперед. Если в буфере был текст ‘`FOOBAR`’, и курсор стоял перед ‘`B`’, то если вы наберете `XX`, вы получите ‘`FOOXXBAR`’ с курсором, оставшимся перед ‘`B`’.

Чтобы *удалить* текст, который вы только что вставили, используйте ⟨DEL⟩. ⟨DEL⟩ удаляет знак *перед* курсором (а не знак под курсором; этот знак стоит *после* курсора). Курсор и все знаки, стоящие после него, сдвигаются назад. Поэтому, если вы набрали печатные знаки и затем набираете ⟨DEL⟩, набранное отменяется.

Чтобы закончить строку и начать набирать новую, нажмите ⟨RET⟩. Это вставит в буфер знак перехода на новую строку. Если точка находится в середине строки, ⟨RET⟩ разбивает эту строку. Набор ⟨DEL⟩, когда курсор находится в начале строки, удаляет предшествующий символ новой строки, соединяя таким образом эту и предшествующую строки.

Emacs может разбивать строки автоматически, когда они становятся слишком длинными, если вы включите специальный второстепенный режим, называемый режимом *Auto Fill*. См. Раздел 21.5 [Заполнение], с. 185, для информации об использовании режима Auto Fill.

Если вы предпочитаете, чтобы знаки текста замещали (перезаписывали) существующий текст, а не сдвигали его вправо, вы можете включить второстепенный режим Overwrite. См. Раздел 31.1 [Второстепенные режимы], с. 341.

Непосредственная вставка работает для печатных знаков и ⟨SPC⟩, но другие знаки действуют как команды редактирования и не вставляют сами себя. Если вам требуется вставить управляющий знак или знак, код которого превышает восьмиричное 200, вы должны *отменить их особый смысл*, набрав перед ними `Control-q` (`quoted-insert`). (Название этого знака обычно записывается как `C-q` для краткости.) Существует два способа использования `C-q`:

- `C-q`, за которым следует любой неграфический знак (даже `C-g`), вставляет этот знак.
- `C-q`, за которым идет последовательность восьмиричных цифр, вставляет знак с заданным знаковым кодом. Вы можете использовать любое число восьмиричных цифр; любой знак, не являющийся цифрой, обрывает последовательность. Если завершающий знак — это ⟨RET⟩, он служит только для завершения последовательности; любой другой нецифровой знак сам используется в качестве ввода после завершения последовательности. (Использование восьмиричных последовательностей запрещено в обычном недвоичном режиме Overwrite, чтобы дать вам удобный способ вставить цифру вместо замещения ей.)

Когда включена поддержка многобайтных знаков, восьмиричные коды от 0200 до 0377 не являются верными знаками; если вы задаете код из этого промежутка, `C-q` считает, что вы собираетесь использовать какой-то из наборов знаков ISO Latin-*n*, и преобразует заданный код к соответствующему коду знаков Emacs. См. Раздел 18.2 [Включение многобайтных

знаков], с. 161. *Какой именно* набор знаков ISO Latin следует использовать, определяется выбранной вами языковой средой (см. Раздел 18.3 [Языковые среды], с. 162).

Чтобы использовать вместо восьмиричных цифр десятичные или шестнадцатиричные, установите переменную `read-quoted-char-radix` равной 10 или 16. Если основание больше десяти, некоторые буквы, начиная с `a`, служат частью знакового кода, так же, как и цифры.

Числовой аргумент для `C-q` указывает, сколько копий знаков с отмененным особым смыслом необходимо вставить (см. Раздел 4.10 [Аргументы], с. 42).

Информация для настройки: ⟨DEL⟩ в большинстве режимов запускает команду `delete-backward-char`; ⟨RET⟩ запускает команду `newline`, а самовставляющиеся печатные знаки запускают команду `self-insert`, которая вставляет тот знак, который был набран при ее вызове. Некоторые основные режимы переприязывают ⟨DEL⟩ к другим командам.

## 4.2 Изменение положения точки

Чтобы сделать что-то большее, чем просто вставка знаков, вы должны знать, как двигается точка (см. Раздел 1.1 [Точка], с. 23). Простейший способ переместить точку — воспользоваться курсорными стрелками или щелкнуть левой кнопкой мыши в том месте, куда вы хотите передвинуть точку.

Есть также Control- и Meta-знаки для перемещения курсора. Некоторые из них эквивалентны курсорным стрелкам (они были придуманы в те дни, когда у терминалов еще не было курсорных стрелок, их можно использовать на таких терминалах). Другие делают более сложные вещи.

`C-a` Передвинуться в начало строки (`beginning-of-line`).

`C-e` Передвинуться в конец строки (`end-of-line`).

`C-f` Сдвинуться на один знак вперед (`forward-char`).

`C-b` Сдвинуться на один знак назад (`backward-char`).

`M-f` Сдвинуться на одно слово вперед (`forward-word`).

`M-b` Сдвинуться на одно слово назад (`backward-word`).

`C-n` Сдвинуться вертикально вниз на одну строку (`next-line`). Эта команда старается оставить горизонтальную позицию неизменной, чтобы если вначале вы находились в середине одной строки, то в конце оказались бы в середине следующей. В случае, если вы находитесь на последней строке текста, `C-n` создает новую строку и передвигается на нее.

`C-p` Сдвинуться вертикально вверх на одну строку (`previous-line`).

`M-r` Сдвинуть точку к левому краю на строку в середине окна (`move-to-window-line`). Текст при этом не перемещается по экрану.

Числовой аргумент говорит, на какой строке экрана поместить точку. Он отсчитывает экранные строки от верха окна (нуль для самой верхней). Отрицательный аргумент отсчитывает строки снизу ($-1$ для нижней строки).

`M-<` Перейти на начало буфера (`beginning-of-buffer`). При аргументе, равном $n$, двигает на $n/10$ от начала. См. Раздел 4.10 [Аргументы], с. 42, для более подробной информации о числовых аргументах.

`M->` Перейти в конец буфера (`end-of-buffer`).

`M-x goto-char`
Считывает число $n$ и сдвигает точку к позиции $n$ в буфере. Позиция 1 — это начало буфера.

`M-x goto-line`
: Считывает число *n* и сдвигает точку к строке с номером *n*. Строка 1 — это начало буфера.

`C-x C-n`
: Велит использовать текущий столбец, в котором находится точка, в качестве *полупостоянного целевого столбца* для `C-n` и `C-p` (`set-goal-column`). В дальнейшем эти команды всегда переходят на этот столбец в каждой строке, к которой вы передвигаетесь, или как можно ближе к нему при данном содержимом строки. Этот целевой столбец остается в силе, пока его не отменят.

`C-u C-x C-n`
: Отменить целевой столбец. В дальнейшем `C-n` и `C-p` снова, как обычно, пытаются избежать изменения горизонтальной позиции.

Если вы установите переменную `track-eol` не равной `nil`, то если точка находится к конце строки, `C-n` и `C-p` передвигают в конец другой строки. Обычно `track-eol` равна `nil`. См. Раздел 31.2 [Переменные], с. 343, чтобы узнать, как установить переменные вроде `track-eol`.

Обычно `C-n`, вызванная на последней строке буфера, добавляет к буферу новую строку. Если переменная `next-line-add-newlines` равна `nil`, то `C-n` вместо этого выдает ошибку (как `C-p` на первой строке).

## 4.3 Стирание текста

⟨DEL⟩
: Удалить знак перед точкой (`delete-backward-char`).

`C-d`
: Удалить знак после точки (`delete-char`).

`C-k`
: Уничтожить все до конца строки (`kill-line`).

`M-d`
: Уничтожить все знаки вперед до конца следующего слова (`kill-word`).

`M-`⟨DEL⟩
: Уничтожить все знаки в обратном направлении вплоть до начала предыдущего слова (`backward-kill-word`).

Вы уже знаете о клавише ⟨DEL⟩, которая удаляет знак перед точкой (то есть перед курсором). Другой ключ, `Control-d` (`C-d` для краткости), удаляет знак после точки (тот знак, на котором курсор). Это сдвигает остальной текст на строке влево. Если вы нажмете `C-d` в конце строки, то эта строка и следующая за ней соединяются.

Чтобы стереть больший кусок текста, используйте ключ `C-k`, который удаляет целую строку. Если вы нажмете `C-k` в начале или в середине строки, то он уничтожает весь текст вплоть до конца этой строки. Если вы наберете `C-k` в конце строки, то он объединяет эту строку со следующей.

См. Раздел 9.1 [Уничтожение], с. 69, для получения информации о более гибких способах уничтожения текста.

## 4.4 Отмена сделанных изменений

Вы можете удалить все недавние изменения в тексте буфера вплоть до определенного предела. Каждый буфер записывает изменения отдельно, и команда отмены всегда относится к текущему буферу. Обычно каждая команда редактирования создает отдельное вхождение в записи отмены, но некоторые команды, как например `query-replace`, создают несколько вхождений, а очень простые команды, например самовставляющиеся знаки, часто объединяются, чтобы сделать процесс отмены менее утомительным.

`C-x u`
: Отменить одну группу изменений — обычно одну стоящую команду (`undo`).

`C-_` То же самое.

`C-u C-x u` Отменить одну группу изменений в области.

Вы делаете отмену с помощью команды `C-x u` или `C-_`. Когда вы впервые даете эту команду, она отменяет последнее сделанное изменение. Точка возвращается к тому месту, где она была до команды, сделавшей изменение.

Последовательное повторение `C-_` или `C-x u` отменяет все более ранние изменения, вплоть до предела доступной информации отмены. Если все записанные изменения уже были отменены, команда отмены печатает сообщение об ошибке и ничего не делает.

Любая команда, отличная от команды отмены, прерывает последовательность команд отмены. Начиная с этого момента предыдущие команды отмены рассматриваются как простые изменения, которые могут быть отменены. Таким образом, чтобы вернуть отмененные вами изменения, наберите `C-f` или любую другую команду, которая не причиняя вреда прервет последовательность отмен, а затем снова набирайте команды отмены.

Обычная отмена относится ко всем изменениям, сделанным в текущем буфере. Вы также можете произвести *выборочную отмену*, ограниченную текущей областью. Чтобы сделать это, задайте желаемую область, а затем запустите команду `undo` с префиксным аргументом (значение не играет роли): `C-u C-x u` или `C-u C-_`. Это отменяет самое последнее изменение в области. Чтобы отменить более ранние изменения, повторите команду `undo` (префиксный аргумент не требуется). В режиме Transient Mark, любое использование `undo`, когда есть активная область, производит выборочную отмену; вам не нужно задавать префиксный аргумент.

Если вы заметили, что буфер был изменен случайно, простейший путь вернуться в первоначальное состояние — это набирать `C-_` несколько раз, пока не исчезнут звездочки, стоящие в начале строки режима. В этот момент все сделанные вами изменения отменены. Всякий раз, когда команды отмены убирают звездочки из строки режима, это означает, что содержимое буфера стало точно таким же, каким оно было, когда файл был последний раз считан или сохранен.

Если вы не помните, намеренно ли вы изменили буфер, наберите `C-_` один раз. Когда вы увидите последнее изменение, которое вы сделали отмененным, вы поймете, было ли это изменение умышленным. Если это была случайность, оставьте его отмененным. Если оно было умышленным, восстановите изменение, как описано выше.

Не все буферы записывают информацию для отмены. Буферы, чьи имена начинаются с пробела, этого не делают. Они используются для внутренних нужд Emacs и его расширений для хранения текста, который обычно не редактируется и не просматривается пользователями.

Вы не можете отменить простое перемещение курсора; только изменения содержимого буфера сохраняют информацию для отмены. Однако, некоторые команды движения курсора устанавливают метку, так что если вы время от времени используете эти команды, вы можете вернуться в окрестности, по которым вы передвигались, с помощью списка пометок (см. Раздел 8.5 [Список пометок], с. 66).

Когда информация отмены для какого-то буфера становится слишком большой, Emacs время от времени (во время сборки мусора) сбрасывает самые старые сведения. Вы можете указать, сколько информации для отмены необходимо хранить, с помощью двух переменных: `undo-limit` и `undo-strong-limit`. Их значения выражаются в байтах.

Переменная `undo-limit` устанавливает гибкий предел: Emacs хранит данные для отмены достаточного числа команд, чтобы оставаться в этих границах и, возможно, выйти за них, но не хранит данные для более ранних команд сверх этого предела. Ее значение по умолчанию равно 20000. Переменная `undo-strong-limit` устанавливает более жесткий предел: команда, которая увеличивает размер за эту границу, забывается сама. Значение этой переменной равно по умолчанию 30000.

Вне зависимости от значений этих переменных, самое последнее изменение никогда не сбрасывается, поэтому нет опасности, что сборка мусора, случившаяся сразу после неумышленного большого изменения, помешает вам отменить его.

Причина того, что для команды `undo` определены два запускающих ключа, `C-x u` и `C-_`, состоит в том, что эта команда стоит того, чтобы иметь однознаковый ключ, но на некоторых клавиатурах нет очевидного способа ввести `C-_`. `C-x u` предоставляет альтернативу, вы можете прямо набрать его на любом терминале.

## 4.5 Файлы

Описанных выше команд достаточно для создания и изменения текста в буфере Emacs; более продвинутые команды Emacs лишь позволяют делать это проще. Но для того чтобы сохранить любой текст, вы должны поместить его в *файл*. Файлы — это именованные единицы текста, которые хранятся операционной системой, чтобы вы могли получить их позже по имени. Чтобы просмотреть или использовать содержимое файла с любой целью, включая редактирование с помощью Emacs, вы должны задать его имя.

Рассмотрим файл с именем '`/usr/rms/foo.c`'. Для того чтобы начать редактирование этого файла в Emacs, наберите

```
C-x C-f /usr/rms/foo.c ⟨RET⟩
```

Здесь имя файла дается как *аргумент* для команды `C-x C-f` (`find-file`). Эта команда использует для считывания аргумента *минибуфер*, а чтобы завершить аргумент, вы набираете ⟨RET⟩ (см. Глава 5 [Минибуфер], с. 45).

Emacs подчиняется этой команде, *обращаясь* к указанному файлу: создавая буфер, копируя в него содержимое этого файла и затем показывая вам буфер для редактирования. Если вы изменили этот буфер, вы можете *сохранить* новый текст в файле, напечатав `C-x C-s` (`save-buffer`). Это делает изменения постоянными путем копирования измененного содержимого буфера снова в файл '`/usr/rms/foo.c`'. До этого ваши изменения существуют только внутри Emacs, а файл '`foo.c`' не изменяется.

Чтобы создать файл, надо просто обратиться к нему с помощью `C-x C-f`, как если бы он уже существовал. Emacs создаст пустой буфер, куда вы можете вставить текст, который хотите занести в файл. Реальный файл создается, когда вы сохраняете этот буфер с помощью `C-x C-s`.

Конечно, об использовании файлов можно узнать гораздо больше. См. Глава 14 [Файлы], с. 105.

## 4.6 Справка

Если вы забыли, что делает какая-то клавиша, вы можете выяснить это с помощью знака справки `C-h` (или ⟨F1⟩, аналога `C-h`). Наберите `C-h k`, а затем имя ключа, о котором вы хотите узнать; например, `C-h k C-n` рассказывает все о том, что делает `C-n`. `C-h` — это префиксная клавиша; `C-h k` — просто одна из ее подкоманд (а именно, команда `describe-key`). Другие подкоманды `C-h` предоставляют другие виды справки. Наберите `C-h` дважды, чтобы получить описание всех возможностей справки. См. Глава 7 [Справка], с. 55.

## 4.7 Пустые строки

Здесь рассматриваются специальные команды и приемы создания и удаления пустых строк.

`C-o` Вставить после курсора одну или несколько пустых строк (`open-line`).

`C-x C-o` Уничтожить все последовательные пустые строки, кроме одной (`delete-blank-lines`).

Когда вы хотите вставить новую строку текста перед уже существующей, вы можете сделать это, набрав новую строку текста и за ней ⟨RET⟩. Однако, может оказаться удобнее видеть то, что вы набираете, если перед этим сперва создать пустую строку, а затем вставить в нее нужный текст. Это легко сделать, используя клавишу `C-o` (`open-line`), которая вставляет новую строку после точки, но оставляет точку перед этой новой строкой. После `C-o` набирайте текст в новую строку. `C-o F O O` дает тот же результат, что и `F O O` ⟨RET⟩, за исключением конечной позиции точки.

Вы можете создать несколько пустых строк, набрав `C-o` несколько раз, или задав ей числовой аргумент, чтобы сказать ей, какое количество строк необходимо создать. См. Раздел 4.10 [Аргументы], с. 42, чтобы узать, как это сделать. Если определен префикс заполнения, то команда `C-o`, когда вы используете ее в начале строки, вставляет на новой строке этот префикс заполнения. См. Раздел 21.5.3 [Префикс заполнения], с. 187.

Простой способ избавиться от лишних пустых строк — воспользоваться командой `C-x C-o` (`delete-blank-lines`). Использование `C-x C-o` среди нескольких пустых строк удаляет их все, кроме одной. `C-x C-o` на одиночной пустой строке удаляет эту строку. Когда точка находится на непустой строке, `C-x C-o` удаляет все пустые строки после этой непустой строки.

## 4.8 Строки продолжения

Если вы добавляете слишком много знаков в одну строку, не прерывая ее нажатием ⟨RET⟩, то эта строка будет расти и займет две (или более) строки на экране; в крайней правой позиции каждой из занятых строк, кроме последней, будет расположен знак ‘\’. Знак ‘\’ говорит, что следующая строка экрана не является отдельной строкой в тексте, что это только *продолжение* строки, слишком длинной для данного экрана. Продолжение также называется *переносом строк*.

Иногда удобно, чтобы Emacs автоматически вставлял знаки перевода строки, когда строка становится слишком длинной; используйте для этого режим Auto Fill (см. Раздел 21.5 [Заполнение], с. 185).

Вместо продолжения Emacs может отображать длинные строки *усеченными*. Это значит, что все знаки, которые не вошли в ширину экрана или окна, не отображаются совсем. Они остаются в буфере, но временно невидимы. В последнем столбце ставится знак ‘$’, а не ‘\’, чтобы информировать о действии усечения.

Усечение вместо продолжения происходит, когда используется горизонтальная прокрутка, и, возможно, во всех окнах, не занимающих полную ширину фрейма (см. Глава 16 [Окна], с. 141). Вы можете включить усечение для конкретного буфера, установив в нем переменную `truncate-lines` не равной `nil`. (См. Раздел 31.2 [Переменные], с. 343.) Изменение значения `truncate-lines` делает его локальным для текущего буфера; до этого момента действует значение по умолчанию. Начальное значение по умолчанию — `nil`. См. Раздел 31.2.4 [Локальные переменные], с. 350.

См. Раздел 11.7 [Переменные изображения], с. 84, о других переменных, влияющих на способ отображения текста.

## 4.9 Информация о позиции курсора

Это команды для получения информации о размере и позиции частей буфера и для подсчета числа строк.

`M-x what-page`
: Напечатать номер страницы, на которой находится точка, и номер строки в пределах этой страницы.

`M-x what-line`
: Напечатать номер строки, в которой находится точка в буфере.

`M-x line-number-mode`
: Переключить режим автоматического отображения номера текущей строки.

`M-=`
: Напечатать количество строк в текущей области (`count-lines-region`). См. Глава 8 [Пометка], с. 63, чтобы узнать, что такое область.

`C-x =`
: Напечатать код знака после точки, знаковое положение точки и столбец точки (`what-cursor-position`).

Есть две команды для работы с номерами строк. `M-x what-line` подсчитывает номер текущей строки и показывает его в эхо-области. Чтобы перейти к строке с заданным номером, используйте `M-x goto-line`; она спросит у вас номер. Номера строк отсчитываются от единицы с начала буфера.

Вы также можете видеть номер текущей строки в строке режима; См. Раздел 1.3 [Строка режима], с. 25. Если вы сузили буфер, то номер строки в строке режима отсчитывается относительно доступной части (см. Раздел 30.8 [Сужение], с. 335).

В противоположность этому, `what-page` подсчитывает номер строки и относительно суженной области, и относительно всего буфера и показывает оба числа. См. Раздел 21.4 [Страницы], с. 184.

По этому поводу мы могли бы упомянуть `M-=` (`count-lines-region`), которая печатает число строк в области (см. Глава 8 [Пометка], с. 63). См. Раздел 21.4 [Страницы], с. 184, для информации о команде `C-x l`, которая считает строки на текущей странице.

Команда `C-x =` (`what-cursor-position`) может быть использована для узнавания столбца, в котором находится курсор, а также другой разнообразной информации о точке. Она печатает в эхо-области строку, которая выглядит следующим образом:

```
Char: c (0143, 99, 0x63) point=24781 of 31832(78%)  column 52
```

(На самом деле, это результат, полученный, когда точка была перед словом ‘`column`’ в самом примере.)

Четыре величины после ‘`Char:`’ описывают знак, следующий за точкой; первое показывает его, а остальные дают его восьмиричный, десятичный и шестнадцатиричный код. Для многобайтного не-ASCII-знака после этих чисел следует ‘`ext`’ и шестнадцатиричное представление в системе кодирования данного буфера, если эта система кодирования безопасно может представить этот знак в одном байте (см. Раздел 18.7 [Системы кодирования], с. 165). Если код знака больше одного байта, Emacs показывает ‘`ext ...`’.

После ‘`point=`’ следует позиция точки, выраженная через количество знаков. Началу буфера присваивается позиция 1, следующему знаку 2 и так далее. Следующее, большее число означает общее число знаков в буфере. Потом в круглых скобках находится позиция, выраженная как процент от общего размера.

После ‘`column`’ пишется горизонтальная позиция точки, в столбцах от левого края окна.

Если буфер был сужен, и некоторое количество текста в начале и в конце временно недоступно, то `C-x =` печатает дополнительный текст, описывающий текущие границы доступа. Например, команда может показать такое:

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

где два добавленных числа дают наименьшую и наибольшую допустимые для точки позиции знаков. Доступными являются знаки между двумя этими позициями. См. Раздел 30.8 [Сужение], с. 335.

Если точка находится в конце буфера (или в конце видимой части), в выводе `C-x =` не описывается знак, стоящий после точки. Вывод может выглядеть так:

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

`C-u C-x =` показывает вместо координат и столбца дополнительную информацию о знаке: имя набора знаков и коды данного знака в этом наборе; ASCII-знаки идентифицируются как принадлежащие набору знаков `ASCII`. Кроме того, после '`ext`' показывается полный код знака, даже если он занимает больше одного байта. Вот пример для знака A с акцентом грав из Latin-1, в буфере с системой кодирования iso-2022-7bit[1]:

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

## 4.10 Числовые аргументы

В математике и компьютерной практике слово *аргумент* означает "данные, предоставляемые функции или операции." Вы можете передать любой команде Emacs *числовой аргумент* (также называемый *префиксным аргументом*). Некоторые команды интерпретируют аргумент как счетчик повторений. Например, команда `C-f` с аргументом, равным десяти, передвигает вперед на десять знаков, а не на один. В этих командах отсутствие аргумента эквивалентно аргументу, равному единице. Отрицательные аргументы говорят большинству таких команд, что надо двигаться или действовать в противоположном направлении.

Если на вашей клавиатуре есть клавиша ⟨META⟩, простейший способ установить числовой аргумент — набрать цифры и/или знак минуса, прижав клавишу META. Например,

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

передвинула бы на пять строк вниз. Знаки `Meta-1`, `Meta-2`, и так далее, а также `Meta-` делают это, так как это ключи, привязанные к командам (`digit-argument` и `negative-argument`), которые определены как передающие аргумент в следующую команду. Цифры и `-` с модификатором Control или Control и Meta также задают числовой аргумент.

Другой способ задать аргумент — это использовать команду `C-u` (`universal-argument`), за которой следуют цифры аргумента. С помощью `C-u` вы можете набрать цифры аргумента, не удерживая клавиши-модификаторы; `C-u` работает на всех терминалах. Чтобы набрать отрицательный аргумент, введите после `C-u` знак минус. Просто знак минуса без цифр обычно обозначает $-1$.

`C-u`, за которой следует знак, не являющийся ни цифрой, ни знаком минус, имеет специальный смысл, "умножение на четыре". Она умножает аргумент последующей команды на четыре. Две последовательные `C-u` умножают его на шестнадцать. Таким образом, `C-u C-u C-f` передвигает вперед на шестнадцать знаков. Это удобный способ передвигаться вперед "быстро", так как он передвигает вперед примерно на 1/5 всех строк, умещающихся на экране обычного размера. Другие полезные сочетания — это `C-u C-n`, `C-u C-u C-n` (двигает вниз на большой кусок экрана), `C-u C-u C-o` (создает "много" пустых строк) и `C-u C-k` (уничтожает четыре строки).

Некоторым командам важно только присутствие или отсутствие аргумента, но не его значение. Например, команда `M-q` (`fill-paragraph`) без аргумента заполняет текст, а с аргументом вдобавок выравнивает текст по правой границе. (См. Раздел 21.5 [Заполнение], с. 185, для более подробной информации о `M-q`). Просто `C-u` дает удобный способ предоставления аргументов для таких команд.

Некоторые команды используют аргумент в качестве счетчика повторений, но их действия с аргументом и без него имеют некоторые специфические черты. Например, команда `C-k` (`kill-line`) с аргументом $n$ уничтожает $n$ строк, включая ограничивающие их знаки

[1] На терминалах, которые поддерживают Latin-1, знак после '`Char:`' отображается как настоящий глиф для A с акцентом грав.

новой строки. Но `C-k` без аргумента действует по-другому: она уничтожает текст вплоть до перевода строки, или, если точка стоит как раз в конце строки, уничтожает перевод строки. Таким образом, две команды `C-k` без аргумента могут уничтожить непустую строку, аналогичные результаты можно получить, набрав команду `C-k` с аргументом, равным единице. (См. Раздел 9.1 [Уничтожение], с. 69, для более подробной информации о `C-k`.)

Несколько команд трактуют простую `C-u` не так, как обыкновенный аргумент. Некоторые другие команды могут различать аргумент в виде просто знака минус и аргумент −1. Эти необычные случаи будут описаны, когда придет черед этих команд; они возникают всегда по причинам удобства использования отдельных команд.

Вы можете использовать числовой аргумент для вставки нескольких копий одного знака. Это работает прямо, если только знак — не цифра; например, `C-u 6 4 a` вставляет 64 копии знака ‘`a`’. Но это не работает для вставки цифр; `C-u 6 4 1` задает аргумент 641, а не вставляет что-либо. Чтобы отделить цифру от аргумента, наберите еще один `C-u`; например, `C-u 6 4 C-u 1` на самом деле вставляет 64 копии знака ‘`1`’.

Мы используем термин “префиксный аргумент”, как и “числовой аргумент”, чтобы подчеркнуть, что вы набираете эти аргументы перед командой, и чтобы отличить эти аргументы от аргументов минибуфера, которые пишутся после команды.

## 4.11 Повторение команды

Команда `C-x z` (`repeat`) предоставляет еще один способ повторить команду Emacs несколько раз. Эта команда повторяет предыдущую команду Emacs, какая бы она ни была. При повторении команда использует те же аргументы, какие у нее были раньше; она не считывает новые аргументы каждый раз.

Чтобы повторить команду несколько раз, напечатайте еще знаки `z`: каждый знак `z` повторит эту команду еще один раз. Повторение заканчивается, когда вы вводите знак, отличный от `z`, или нажимаете кнопку мыши.

Предположим например, что вы напечатали `C-u 2 0 C-d`, чтобы удалить 20 знаков. Вы можете повторить эту команду (вместе с ее аргументом) еще три раза, чтобы удалить в общей сложности 80 знаков, напечатав `C-x z z z`. Первый `C-x z` повторяет эту команду один раз, а каждый последующий `z` повторяет еще раз.

# 5 Минибуфер

*Минибуфер* используется командами Emacs для чтения более сложных, чем простое число, аргументов. Аргументами минибуфера могут быть имена файлов, имена буферов, имена функций Лиспа, имена команд Emacs и много других вещей в зависимости от считывающих аргумент команд. Для редактирования аргументов в минибуфере могут быть использованы обычные команды редактирования Emacs.

Когда вы используете минибуфер, он появляется в эхо-области, и туда передвигается курсор терминала. Начало строки минибуфера показывает *подсказку*, которая говорит, какой ввод вы должны давать, и как он будет применен. Часто подсказка происходит от имени команды, для которой нужен этот аргумент. Подсказка обычно кончается двоеточием.

Иногда в круглых скобках после двоеточия появляется *аргумент по умолчанию*; он тоже является частью подсказки. Значение по умолчанию будет использовано как значение аргумента, если вы вводите пустой аргумент (например, просто наберете ⟨RET⟩). К примеру, команды, считывающие имена буферов, всегда показывают значение, принимаемое по умолчанию; оно является именем буфера, который будет использован в дальнейшем, если вы просто наберете ⟨RET⟩.

Простейший способ дать аргумент минибуферу — набрать желаемый текст и завершить его ⟨RET⟩, клавишей для выхода из минибуфера. Вы можете отменить команду, которая запрашивает аргумент, и выйти из минибуфера, набрав `C-g`.

Минибуфер использует пространство экрана, отведенное под эхо-область, но это может противоречить другим способам использования эхо-области в Emacs. Emacs обрабатывает такие конфликты следующим образом:

- Если какая-то команда получает ошибку, пока вы находитесь в минибуфере, то это не отменяет минибуфер. Однако, эхо-область нужна для сообщения об ошибке, и поэтому сам минибуфер прячется на время. Он возвращается через несколько секунд или сразу же, как только вы наберете что-нибудь.
- Если в минибуфере вы используете команду, чьей целью является печать сообщения в эхо-области, такую как `C-x =`, то сообщение отображается как обычно, а минибуфер прячется на время. Он возвращается через несколько секунд, или как только вы наберете что-то еще.
- Когда используется минибуфер, эхо набираемых знаков не производится.

## 5.1 Минибуфер для имен файлов

Иногда минибуфер появляется уже с текстом. Например, когда вам предлагается задать имя файла, минибуфер появляется с содержащимся в нем *каталогом, заданным по умолчанию*, который оканчивается косой чертой. Это делается, чтобы проинформировать вас о том, в каком каталоге будет находиться файл, если вы не зададите каталог. Например, минибуфер может начинаться так:

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

где ‘`Find File: `’ служит подсказкой. Напечатав `buffer.c`, вы задаете файл с именем ‘`/u2/emacs/buffer.c`’. Чтобы найти файлы в соседних каталогах, используется `..`; таким образом, если вы наберете `../lisp/simple.el`, то получите файл ‘`/u2/emacs/lisp/simple.el`’. Или вы можете уничтожить ненужное вам имя каталога с помощью `M-`⟨DEL⟩ (см. Раздел 21.1 [Слова], с. 181).

Если вам не нужно ничего из каталога по умолчанию, вы можете уничтожить его с помощью `C-a C-k`. Но в этом нет необходимости; вы можете просто проигнорировать его. Вставьте после каталога по умолчанию абсолютное имя файла, начинающееся с косой черты или тильды. Например, чтобы задать файл ‘`/etc/termcap`’, просто вставьте это имя, при этом вы получите в минибуфере такую строку:

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

GNU Emacs придает двойной косой черте особое назначение (обычно нет смысла писать две косые черты подряд): они означают “пропустить все перед второй косой чертой в паре”. Таким образом, ‘`/u2/emacs/src/`’ в предыдущем примере будет проигнорировано, и вы получите файл ‘`/etc/termcap`’.

Если вы установите `insert-default-directory` равной `nil`, каталог по умолчанию не появляется в минибуфере. Таким образом, минибуфер появляется пустым. Но набираемое вами имя, если оно относительно, по-прежнему интерпретируется с учетом того же каталога по умолчанию.

## 5.2 Редактирование в минибуфере

Минибуфер — это буфер Emacs (хотя и своеобразный), и обычные команды Emacs годятся для редактирования текста вводимых вами аргументов.

Так как ⟨RET⟩ в минибуфере определен для выхода, вы не можете использовать его для вставки ограничителя новой строки в минибуфер. Чтобы вставить перевод строки, введите `C-o` или `C-q C-j`. (Напомним, что ограничитель новой строки — это на самом деле знак control-J.)

Минибуфер имеет свое собственное окно, которое всегда присутствует на экране, но действует так, как если бы его не было, когда минибуфер не используется. Когда же минибуфер используется, его ничем не отличается от всех остальных окон; вы можете перейти в другое окно с помощью `C-x o`, отредактировать текст в других окнах и может быть даже обратиться к нескольким файлам перед возвращением в минибуфер для представления аргумента. Вы можете уничтожить текст в другом окне, вернуться в окно минибуфера и затем восстановить этот текст, чтобы использовать его в аргументе. См. Глава 16 [Окна], с. 141.

Однако существуют некоторые ограничения на использование окна минибуфера. Вы не можете переключить в нем буферы — минибуфер и его окно связаны вместе. Точно также, вы не можете поделить или уничтожить окно минибуфера. Но вы можете сделать его выше обычным способом с помощью `C-x ^`. Если вы включите режим Resize-Minibuffer, то окно минибуфера по необходимости увеличивается вертикально, чтобы вместить текст, который вы поместили в минибуфер. Используйте `M-x resize-minibuffer-mode`, чтобы включить или выключить этот второстепенный режим (см. Раздел 31.1 [Второстепенные режимы], с. 341).

Прокрутка работает в окне минибуфера особым образом. Когда высота минибуфера равна только одной строке, и он содержит длинную строку текста, которая не уместилась бы не экране, с помощью прокрутки автоматически поддерживается перекрытие одной строки продолжения другой на определенное число знаков. Переменная `minibuffer-scroll-overlap` указывает число знаков, на которое нужно делать перекрытие; по умолчанию это 20.

Если во время нахождения в минибуфере вы запускаете команду, которая показывает в другом окне справочный текст любого вида, вы можете использовать в минибуфере команду `C-M-v` для прокрутки этого окна. Это свойство вступает в действие, если попытка получить завершение в минибуфере дает вам список возможных вариантов. См. Раздел 16.3 [Другие окна], с. 142.

Emacs обычно запрещает большинство команд, использующих минибуфер, когда минибуфер активен. Это правило было введено, чтобы рекурсивные минибуферы не запутывали начинающих пользователей. Если вы хотите получить возможность использования таких команд в минибуфере, установите переменную `enable-recursive-minibuffers` не равной `nil`.

## 5.3 Завершение

Для ввода значений некоторых видов аргументов вы можете использовать *завершение*. Завершение означает, что вы набираете часть аргумента, а Emacs явно дополняет остаток полностью или настолько, насколько можно определить по набранной вами части.

Когда завершение доступно, определенные клавиши — ⟨TAB⟩, ⟨RET⟩ и ⟨SPC⟩ — переприв-язываются для дополнения представленного в минибуфере текста до более длинной строки, которую этот текст представляет, путем сопоставления его с *вариантами завершения*, ко-торые предоставляются командой, считывающей этот аргумент. Знак `?` определен как показывающий список возможных завершений того, что вы уже ввели.

Например, когда `M-x` использует минибуфер для чтения имени команды, она предостав-ляет для завершения список всех имеющихся имен команд Emacs. Клавиши завершения сопоставляют текст в минибуфере со всеми именами команд, находят какие-то дополни-тельные знаки имени, которые подразумеваются уже представленными в минибуфере, и добавляют эти знаки к тем, которые вы дали. Поэтому можно напечатать `M-x ins` ⟨SPC⟩ `b` ⟨RET⟩ вместо `M-x insert-buffer` ⟨RET⟩ (например).

Регистр обычно имеет значение при завершении, так как он имеет значение в боль-шинстве имен, которые вы можете завершить (имена буферов, файлов, команд). Таким образом, ‘`fo`’ не завершится до ‘`Foo`’. Когда вы завершаете имя, в котором регистр не имеет значения, то при завершении регистр может быть проигнорирован.

### 5.3.1 Пример завершения

Здесь может помочь конкретный пример. Если вы наберете `M-x au` ⟨TAB⟩, ⟨TAB⟩ ищет варианты (в данном случае имена команд), которые начинаются с ‘`au`’. Их несколько, включая `auto-fill-mode` и `auto-save-mode`, но все они одинаково начинаются на `auto-`, поэтому ‘`au`’ в минибуфере изменяется на ‘`auto-`’.

Если вы немедленно снова наберете ⟨TAB⟩, то появятся несколько возможностей для следующих знаков — это может быть любой из ‘`cfilrs`’, — поэтому больше ничего не добавляется, но в другом окне показывается список всех возможных завершений.

Если вы продолжаете набирать `f` ⟨TAB⟩, то этот ⟨TAB⟩ увидит ‘`auto-f`’. Единственное имя команды, которое так начинается, — это `auto-fill-mode`, таким образом завершение вставляет ее остаток. Теперь вы имеете в минибуфере ‘`auto-fill-mode`’ просто после набора `au` ⟨TAB⟩ `f` ⟨TAB⟩. Заметьте, что ⟨TAB⟩ обладает в минибуфере таким действием, потому что когда возможно завершение, он привязан к команде `minibuffer-complete`.

### 5.3.2 Команды завершения

Здесь представлен список всех команд завершения, определенных в минибуфере, когда завершение доступно.

⟨TAB⟩ Завершить представленный в минибуфере текст насколько это возможно (`minibuffer-complete`).

⟨SPC⟩ Завершить текст в минибуфере, но не прибавлять более одного слова (`minibuffer-complete-word`).

⟨RET⟩ Представить текст в минибуфере как аргумент, возможно сначала дополняя его, как описано ниже (`minibuffer-complete-and-exit`).

`?` Напечатать список всех возможных завершений текста в минибуфере (`minibuffer-list-completions`).

⟨SPC⟩ завершает очень похоже на ⟨TAB⟩, но никогда не идет дальше следующего дефи-са или пробела. Если в минибуфере есть ‘`auto-f`’, и вы наберете ⟨SPC⟩, он обнаружит,

что завершение — это ‘auto-fill-mode’, но остановит завершение после ‘fill-’. Это даст ‘auto-fill-’. Еще один ⟨SPC⟩ в этом месте завершает полностью до ‘auto-fill-mode’. ⟨SPC⟩ в минибуфере, когда возможно завершение, запускает функцию minibuffer-complete-word.

Вот несколько команд, которые вы можете использовать для выбора завершения в окне, показывающем перечень вариантов:

Mouse-2 Щелчок второй кнопкой мыши выбирает одно из завершений из перечня возможных вариантов (mouse-choose-completion). Обычно вы используете эту команду, когда точка находится в минибуфере; но вы должны щелкнуть в перечне завершений, а не самом минибуфере.

⟨PRIOR⟩
M-v Нажатие ⟨PRIOR⟩ (или ⟨PAGE-UP⟩) или M-v, когда вы находитесь в минибуфере, выбирает окно, показывающее буфер с перечнем завершений (switch-to-completions). Это прокладывает путь к использованию команд, перечисленных ниже. (Выбор это окна обычным способом имеет тот же эффект, но этот способ удобнее.)

⟨RET⟩ Нажатие ⟨RET⟩ *в буфере с перечнем завершений* выбирает завершение, на котором или после которого находится точка (choose-completion). Чтобы использовать эту команду, вы должны сначала переключиться в окно, показывающее перечень завершений.

⟨RIGHT⟩ Нажатие правой курсорной стрелки ⟨RIGHT⟩ *в буфере с перечнем завершений* передвигает точку к следующему завершению (next-completion).

⟨LEFT⟩ Нажатие левой курсорной стрелки ⟨LEFT⟩ *в буфере с перечнем завершений* передвигает точку ближе к началу буфера, к предыдущему завершению (previous-completion).

### 5.3.3 Строгое завершение

Существует три различных варианта работы ⟨RET⟩ при завершении в минибуфере, зависящие от того, как будет использован аргумент.

- *Строгое* завершение используется, когда бессмысленно давать какой-либо другой аргумент, кроме одной из известных альтернатив. Например, когда C-x k считывает имя буфера для уничтожения, то бессмысленно давать что либо, кроме имени существующего буфера. При строгом завершении ⟨RET⟩ отказывается выходить, если текст в минибуфере не завершается с точным соответствием шаблону.
- *Осторожное* завершение подобно строгому завершению за исключением того, что ⟨RET⟩ выходит, только если текст уже совпадал в точности с шаблоном, не нуждающемся в завершении. Если текст — не точное совпадение, ⟨RET⟩ не выходит, но делает завершение текста. Если это завершает текст до точного совпадения, то второй ⟨RET⟩ выйдет.

  Осторожное завершение используется для чтения имен файлов, которые должны уже существовать.
- *Свободное* завершение используется, когда любая строка является подходящей, и список альтернатив завершения дается просто для справки. Например, когда C-x C-f считывает имя файла для обращения, допустимо любое имя файла, на случай, если вы захотите создать новый файл. При свободном завершении ⟨RET⟩ берет текст в минибуфере точно как он дан, не завершая его.

Команды завершения показывают в окне список всех возможных вариантов всякий раз, когда возможно более чем одно завершение для следующего знака. Кроме того, набор ? явно запрашивает такой список. Если перечень завершений длинный, вы можете прокрутить его с помощью C-M-v (см. Раздел 16.3 [Другие окна], с. 142).

### 5.3.4 Параметры завершения

При завершении имен файлов определенные имена обычно игнорируются. Переменная `completion-ignored-extension` содержит список строк; файл, чье имя кончается на любую из этих строк, игнорируется как возможное завершение. Стандартное значение этой переменной имеет несколько элементов, включая `".o"`, `".elc"`, `".dvi"` и `"~"`. Действие таково: например, ‘`foo`’ может завершиться до ‘`foo.c`’, даже если ‘`foo.o`’ также существует. Однако, если *все* возможные завершения кончаются на “игнорируемые” строки, тогда они не игнорируются. Игнорируемые расширения не относятся к спискам завершений — эти списки всегда упоминают все возможные завершения.

Обычно команда завершения, которая обнаруживает, что следующий знак нельзя определить, автоматически показывает список всех возможных завершений. Если переменная `completion-auto-help` установлена в значение `nil`, этого не происходит, и чтобы просмотреть возможные варианты, вы должны набрать `?`.

Библиотека `complete` реализовывает более мощный вид завершения, который может дополнять несколько слов одновременно. Например, она может завершить сокращение имени команды `p-b` до `print-buffer`, потому что ни одна другая команда не начинается с двух слов, чьи первые буквы — это ‘`p`’ и ‘`b`’. Чтобы использовать эту библиотеку, напишите `(load "complete")` в вашем файле ‘`~/.emacs`’ (см. Раздел 31.7 [Файл инициализации], с. 366).

Режим Icomplete предоставляет постоянно обновляющуюся информацию, говорящую вам, какие завершения доступны для уже введенного текста. Этот второстепенный режим включается и выключается с помощью команды `M-x icomplete-mode`.

## 5.4 История минибуфера

Все введенные вами в минибуфер аргументы сохраняются в *списке истории минибуфера*, чтобы вы могли снова использовать их позже в новом аргументе. Текст старых аргументов вставляется в минибуфер особыми командами. Они сбрасывают старое содержимое минибуфера, так что вы можете думать о них как о командах движения по списку предыдущих аргументов.

⟨UP⟩
`M-p` Перемещается к предыдущей строке аргумента, сохраненной в истории минибуфера (`previous-history-element`).

⟨DOWN⟩
`M-n` Перемещается к следующей строке аргумента, сохраненной в истории минибуфера (`next-history-element`).

`M-r` *regexp* ⟨RET⟩
Перемещается к более раннему аргументу в истории минибуфера, в котором есть совпадение с *regexp* (`previous-matching-history-element`).

`M-s` *regexp* ⟨RET⟩
Перемещается к более позднему аргументу в истории минибуфера, в котором есть совпадение с *regexp* (`next-matching-history-element`).

Простейший способ повторно использовать сохраненные аргументы из списка истории — передвигаться по списку истории по одному аргументу. Находясь в минибуфере, используйте `M-p` или стрелку вверх (`previous-history-element`) чтобы “перейти” к предыдущему вводу минибуфера, и `M-n` или стрелку вниз (`next-history-element`) для “перехода” к следующему.

Предыдущий ввод, который вы извлекаете из истории, полностью замещает содержимое минибуфера. Чтобы использовать его в качестве аргумента, выйдите из минибуфера

как обычно, с помощью ⟨RET⟩. Вы также можете отредактировать текст перед использованием; это не изменяет элемент истории, к которому вы "перешли", но ваш новый аргумент вставляется в конец списка истории как отдельный элемент.

Для многих аргументов минибуфера есть значение "по умолчанию". В некоторых случаях команды истории знают значение по умолчанию. Тогда вы можете вставить это значение в минибуфер в виде текста, используя `M-n` для перехода "в будущее" по истории. Мы надеемся в конце концов сделать это свойство доступным всегда, когда у минибуфера есть значение по умолчанию.

Также есть команды для поиска вперед или назад по истории; они производят поиск элементов истории, которые соответствуют задаваемому вами в минибуфере регулярному выражению. `M-r` (`previous-matching-history-element`) производит поиск более старых элементов истории, тогда как `M-s` (`next-matching-history-element`) производит поиск более новых элементов. По особому разрешению эти команды могут использовать минибуфер для считывания аргументов, хотя вы уже находитесь в минибуфере, когда запускаете их. Как и при наращиваемом поиске, заглавная буква в регулярном выражении делает поиск регистрозависимым (см. Раздел 12.6 [Поиск и регистр], с. 95).

Каждое использование минибуфера записывает ваш ввод в список истории, но есть отдельные списки для различных видов аргументов. Например, есть список для имен файлов, используемый всеми командами, считывающими имена файлов. (У него есть особенность: этот список истории записывает абсолютное имя файла, не больше и не меньше, даже если вы ввели его иначе.)

Существуют несколько других особенных списков истории, включая список для имен команд, считываемых командой `M-x`, список для имен буферов, список для аргументов команд вроде `query-replace` и список команд компиляции, считываемых командой `compile`. И наконец, есть список истории для "разного", который используют большинство аргументов минибуфера.

Переменная `history-length` задает максимальную длину списка истории минибуфера; как только список дорастает до этой длины, самый старый элемент удаляется при добавлении нового. Если значение `history-length` равно `t`, то ограничения на длину нет, и элементы не удаляются никогда.

## 5.5 Повтор команд минибуфера

Каждая команда, которая использует минибуфер по крайней мере один раз, записывается в специальный список истории вместе со значением аргументов минибуфера, так что вы можете легко ее повторить. В частности, записывается каждое использование `M-x`, так как `M-x` использует минибуфер для чтения имени команды.

`C-x` ⟨ESC⟩ ⟨ESC⟩
: Повторное выполнение последней команды минибуфера (`repeat-complex-command`).

`M-x list-command-history`
: Вывести полную историю всех команд, которые может повторить `C-x` ⟨ESC⟩ ⟨ESC⟩, начиная с более новых.

`C-x` ⟨ESC⟩ ⟨ESC⟩ используется для повторного выполнения недавней команды, использовавшей минибуфер. Запущенная без аргумента, она повторяет последнюю такую команду. Числовой аргумент определяет, какую команду повторить; единица означает первую от конца, а большие числа определяют более ранние команды.

`C-x` ⟨ESC⟩ ⟨ESC⟩ действует при помощи превращения предыдущей команды в выражение Лиспа и последующего входа в минибуфер, инициализированый текстом этого выражения. Если вы набираете просто ⟨RET⟩, то команда повторяется как прежде. Вы можете также

изменить команду путем редактирования лисповского выражения. То выражение, что вы в конце концов представите, и будет выполнено. Повторенная команда добавляется в начало истории команд, если она не идентична самой последней выполненной команде, которая уже была там.

Даже если вы не понимаете синтаксиса Лиспа, вероятнее всего будет очевидно, какая команда отражается для повторения. Если вы не меняли текст, вы можете быть уверены, что он повторится точно так, как прежде.

Находясь в минибуфере по команде `C-x` ⟨ESC⟩ ⟨ESC⟩, вы можете использовать команды истории минибуфера (`M-p`, `M-n`, `M-r`, `M-s`; см. Раздел 5.4 [История минибуфера], с. 49) для продвижения по списку сохраненных полностью команд. После того, как вы найдете желаемую предыдущую команду, вы можете как обычно отредактировать ее выражение и затем предоставить его снова, набрав ⟨RET⟩.

Перечень предыдущих команд, использовавших минибуфер, хранится как лисповский список в переменной `command-history`. Каждый элемент — это лисповское выражение, которое описывает одну команду и ее аргументы. Программы на Лиспе могут повторно выполнять команды путем передачи в `eval` соответствующего элемента `command-history`.

# 6 Запуск команд по имени

Команды Emacs, которые применяются часто или должны быстро набираться, привязываются к ключам — коротким последовательностям знаков — для удобства использования. Другие команды Emacs, не нуждающиеся в краткости, не привязаны к ключам; чтобы запустить их, вы должны обратиться к ним по имени.

По соглашению имя команды создается из одного или более слов, разделенных дефисами; например, `auto-fill-mode` или `manual-entry`. Использование английских слов делает имена команд более легкими для запоминания, чем составленные из непонятных знаков ключи, даже если слова дольше набирать.

Чтобы запустить на выполнение команду по имени, сначала наберите `M-x`, за ним имя команды и закончите набором ⟨RET⟩. '`M-x`' использует минибуфер для чтения имени команды. ⟨RET⟩ выходит из минибуфера и запускает команду на выполнение. Строка '`M-x`' показывается в начале буфера как *подсказка*, чтобы напомнить вам, что вы должны ввести имя команды, которая будет запущена на выполнение. См. Глава 5 [Минибуфер], с. 45, для полной информации о свойствах минибуфера.

При вводе имени команды вы можете использовать завершение. Например, команда `forward-char` может быть запущена по имени при помощи

```
M-x forward-char ⟨RET⟩
```

или

```
M-x forw ⟨TAB⟩ c ⟨RET⟩
```

Заметим, что `forward-char` — это та же самая команда, которую вы вызываете с помощью ключа `C-f`. Любая команда Emacs может быть вызвана по имени с использованием `M-x` независимо от того, существует ли связанный с ней ключ.

Если во время считывания имя команды вы набираете `C-g`, то вы отменяете команду `M-x` и покидаете минибуфер, по завершении поднимаясь на верхний уровень.

Чтобы передать числовой аргумент команде, которую вы вызываете с помощью `M-x`, задайте числовой аргумент перед `M-x`. `M-x` передает аргумент команде, которую вызывает. Значение аргумента появляется в подсказке в то время, когда считывается имя команды.

Если набираемая вами команда привязана к собственному ключу, Emacs говорит об этом в эхо-области спустя две секунды после завершения команды (если вы до этого ничего не ввели). Например, если вы напечатаете `M-x forward-word`, вы получите сообщение, говорящее, что эту же команду можно запустить более просто, введя `M-f`. Вы можете отключить эти сообщения, установив `suggest-key-bindings` в значение `nil`.

В этом руководстве при описании команд, запускаемых по имени, мы обычно опускаем ⟨RET⟩, который необходим для ограничения имени. То есть мы чаще пишем `M-x auto-fill-mode`, а не `M-x auto-fill-mode` ⟨RET⟩. Мы упоминаем ⟨RET⟩, только когда необходимо подчеркнуть его присутствие, например при описании последовательности ввода, которая содержит имя команды и следующие за ним аргументы.

`M-x` определен на выполнение команды `execute-extended-command`, которая отвечает за считывание имени другой команды и ее запуск.

# 7 Справка

Emacs предоставляет широкие возможности предоставления справки, которые доступны через один знак, `C-h`. `C-h` — это префиксный ключ, который используется только для команд печати документации. Знаки, которые вы можете набрать после `C-h`, называются параметрами справки. Одним из них является `C-h`; с его помощью вы запрашиваете информацию об использовании `C-h`. Функциональная клавиша ⟨F1⟩ служит эквивалентом `C-h`.

`C-h C-h` (`help-for-help`) печатает список всех возможных параметров справки с кратким описанием каждого из них. До набора параметра справки вы можете пролистать этот список с помощью ⟨SPC⟩ или ⟨DEL⟩.

`C-h` или ⟨F1⟩ означает "помощь" также и во многих других контекстах. Например, в процессе работы `query-replace` они описывают доступные возможные действия над текущим найденным совпадением. После префиксного ключа они выводят перечень вариантов, которые могут следовать за этим префиксным ключом. (Некоторые префиксные ключи не поддерживают `C-h`, потому что они определяют для него другие значения; но все они поддерживают ⟨F1⟩.)

Большинство справочных буферов используют специальный основной режим, режим Help, который позволяет вам удобно делать прокрутку с помощью ⟨SPC⟩ и ⟨DEL⟩.

Ниже приведен обзор всех определенных команд для получения справки.

`C-h a` *regexp* ⟨RET⟩
: Показать список команд, чьи имена соответствуют регулярному выражению *regexp* (`apropos-command`).

`C-h b`
: Показать таблицу всех привязок ключей, действующих на данный момент в следующем порядке: привязки текущих второстепенных режимов, привязки основного режима и глобальные привязки (`describe-bindings`).

`C-h c` *ключ*
: Напечатать имя команды, которую запускает на выполнение заданный *ключ* (`describe-key-briefly`). Здесь `c` означает 'character' ('знак'). Для получения более подробной информации о *ключе*, используйте `C-h k`.

`C-h f` *функция* ⟨RET⟩
: Показать документацию на лисповскую функцию с именем *функция* (`describe-function`). Так как команды являются лисповскими функциями, вы можете использовать имя команды.

`C-h h`
: Вывести файл '`hello`', который показывает примеры различных наборов знаков.

`C-h i`
: Запустить Info, программу для просмотра файлов документации (`info`). Полное руководство по Emacs существует как диалоговый файл в Info.

`C-h k` *ключ*
: Показать имя и описание команды, которую запускает *ключ* (`describe-key`).

`C-h l`
: Показать описание последних 100 набранных вами знаков (`view-lossage`).

`C-h m`
: Показать описание текущего основного режима (`describe-mode`).

`C-h n`
: Показать описание изменений в Emacs, первым самое последнее (`view-emacs-news`).

`C-h p`
: Найти пакет по ключевому слову (`finder-by-keyword`).

`C-h s`
: Показать текущее содержимое синтаксической таблицы и объяснить, что оно означает (`describe-syntax`). См. Раздел 31.6 [Синтаксис], с. 366.

`C-h t` Войти в интерактивный самоучитель по Emacs (`help-with-tutorial`).

`C-h v` *пер* ⟨RET⟩
Показать описание лисповской переменной *пер* (`describe-variable`).

`C-h w` *команда* ⟨RET⟩
Напечатать, какие ключи запускают на выполнение команду с именем *команда* (`where-is`).

`C-h C` *кодировка* ⟨RET⟩
Вывести описание системы кодирования с именем *кодировка* (`describe-coding-system`).

`C-h C` ⟨RET⟩
Описать используемую в данный момент систему кодирования.

`C-h I` *метод* ⟨RET⟩
Описать метод ввода (`describe-input-method`).

`C-h L` *языковая-среда* ⟨RET⟩
Показать сведения о наборах знаков, системах кодирования и методах ввода, используемых в *языковой-среде* (`describe-language-environment`).

`C-h C-c` Показать условия копирования GNU Emacs.

`C-h C-d` Показать информацию о получении новых версий GNU Emacs.

`C-h C-f` *функция* ⟨RET⟩
Войти в Info и перейти к ноде, описывающей функцию Emacs с именем *функция* (`Info-goto-emacs-command-node`).

`C-h C-k` *ключ*
Войти в Info и перейти к ноде, описывающей заданную последовательность ключей (`Info-goto-emacs-key-command-node`).

`C-h C-p` Показать информацию о Проекте GNU.

`C-h` ⟨TAB⟩ *языковой-символ* ⟨RET⟩
Показать документацию Info на *языковой-символ* в соответствии с языком программирования, на котором вы пишите (`info-lookup-symbol`).

## 7.1 Описания для ключей

Основными параметрами для `C-h` являются `C-h c` (`describe-key-briefly`) и `C-h k` (`describe-key`). `C-h c` *ключ* печатает в эхо-области имя команды, к которой привязан *ключ*. Например, `C-h c C-f` печатает ‘`forward-char`’. Поскольку имена команд выбираются так, чтобы они описывали действие команды, это удобный способ получить очень короткое описание того, что делает *ключ*.

`C-h k` *ключ* похожа на предыдущую, но дает больше информации. Она показывает строку документации команды, а также ее имя. Это слишком много для эхо-области, поэтому для показа используется окно.

`C-h c` и `C-h k` работают для любых видов ключей, включая функциональные клавиши и события от мыши.

## 7.2 Справка по имени команды или переменной

`C-h f` (`describe-function`) читает имя функции Лиспа, используя минибуфер, затем показывает строку документации этой функции в окне. Так как команды — это лисповские функции, вы можете пользоваться этим для получения описания команды, известной по имени. Например,

```
C-h f auto-fill-mode ⟨RET⟩
```

выдает документацию для `auto-fil-mode`. Это единственный способ увидеть документацию команды, которая не привязана к какому-нибудь ключу (одной из тех, которых вы обычно вызываете при помощи `M-x`).

`C-h f` также полезна для лисповских функций, которые вы планируете использовать в программе на Лиспе. Например, если вы просто написали выражение `(make-vector len)` и хотите быть уверенными в том, что вы верно использовали `make-vector`, наберите `C-h f make-vector ⟨RET⟩`. Так как `C-h f` воспринимает имена всех функций, а не только имена команд, то вы можете обнаружить, что некоторые из ваших любимых сокращений, которые работают в `M-x`, не работают в `C-h f`. Сокращения могут быть уникальными среди имен команд и уже не быть уникальными, когда рассматриваются также и имена других функций.

Имя функции для `C-h f` имеет значение по умолчанию, которое используется, если вы наберете ⟨RET⟩, оставляя минибуфер пустым. По умолчанию это будет функция, вызванная самым глубоким лисповским выражением в буфере в районе точки, *при условии*, что это правильное, определенное имя функции Лиспа. Например, если точка расположена вслед за текстом ‘`(make-vector (car x)`’, то самый глубокий список, содержащий точку, это тот, что начинается с ‘`(make-vector`’; таким образом, по умолчанию будет описана функция `make-vector`.

`C-h f` часто полезна просто для проверки правильности написания имени функции с точки зрения орфографии. Если `C-h f` упоминает в подсказке значение по умолчанию, то вы набрали имя определенной лисповской функции. Если это все, что вы хотели узнать, просто наберите `C-g`, чтобы отменить команду `C-h f` и продолжить редактирование.

`C-h w` *команда* ⟨RET⟩ сообщает вам, какие ключи привязаны к *команде*. Она печатает список ключей в эхо-области. Если она говорит, что команда не привязана ни к одному ключу, вы должны использовать для вызова этой команды `M-x`. `C-h w` запускает команду `where-is`.

`C-h v` (`describe-variable`) похожа на `C-h f`, но описывает переменные Лиспа, а не функции. Ее значение по умолчанию — это лисповский символ поблизости или перед точкой, но только если это имя известной лисповской переменной. См. Раздел 31.2 [Переменные], с. 343.

## 7.3 Поиск по контексту

Более сложный вид запросов — это вопросы вроде “Какие команды используются для работы с файлами?” Чтобы задать такой вопрос, наберите `C-h a file ⟨RET⟩`, которая покажет список всех имен команд, которые содержат ‘`file`’, такие как `copy-file`, `find-file` и так далее. Вместе с именем команды показывается краткое описание того, как ее использовать, и какие ключи ее запускают. Например, вам сообщат, что вы можете запустить `find-file`, набрав `C-x C-f`. Здесь `a` в `C-h` — это сокращение для ‘Apropos’, `C-h a` запускает на выполнение функцию `command-apropos`. Эта команда проверяет только команды (интерактивные функции); если вы зададите префиксный аргумент, она просмотрит также и неинтерактивные функции.

Так как `C-h a` ищет только функции, чьи имена содержат заданную вами строку, вы должны быть изобретательны в выборе строк. Если вы ищете команды для уничтожения

в обратном направлении, и `C-h a kill-backward` ⟨RET⟩ ничего не показывает, не отчаивайтесь. Попробуйте просто `kill`, или просто `backward` или просто `back`. Будьте настойчивы. Притворитесь, что вы играете в Adventure. Также отметим, что для большей гибкости вы можете использовать в качестве аргумента регулярное выражение (см. Раздел 12.5 [Регулярные выражения], с. 91).

Здесь представлен набор аргументов для передачи в `C-h a`, который охватывает множество классов команд, так как существуют строгие соглашения для имен стандартных команд Emacs. Давая вам почувствовать принятые соглашения об именовании, этот набор также должен служить вам помощью в совершенствовании технических приемов для подбора строк `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.

Для получения перечня всех пользовательских переменных, которые соответствуют регулярному выражению, используйте команду `M-x apropos-variable`. Эта команда показывает по умолчанию только пользовательские переменные; если вы зададите префиксный аргумент, она просмотрит все переменные.

Чтобы перечислить все лисповские символы, которые содержат совпадение с регулярным выражением, а не только те, которые определены в качестве команд, используйте команду `M-x apropos`, а не `C-h a`. По умолчанию эта команда не проверяет привязки ключей; если вы хотите узнать их, задайте этой команде префиксный аргумент.

Команда `apropos-documentation` похожа на `apropos`, но производит поиск совпадений с регулярным выражением не только в именах символов, но и в строках описания.

Команда `apropos-value` действует как `apropos`, за исключением того, что ищет совпадения с регулярным выражением в значениях символов. Эта команда по умолчанию не проверяет определения функций или списки свойств; задайте ей числовой аргумент, если вы хотите проверить и их.

Если переменная `apropos-do-all` не равна `nil`, то все описанные выше команды ведут себя так, как-будто им задан префиксный аргумент.

Если вы хотите получить больше информации об определении функции, о переменной или о свойствах символа, перечисленных в буфере Apropos, вы можете щелкнуть на них `Mouse-2` или переместиться туда и нажать ⟨RET⟩.

## 7.4 Поиск в библиотеках Лиспа по ключевым словам

Команда `C-h p` позволяет вам производить поиск в стандартных библиотеках Emacs Lisp по тематическим ключевым словам. Вот неполный перечень ключевых слов, которые вы можете использовать:

abbrev — управление сокращениями, быстрые клавиши, макросы.
bib — поддержка обработчика библиографий `bib`.
c — поддержка языков Си и Си++.
calendar — поддержка календаря и операций со временем.
comm — коммуникации, сети, удаленный доступ к файлам.
data — поддержка редактирования файлов с данными.
docs — поддержка документации Emacs.
emulations — эмуляция других редакторов.
extensions — расширения языка Emacs Lisp.
faces — поддержка разных начертаний (шрифтов и цветов).
frames — поддержка фреймов и оконных систем.
games — игры, шутки и развлечения.

hardware — поддержка интерфейсов с экзотической аппаратурой.
help — поддержка интерактивных справочных систем.
hypermedia — поддержка для ссылок внутри текста.
i18n — поддержка разных языков и алфавитов.
internal — внутренний код Emacs, сборка, значения по умолчанию.
languages — специализированные режимы для редактирования кода.
lisp — поддержка использования Лиспа (включая Emacs Lisp).
local — библиотеки, локальные для вашей системы.
maint — средства поддержки для группы разработчиков Emacs.
mail — режимы для работы с электронной почтой.
matching — поисковые программы.
news — поддержка чтения и отправки сетевых новостей.
non-text — поддержка для редактирования нетекстовых файлов.
oop — поддержка объектно-ориентированного программирования.
outlines — просмотр иерархической структуры текста.
processes — процессы, оболочка, компиляция и управление задачами.
terminals — поддержка разных типов терминалов.
tex — поддержка для программы компьютерного набора TEX.
tools — утилиты для программирования.
unix — интерфейсы или эмуляторы возможностей Unix.
vms — поддержка VMS.
wp — обработка текста.

## 7.5 Справка о поддержке различных языков

Для получения справки о поддержке определенной языковой среды используйте команду `C-h L` (`describe-language-environment`). См. Раздел 18.3 [Языковые среды], с. 162. Эта команда говорит вам, для каких языков полезна данная языковая среда, и перечисляет наборы знаков, системы кодирования и методы ввода, которые используются в этой среде. Она также показывает образец текста для демонстрации его внешнего вида.

Команда `C-h h` (`view-hello-file`) отображает файл ‘`etc/HELLO`’, который показывает, как сказать “Здравствуйте” на разных языках.

Команда `C-h I` (`describe-input-method`) описывает метод ввода, либо заданный явно, либо, по умолчанию, используемый в данный момент. См. Раздел 18.4 [Методы ввода], с. 163.

Команда `C-h C` (`describe-coding-system`) описывает систему кодирования, либо заданную явно, либо, по умолчанию, используемые в данный момент. См. Раздел 18.7 [Системы кодирования], с. 165.

## 7.6 Команды режима Help

Справочные буферы предоставляют команды режима View (см. Раздел 14.10 [Файлы Разное], с. 132) плюс несколько собственных особых команд.

⟨SPC⟩ Прокручивает вперед.

⟨DEL⟩ Прокручивает назад.

⟨RET⟩ Переходит по перекрестной ссылке в точке.

⟨TAB⟩ Перемещает вперед к следующей перекрестной ссылке.

`S-`⟨TAB⟩ Перемещает назад к предыдущей перекрестной ссылке.

`Mouse-2` Переходит по перекрестной ссылке, на которой вы щелкнули.

Когда в описании появляется имя команды (см. Глава 6 [Запуск команд по имени], с. 53) или имя переменной (см. Раздел 31.2 [Переменные], с. 343), оно обычно показывается в парных одиночных кавычках. Вы можете щелкнуть на имени с помощью `Mouse-2` или передвинуть к нему точку и нажать ⟨RET⟩, чтобы просмотреть документацию на эту команду или переменную. Используйте `C-c C-b` для возврата.

Есть удобные команды для перемещения точки к перекрестным ссылкам в тексте справки. ⟨TAB⟩ (`help-next-ref`) передвигает точку вперед к следующей перекрестной ссылке. Чтобы переместить точку к предыдущей ссылке, используйте `S-`⟨TAB⟩ (`help-previous-ref`).

## 7.7 Другие команды для получения справки

`C-h i` (`info`) запускает на выполнение программу Info, которая используется для просмотра структурированных файлов документации. Внутри Info имеется полное руководство по Emacs. В конце концов там будет доступна вся документация по системе GNU. Чтобы запустить самоучитель по использованию Info, наберите `h` после входа в Info.

Если вы зададите числовой аргумент, `C-h i` запросит имя файла документации. Таким образом вы сможете просмотреть файл, для которого нет вхождения в меню Info верхнего уровня. Это также удобно, когда вам нужно получить документацию быстро, и вы знаете точное имя этого файла.

Существуют две специальные справочные команды для получения документации по Emacs через Info. `C-h C-f` *функция* ⟨RET⟩ входит в Info и переходит непосредственно к описанию функции Emacs с именем *функция*. `C-h C-k` *ключ* входит в Info и переходит к описанию *ключа*. Эти ключи запускают команды `Info-goto-emacs-command-node` и `Info-goto-emacs-key-command-node`.

При редактировании программы, если у вас есть Info-версия руководства для этого языка программирования, вы можете использовать команду `C-h C-i` для просмотра описания символа (ключевого слова, функции или переменной). Детали работы этой программы зависят от основного режима.

Если происходит что-то странное, и вы не уверены, какую команду вы набрали, используйте `C-h l` (`view-lossage`). `C-h l` печатает последние 100 набранных вами командных знаков. Если вы увидели команды, которые вам не знакомы, вы можете использовать `C-h c`, чтобы отыскать информацию о том, что они делают.

Emacs имеет множество основных режимов, каждый из которых переопределяет несколько ключей и делает некоторые другие изменения в работе редактора. `C-h m` (`describe-mode`) печатает документацию о текущем основном режиме, в которой обычно описаны все команды, измененные в этом режиме.

`C-h b` (`describe-bindings`) и `C-h s` (`describe-syntax`) предоставляют другую информацию о текущем режиме Emacs. `C-h b` показывает список всех привязанных ключей, действующих в данный момент; сначала идут локальные привязки текущих второстепенных режимов, за ними локальные привязки текущего основного режима и, наконец, глобальные привязки (см. Раздел 31.4 [Привязки ключей], с. 356). `C-h s` показывает содержимое синтаксической таблицы с объяснением синтаксиса каждого знака (см. Раздел 31.6 [Синтаксис], с. 366).

Вы можете получить подобный перечень для конкретного префиксного ключа, набрав после него `C-h`. (Есть несколько префиксных ключей, для которых это не работает — они придают свои привязки для `C-h`. Один из них — это ⟨ESC⟩, потому что ⟨ESC⟩ `C-h` на самом деле эквивалентен `C-M-h`, который помечает определение функции.)

Другие параметры `C-h` показывают разнообразные файлы с полезной информацией. `C-h C-w` показывает подробности об отсутствии гарантий для GNU Emacs. `C-h n` (`view-emacs-news`) показывает файл ‘`emacs/etc/NEWS`’, который содержит описание изменений в Emacs,

сортированное в хронологическом порядке. `C-h F` (`view-emacs-FAQ`) выводит список часто отвечаемых вопросов о Emacs. `C-h t` (`help-with-tutorial`) показывает обучающее руководство по Emacs. `C-h C-c` (`describe-copying`) показывает файл ‘`emacs/etc/COPYING`’, который объясняет условия, которые вы должны выполнить при распространении копий Emacs. `C-h C-d` (`describe-distribution`) показывает файл ‘`emacs/etc/DISTRIB`’, в котором рассказывается, как вы можете заказать копию последней версии Emacs. `C-h C-p` (`describe-project`) выводит общую информацию о Проекте GNU.

# 8 Пометка и область

Существует множество команд Emacs, которые работают с произвольной непрерывной частью текущего буфера. Чтобы задать текст, над которым работают такие команды, вы устанавливаете *метку* в одном конце этого текста, и передвигаете точку в другой его конец. Текст, расположенный между точкой и меткой, называется *областью*. Если вы задействовали режим Transient Mark, Emacs выделяет область цветом (см. Раздел 8.2 [Transient Mark], с. 64).

Чтобы регулировать границы области, вы можете двигать точку или метку. Не имеет значения, которая из них установлена первой в хронологическом порядке, или какая появляется первой в тексте. Как только метка установлена, она остается до тех пор, пока не будет установлена снова в другом месте. Каждый буфер имеет свою собственную метку, таким образом, когда вы возвращаетесь в буфер, который выбирался прежде, он сохраняет ту же самую метку, что и прежде.

Многие команды, которые вставляют текст, например `C-y` (`yank`) и `M-x insert-buffer`, располагают метку в одном из концов вставленного текста, противоположном тому, где помещается точка, таким образом, область содержит только что вставленный текст.

Помимо ограничения области, пометка также полезна для запоминания места, к которому вы, возможно, захотите вернуться. Чтобы сделать эту возможность более полезной, Emacs запоминает 16 предыдущих позиций метки в *списке пометок*.

## 8.1 Установка метки

Здесь приводятся некоторые команды для установки метки:

`C-`⟨SPC⟩ Установить метку в точке (`set-mark-command`).

`C-@` То же самое.

`C-x C-x` Поменять местами метку и точку (`exchange-point-and-mark`).

`Drag-Mouse-1`
Установить точку и пометить текст, по которому вы провели.

`Mouse-3` Установить метку в точке и переместить точку к позиции, на которой вы щелкнули. (`mouse-save-then-kill`).

Например, если вы хотите преобразовать часть буфера так, чтобы он весь был набран прописными буквами, вы можете использовать команду `C-x C-u` (`upcase-region`), которая оперирует с текстом в области. Вы можете сначала отправится в начало текста, который должен быть написан прописными буквами, набрать `C-`⟨SPC⟩, чтобы установить там метку, передвинуться в конец текста и затем набрать `C-x C-u`. Либо вы можете установить метку в конце текста, сдвинуться в начало и затем набрать `C-x C-u`.

Наиболее общий способ установки метки — делать это с помощью команды `C-`⟨SPC⟩ (`set-mark-command`). Так устанавливается пометка в позиции точки. Затем вы можете отодвинуть точку, оставляя метку на месте.

Есть два способа установить метку с помощью мыши. Вы можете провести мышью с нажатой первой кнопкой по отрезку текста; это поместит точку там, где вы отпустили кнопку мыши, и установит метку на другом конце этого отрезка. Или вы можете щелкнуть третьей кнопкой мыши, что устанавливает метку в точке (как `C-`⟨SPC⟩) и затем перемещает точку (как `Mouse-1`). Оба этих способа помимо установки метки копируют область в список уничтожений; такое поведение соответствует поведению других оконных приложений, но если вы не хотите изменять список уничтожений, вы должны использовать для установки метки команды клавиатуры. См. Раздел 17.1 [Команды мыши], с. 147.

На обычных терминалах есть только один курсор, так что Emacs не может показать вам местонахождение метки. Вы должны его помнить. Обычный выход из ситуации — установить метку, а затем быстро ее использовать до того, как вы забудете, где она расположена. Но вы можете увидеть позицию метки с помощью команды `C-x C-x` (`exchange-point-and-mark`), которая устанавливает метку туда, где была точка, а точку туда, где была метка. Положение области остается неизменным, а курсор и точка находятся теперь в предыдущей позиции метки. В режиме Transient Mark эта команда активизирует пометку.

`C-x C-x` также полезна, когда вы удовлетворены положением точки, но хотите передвинуть другой конец области (где находится метка); сделайте `C-x C-x` для того, чтобы установить точку на место метки, и затем вы можете двигать метку. Используя `C-x C-x` во второй раз, если это необходимо, вы установите метку в новой позиции, а точку вернете в ее прежнюю позицию.

В ASCII не существует знака `C-`⟨SPC⟩; когда вы набираете ⟨SPC⟩, удерживая ⟨CTRL⟩ в нажатом положении, на большинстве обычных терминалов вы получите знак `C-@`. Это и есть ключ, фактически привязанный к `set-mark-command`. Но если вы не настолько неудачливы, чтобы получить терминал, где набирая `C-`⟨SPC⟩, вы не получаете `C-@`, вы можете думать об этом знаке как о `C-`⟨SPC⟩. Под X, `C-`⟨SPC⟩ на самом деле является отдельным знаком, но привязан он все равно к `set-mark-command`.

## 8.2 Режим Transient Mark

Emacs может подсвечивать текущую область, используя X Windows. Но обычно он этого не делает. Почему?

Подсветка области обычно не работает в Emacs хорошо, потому что как только вы поставили метку, *всегда* есть область (в этом буфере). А постоянная подсветка области раздражала бы.

Вы можете включить подсветку помеченной области, задействовав режим Transient Mark. Это более жесткий режим действий, в котором область "существует" только временно, так что вы должны задавать область для каждой команды, которая ее использует. В режиме Transient Mark большую часть времени области нет; таким образом, подсветка области, когда она существует, удобна.

Чтобы включить режим Transient Mark, выполните команду `M-x transient-mark-mode`. Эта команда переключает данный режим, а значит вы можете повторить ее для выключения этого режима.

Вот подробности о режиме Transient Mark:

- Чтобы установить метку, введите `C-`⟨SPC⟩ (`set-mark-command`). Это сделает метку активной; при перемещении точки вы увидите, что область увеличивается или уменьшается.
- Команды мыши для задания пометки также делают ее активной. То же делают и команды клавиатуры, чья цель — установить область, включая `M-@`, `C-M-@`, `M-h`, `C-M-h`, `C-x C-p` и `C-x h`.
- Когда метка активна, вы можете выполнять команды, действующие на область, такие как уничтожение, создание отступа или запись в файл.
- Любое изменение буфера, например вставка или удаление знака, деактивизирует пометку. Это означает, что последующие команды, действующие на область, получат ошибку и не смогут работать. Вы можете сделать область снова активной, введя `C-x C-x`.
- Команды вроде `M->` и `C-s`, которые "оставляют метку" помимо какой-то другой основной цели, не активизируют новую метку. Вы можете активизировать новую область, выполнив `C-x C-x` (`exchange-point-and-mark`).
- `C-s` не изменяет метку, если она активна.

- Выход с помощью `C-g` деактивизирует метку.

Для подсветки области используется начертание `region`; вы можете настроить способ подсветки области, изменив это начертание. См. Раздел 31.2.2.3 [Настройка начертаний], с. 347.

Когда несколько окон показывают один и тот же буфер, они имеют разные области, потому что у них могут быть разные значения точки (хотя все они имеют одинаковую позицию метки.) Обычно только выбранное окно подсвечивает область (см. Глава 16 [Окна], с. 141). Однако, если переменная `highlight-nonselected-windows` не равна `nil`, то каждое окно подсвечивает свою область (если включен режим Transient Mark, и метка в буфере этого окна активна).

Когда режим Transient Mark не включен, каждая команда, устанавливающая метку, также и активизирует ее, но ничто не деактивизирует метку.

Если в режиме Transient Mark переменная `mark-even-if-inactive` не равна `nil`, то команды могут использовать метку и область, даже если они не активны. Подсветка области появляется и исчезает так же, как обычно в режиме Transient Mark, но метка на самом деле не исчезает, когда исчезает подсветка.

Режим Transient Mark также известен как “режим Zmacs”, потому что редактор Zmacs на MIT Lisp Machine обращался с пометкой похожим образом.

## 8.3 Работа с областью

Как только вы создали область, и метка активна, вы можете делать различные вещи с текстом этой области:

- Уничтожить его с помощью `C-w` (см. Раздел 9.1 [Уничтожение], с. 69).
- Записать его в регистр с помощью `C-x r s` (см. Глава 10 [Регистры], с. 77).
- Записать его в буфер или файл (см. Раздел 9.3 [Накопление текста], с. 73).
- Преобразовать регистр с помощью `C-x C-l` или `C-x C-u` (см. Раздел 21.6 [Преобразование регистра букв], с. 189).
- Сделать отступ с помощью `C-x` ⟨TAB⟩ или `C-M-\` (см. Глава 20 [Отступы], с. 177).
- Заполнить его как текст с помощью `M-x fill-region` (см. Раздел 21.5 [Заполнение], с. 185).
- Распечатать с помощью `M-x print-region` (см. Раздел 30.4 [Распечатка], с. 331).
- Вычислить его как Лисп-код с помощью `M-x eval-region` (см. Раздел 23.8 [Вычисление Лиспа], с. 254).

Большинство команд, которые оперируют с текстом в области, в своем имени содержат слово `region`.

## 8.4 Команды для пометки текстуальных объектов

Здесь представлены команды для помещения точки и метки вокруг текстуальных объектов, таких как слово, список, абзац или страница.

`M-@` Установить метку после конца следующего слова (`mark-word`). Эта и следующая команды не передвигают точку.

`C-M-@` Установить метку после конца следующего Лисп-выражения (`mark-sexp`).

`M-h` Установить область вокруг абзаца (`mark-paragraph`).

`C-M-h` Установить область вокруг текущего определения функции Лиспа (`mark-defun`).

`C-x h` Установить область вокруг всего текущего буфера (`mark-whole-buffer`).

`C-x C-p` Установить область вокруг текущей страницы (`mark-page`).

`M-@` (`mark-word`) устанавливает метку в конце следующего слова, а `C-M-@` (`mark-sexp`) устанавливает ее в конец следующего выражения Лиспа. Эти команды обрабатывают свои аргументы так же, как `M-f` и `C-M-f`.

Другие команды устанавливают как метку, так и точку, чтобы ограничить объект в буфере. Например, `M-h` (`mark-paragraph`) передвигает точку в начало абзаца, который окружает точку или следует за ней, и устанавливает метку в конце этого абзаца (см. Раздел 21.3 [Абзацы], с. 183). Эта команда подготавливает область, чтобы вы могли сделать отступы, переключить регистр или уничтожить весь абзац.

`C-M-h` (`mark-defun`) также устанавливает точку впереди, а метку после текущего или следующего определения функции (см. Раздел 22.4 [Определения функций], с. 208). `C-x C-p` (`mark-page`) устанавливает точку перед текущей страницей, а метку — в конце страницы (см. Раздел 21.4 [Страницы], с. 184). Пометка идет после ограничителя страницы (чтобы включить и его), точка идет после ограничителя предыдущей страницы (чтобы исключить его). Числовой аргумент задает более позднюю страницу (если аргумент положителен) или более раннюю (если он отрицателен) вместо текущей страницы.

Наконец, `C-x h` (`mark-whole-buffer`) помечает весь буфер как область, устанавливая точку в начало, а метку в конец.

В режиме Transient Mark все эти команды активизируют метку.

## 8.5 Список пометок

Помимо ограничения области, пометка служит также для запоминания места, в которое вы хотели бы вернуться. Чтобы сделать эту возможность более полезной, Emacs запоминает 16 предыдущих позиций метки в *списке пометок*. Большинство команд, которые устанавливают метку, вставляют старую метку в этот список. Чтобы вернуться к отмеченной позиции, используют `C-u C-`⟨SPC⟩ (или `C-u C-@`), это команда `set-mark-command`, которой дан числовой аргумент. Она передвигает точку туда, где была пометка, и восстанавливает метку из списка прежних пометок. Так, повторное использование этой команды передвигает точку ко всем старым пометкам в списке, к одной за другой. Метки, через которые вы проходите, попадают в конец списка, таким образом ни одна из них не пропадает.

Каждый буфер имеет свой собственный список пометок. Все команды редактирования используют список пометок текущего буфера. В частности, `C-u C-`⟨SPC⟩ всегда остается в том же самом буфере.

Многие команды, передвигающие на большие расстояния, такие, например, как `M-<` (`beginning-of-buffer`), начинают с установки метки и записи старого значения в список пометок. Это сделано для того, чтобы вам было легче впоследствии вернуться назад. Поиски делают это за исключением тех случаев, когда они фактически не двигают точку. Вы можете видеть, когда какая-либо команда устанавливает метку, по тому, что она печатает в эхо-области сообщение ‘`Mark Set`’.

Если вы хотите возвращаться к одному и тому же месту снова и снова, список пометок может оказаться недостаточно удобным. В таком случае вы можете записать позицию в регистр для последующего получения (см. Раздел 10.1 [Хранение позиций в регистрах], с. 77).

Переменная `mark-ring-max` задает максимальное число элементов в списке пометок. Если уже имеется максимальное количество элементов, и заталкивается еще один, то последний из существующего списка сбрасывается. Повторение `C-u C-`⟨SPC⟩ прокручивает среди позиций, которые находятся в данный момент в списке.

Переменная `mark-ring` хранит сам список пометок как список помеченных объектов, где первым идет записанный самым последним. Данная переменная является локальной в каждом буфере.

## 8.6 Глобальный список пометок

Кроме обычного списка пометок, который принадлежит каждому буферу, в Emacs есть единый *глобальный список пометок*. В нем записывается последовательность буферов, в которых вы недавно устанавливали метки, чтобы вы могли вернуться к этим буферам.

При установке метки всегда создается вхождение в списке пометок текущего буфера. Если вы переключали буферы после предыдущей установки метки, позиция новой метки вносится также и в глобальный список пометок. В результате глобальный список пометок хранит последовательность буферов, в которых вы побывали, и место в каждом из них, где вы поставили метку.

Команда `C-x C-`⟨SPC⟩ (`pop-global-mark`) переводит к буферу и позиции последнего вхождения в глобальном списке пометок. Она также проматывает этот список, чтобы последовательное применение команды `C-x C-`⟨SPC⟩ перемещало вас ко все более ранним буферам.

# 9 Уничтожение и перемещение текста

*Уничтожение* означает стирание текста и запись его в *список уничтожений*, из которого он может быть получен с помощью *восстановления*. Некоторые системы, ставшие недавно популярными, используют для обозначения этих операций термины “вырезка” и “вставка”.

Самый распространенный способ переноса или копирования текста в Emacs — сначала уничтожить его, а потом восстановить в одном или нескольких местах. Это очень надежно, так как все уничтоженные за последнее время куски текста запоминаются. И это удобно, так как многочисленные команды для уничтожения синтаксических единиц могут быть также использованы и для их перемещения. Но есть и другие способы копирования текста для особых целей.

Emacs хранит единый для всех буферов список уничтожений, так что вы можете уничтожить текст в одном буфере и восстановить его в другом.

## 9.1 Удаление и уничтожение

Большинство команд, которые стирают текст из буфера, сохраняют его в списке уничтожений, чтобы вы могли перенести или скопировать его в другие части буфера. Эти команды известны как команды *уничтожения*. Остальные команды, стирающие текст, не записывают его в список уничтожений; они известны как команды *удаления*. (Такое отличие делается только для стирания текста в буфере). Если вы уничтожили или удалили текст по ошибке, вы можете использовать команду `C-x u` (`undo`) для отмены изменений (см. Раздел 4.4 [Отмена], с. 37).

Команды удаления включают `C-d` (`delete-char`) и ⟨DEL⟩ (`delete-backward-char`), которые удаляют только один знак за один раз, и те команды, которые удаляют только пробелы или ограничители новой строки. Команды, которые могут уничтожить существенное количество нетривиальных данных, обычно являются командами уничтожения. Имена команд и их описания используют слова ‘`kill`’ и ‘`delete`’, чтобы пояснять что они делают.

### 9.1.1 Удаление

`C-d` Удалить следующий знак (`delete-char`).

⟨DEL⟩ Удалить предыдущий знак (`delete-backward-char`).

`M-\` Удалить все пробелы и табуляцию вокруг точки (`delete-horizontal-space`).

`M-`⟨SPC⟩ Удалить пробелы и табуляцию вокруг точки, оставляя один пробел (`just-one-space`).

`C-x C-o` Удалить пустые строки вокруг текущей строки (`delete-blank-lines`).

`M-^` Объединить две строки, удаляя находящийся между ними ограничитель новой строки и любой отступ, следующий за ним (`delete-indentation`).

Самые основные команды удаления — это `C-d` (`delete-char`) и ⟨DEL⟩ (`delete-backward-char`). `C-d` удаляет знак после точки, над которым находится курсор. Точка не передвигается. ⟨DEL⟩ удаляет знак перед курсором и передвигает точку назад. Ограничитель новой строки может быть удален точно также, как и любой другой знак в буфере. Удаление ограничителя новой строки объединяет две строки. Фактически `C-d` и ⟨DEL⟩ не всегда являются командами удаления; если им дать аргумент, они уничтожают, так как этим способом они могут стереть уже более одного знака.

Другие команды удаления — это те, что удаляют только пробельные знаки: пробелы, табуляцию и ограничители новых строк. `M-\` (`delete-horizontal-space`) удаляет все

пробелы и символы табуляции перед и после точки. M-⟨SPC⟩ (just-one-space) делает то же самое, но оставляет одиночный пробел после точки, независимо от количества пробелов, существовавших прежде (даже если оно было равно нулю).

C-x C-o (delete-blank-lines) уничтожает все пустые строки после текущей строки. Если текущая строка пустая, то также уничтожает все пустые строки, предшествующие текущей строке (оставляя одну пустую строку, текущую).

M-^ (delete-indentation) объединяет текущую строку и предшествующую ей, удаляя ограничитель новой строки и все окружающие пробелы, обычно оставляя одиночный пробел. См. Глава 20 [Отступы], с. 177.

### 9.1.2 Уничтожение строк

C-k Уничтожить остаток строки, либо одну или несколько строк (kill-line).

Простейшей командой уничтожения является C-k. Если она дается в начале строки, то уничтожает весь текст на строке, оставляя ее пустой. Если команда дана на пустой строке, то уничтожает всю строку, включая ее ограничитель. Чтобы уничтожить непустую строку целиком, перейдите в ее начало и нажмите C-k дважды.

В более общем виде, C-k уничтожает все от точки вплоть до конца строки, если это не происходит в конце строки. В этом случае она уничтожает ограничитель новой строки, следующий за строкой, таким образом происходит слияние следующей строки с текущей. При решении вопроса о том, какой случай применять, невидимые пробелы и табуляция в конце строки игнорируются, так, если точка выглядит стоящей в конце строки, вы можете быть уверены, что C-k уничтожит перевод строки.

Если C-k будет присвоен положительный аргумент, она уничтожит ровно столько строк вместе со следующими за ними ограничителями (однако, текст на текущей строке перед точкой сохраняется). С отрицательным аргументом $-n$ она уничтожает $n$ строк, предшествующих текущей (вместе с текстом на текущей строке перед точкой). Таким образом, C-u - 2 C-k в начале строки уничтожает две предыдущие строки.

C-k с аргументом, равным нулю, уничтожает текст перед точкой на текущей строке.

Если переменная kill-whole-line не равна nil, C-k в самом начале строки уничтожает всю эту строку, включая последующий перевод строки. Эта переменная обычно равна nil.

### 9.1.3 Другие команды уничтожения

C-w Уничтожить область (от точки до метки) (kill-region).

M-d Уничтожить слово (kill-word). См. Раздел 21.1 [Слова], с. 181.

M-⟨DEL⟩ Уничтожить предыдущее слово (backward-kill-word).

C-x ⟨DEL⟩ Уничтожить назад до начала предложения (backward-kill-sentence). См. Раздел 21.2 [Предложения], с. 182.

M-k Уничтожить до конца предложения (kill-sentence).

C-M-k Уничтожить s-выражение (kill-sexp). См. Раздел 22.2 [Списки], с. 206.

M-z *знак* Уничтожить вплоть до следующего появления *знака* (zap-to-char).

C-w (kill-region) — очень распространенная команда уничтожения, которая уничтожает все между точкой и меткой. С помощью этой команды вы можете уничтожить любую непрерывную последовательность знаков, если сначала установите метку в одном ее конце и отправитесь в другой конец.

Удобный способ уничтожения — это уничтожение скомбинированное с поиском: M-z (zap-to-char) считывает знак и уничтожает от точки вплоть до следующего появления

этого знака (и включая его) в буфере. Числовой аргумент действует как счетчик повторов. Отрицательный аргумент означает поиск в обратную сторону и уничтожение текста перед точкой.

Могут уничтожаться другие синтаксические единицы: слова, с помощью `M-`⟨DEL⟩ и `M-d` (см. Раздел 21.1 [Слова], с. 181); s-выражения, с помощью `C-M-k` (см. Раздел 22.2 [Списки], с. 206); и предложения, с помощью `C-x` ⟨DEL⟩ и `M-k` (см. Раздел 21.2 [Предложения], с. 182).

Вы можете использовать команды уничтожения в буферах, доступных только для чтения. На самом деле они не изменяют буфер и подают звуковой сигнал, чтобы предупредить вас об этом, но они действительно копируют текст, который вы попытались уничтожить, в список уничтожений, так что вы можете восстановить его в других буферах. Большинство команд уничтожения передвигают точку по тексту, который они копируют таким способом, поэтому последовательные команды уничтожения, как обычно, создают единое вхождение в списке уничтожений.

## 9.2 Восстановление

*Восстановление* возвращает обратно текст, который был ранее уничтожен. Это то же самое, что в других системах называется “вставкой”. Обычный способ копирования или перемещения текста — уничтожить его, а затем восстановить один или несколько раз.

`C-y` Восстановить последний уничтоженный текст (`yank`).

`M-y` Заменить только что восстановленный текст предшествующим куском уничтоженного текста (`yank-pop`).

`M-w` Сохранить область как последний уничтоженный текст без фактического уничтожения (`kill-ring-save`).

`C-M-w` Добавить следующее уничтожение к последнему куску уничтоженного текста (`append-next-kill`).

### 9.2.1 Список уничтожений

Весь уничтоженный текст записывается в *кольцевой список уничтожений*, список блоков текста, который был уничтожен. Существует только один список уничтожений, используемый во всех буферах, таким образом вы можете уничтожить текст в одном буфере и восстановить его в другом. Это обычный способ перемещения текста из одного файла в другой. (См. Раздел 9.3 [Накопление текста], с. 73, для получения информации о других способах).

Команда `C-y` (`yank`) вновь вставляет текст, уничтоженный самым последним. Она оставляет курсор в конце текста, метка устанавливается в его начале. См. Глава 8 [Пометка], с. 63.

`C-u C-y` оставляет курсор перед текстом и устанавливает метку после него. Это происходит, только если аргумент задан с помощью просто `C-u`. Любой другой вид аргумента, включая `C-u` и цифры, обозначают восстановление уничтоженного ранее (см. Раздел 9.2.3 [Ранее уничтоженное], с. 72).

Если вы хотите скопировать блок текста, вы можете использовать `M-w` (`kill-ring-save`), которая копирует область в список уничтожений без удаления его из буфера. Это приблизительный эквивалент `C-w`, за которой следует `C-x u`, за исключением того, что `M-w` не изменяет историю восстановлений и не изменяет на время экран.

### 9.2.2 Добавление уничтожений

Обычно каждая команда уничтожения добавляет новый блок в список уничтожений. Однако, две или более команды уничтожения подряд объединяют текст в единый элемент, так что одиночная C-y возьмет весь его обратно таким, каким он был перед уничтожением.

Таким образом, если вы хотите восстановить текст как одно целое, вы не должны уничтожать весь этот текст одной командой; вы можете продолжать уничтожение строки за строкой или слова за словом до тех пор, пока не уничтожите весь текст полностью, и вы можете получить его обратно сразу целиком.

Команды, которые уничтожают текст вперед от точки, добавляют уничтоженное в конец предыдущего уничтоженного текста. Команды, которые уничтожают назад от точки, добавляют это в начало. Таким образом, любая последовательность смешанных команд уничтожения перед точкой и после нее объединяет весь уничтоженный текст в один элемент без переупорядочения. Числовой аргумент не прерывает последовательность добавления уничтожений. Например, предположим, что буфер содержит такой текст:

```
Это пример строки ⋆с образцом текста.
```

с точкой, показанной как ⋆. Если вы наберете M-d M-⟨DEL⟩ M-d M-⟨DEL⟩, уничтожая по очереди вперед и назад, то в конце получите '**пример строки с образцом**' в качестве одного вхождения в списке уничтожений и '**Это текста.**' в буфере. (Обратите внимание на двойной пробел, который вы можете очистить с помощью M-⟨SPC⟩ или M-q.)

Другой способ уничтожить тот же текст — переместиться назад на два слова с помощью M-b M-b и уничтожить все четыре слова вперед командой C-u M-d. Это дает такой же результат в буфере и в списке уничтожений. M-f M-f C-u M-⟨DEL⟩ уничтожает тот же текст, проходя все время назад; и опять результат будет тем же. Текст в элементе списка уничтожений всегда имеет тот же порядок, что он имел в буфере до того, как вы его уничтожили.

Если команда уничтожения отделена от предыдущей другими командами (не просто числовым аргументом), то она начинает новый элемент в списке уничтожений. Но вы можете заставить эту команду уничтожения добавлять текст к предыдущему элементу, набрав перед ней команду C-M-w (append-next-kill). C-M-w приказывает следующей команде, если это команда уничтожения, добавить уничтоженный ею текст к последнему уничтоженному тексту вместо того, чтобы начинать новый элемент. С помощью C-M-w вы можете уничтожить несколько отдельных кусков текста и накопить их для дальнейшего восстановления в одном месте.

Команда уничтожения, следующая после M-w, не добавляет к тексту, который M-w скопировала в список уничтожений.

### 9.2.3 Восстановление ранее уничтоженного

Для того чтобы вернуть уничтоженный текст, который уже не является последним уничтоженным, используйте команду M-y (yank-pop). Она берет прежде восстановленный текст, и заменяет его текстом более раннего уничтожения. Так, чтобы вернуть текст, предшествующий последнему уничтоженному, сначала используйте C-y, чтобы восстановить последнее уничтожение, а затем M-y, чтобы заменить его предыдущим. M-y может использоваться только после C-y или другой M-y.

M-y можно понимать в терминах указателя на "последнее восстановление", который указывает на элемент в списке уничтожения. Каждый раз, когда вы уничтожаете, указатель на "последнее восстановление" передвигается в последний созданный элемент в начале списка. C-y восстанавливает элемент, на который ссылается указатель "последнего восстановления". M-y двигает указатель на "последнее восстановление" к другому элементу, и текст в буфере соответственно изменяется. Достаточное количество команд M-y может переместить указатель к любому элементу в списке, таким образом, вы можете получить

любой элемент в буфере. Когда наконец указатель достигает последнего элемента списка, следующая команда `M-y` снова помещает его на первый элемент.

`M-y` двигает указатель на “последнее восстановление” по списку, но это не меняет порядок элементов, которые всегда идут от самого последнего уничтожения до самого старого, которое еще хранится.

`M-y` может получить числовой аргумент, который говорит, на сколько элементов вперед продвинуть указатель на “последнее восстановление”. Отрицательный аргумент двигает этот указатель по направлению к началу списка; от начала списка он двигает к последнему элементу и продолжает движение вперед оттуда.

Как только искомый текст помещен в буфер, вы можете прекратить запускать команды `M-y`, и текст останется там. Это просто копия элемента списка уничтожения, поэтому редактирование его в буфере не изменяет содержимое этого списка. До тех пор, пока не делается новое уничтожение, указатель на “последнее восстановление” остается в том же самом месте в списке уничтожений, таким образом, повторение `C-y` восстановит еще одну копию того же самого старого уничтожения.

Если вы знаете, сколько команд `M-y` необходимо набрать, чтобы найти интересующий вас текст, вы можете восстановить этот текст за один шаг, используя `C-y` с числовым аргументом. `C-y` с аргументом восстанавливает текст, записанный в списке уничтожения на заданное число элементов назад. Таким образом, `C-u 2 C-y` достает следующий за последним блок уничтоженного текста. Это эквивалентно `C-y M-y`. `C-y` с числовым аргументом начинает подсчет от указателя на “последнее восстановление” и устанавливает этот указатель на элемент, который эта команда восстанавливает.

Длина списка уничтожений управляется переменной `kill-ring-max`; не может быть записано больше блоков текста, чем определено этой величиной.

Действительное содержимое списка уничтожений хранится в переменной `kill-ring`; вы можете просмотреть все содержимое списка уничтожений с помощью команды `C-h v kill-ring`.

## 9.3 Накопление текста

Обычно мы копируем или переносим текст путем его уничтожения и восстановления, но существуют и другие способы, удобные для копирования одного блока текста во многие места или для копирования многих рассеянных блоков текста в одно место. Чтобы скопировать один блок во много мест, запишите его в регистр (см. Глава 10 [Регистры], с. 77). Здесь мы описываем команды для накопления разбросанных кусков текста в буфер или в файл.

`M-x append-to-buffer`
: Добавить область в заданный буфер после точки.

`M-x prepend-to-buffer`
: Добавить область в заданный буфер перед точкой.

`M-x copy-to-buffer`
: Копировать область в заданный буфер, удаляя старое содержимое буфера.

`M-x insert-buffer`
: Вставить содержимое заданного буфера в текущий буфер в точке.

`M-x append-to-file`
: Добавить область в конец заданного файла.

Для накопления текста в буфере используется `M-x append-to-buffer`. Она считывает имя буфера, а затем вставляет в этот буфер копию области. Если буфер с таким именем не существует, то `append-to-buffer` создаст его. Текст вставляется в то место в этом

буфере, где находится точка. Если вы использовали этот буфер для редактирования, скопированный текст вставляется в середину текста, где оказалась точка.

Точка в этом буфере остается в конце копируемого текста, таким образом, последовательное использование `append-to-buffer` накапливает текст в заданном буфере в том же самом порядке, в котором части были скопированы. Строго говоря, `append-to-buffer` не всегда добавляет к тексту, уже находящемуся в буфере — она добавляет, если точка находится в конце этого буфера. Однако, если `append-to-buffer` является единственной командой, которую вы используете для изменения буфера, точка всегда расположена в конце.

`M-x prepend-to-buffer` подобна `append-to-buffer` за исключением того, что точка в другом буфере остается перед скопированным текстом, таким образом, последовательное применение этой команды добавляет текст в обратном порядке. `M-x copy-to-buffer` действует так же, за исключением того, что любой существующий текст в другом буфере удаляется, так что в буфере остается лишь вновь скопированный текст.

Чтобы вернуть накопленный текст из другого буфера, используйте команду `M-x insert-buffer`; она также принимает имя буфера как аргумент. Эта команда вставляет копию текста из буфера с заданным именем в выбранный буфер. Или вы можете выбрать другой буфер для редактирования, возможно, перемещая впоследствии текст из него путем уничтожения. См. Глава 15 [Буферы], с. 135, для получения базовой информации о буферах.

Вместо накопления текста внутри буфера Emacs вы можете добавить текст непосредственно в файл с помощью команды `M-x append-to-file`, которая использует имя файла в качестве аргумента. Она добавляет текст области в конец заданного файла. Файл на диске изменяется сразу.

Вы должны использовать `append-to-file` только с файлами, к которым вы *не* обращаетесь из Emacs. Использование этой команды для файла, к которому обратились из Emacs, может изменить файл без ведома Emacs, что может привести к потере некоторых результатов вашего редактирования.

## 9.4 Прямоугольники

Команды для прямоугольников работают с прямоугольными областями текста: всеми знаками между определенной парой столбцов в определенном диапазоне строк. Эти команды предназначены для уничтожения прямоугольников, восстановления уничтоженных прямоугольников, их очистки, заполнения пробелами или текстом, или удаления. Команды для прямоугольников полезны для работы с текстом в многоколоночных форматах и для приведения текста к такому формату или извлечения из него.

Когда вам нужно задать прямоугольник для команды, которая будет с ним работать, вы делаете это, устанавливая метку в одном углу и точку в противоположном. Прямоугольник, описанный таким образом, называется *областью-прямоугольником*, так как вы управляете им почти так же, как и областью. Но помните, что данная комбинация значений точки и метки может быть интерпретирована и как область, и как прямоугольник в зависимости от команды, которая их использует.

Если точка и метка находятся на одном столбце, то прямоугольник, который они ограничивают, пуст. Если они находятся на одной строке, то прямоугольник имеет высоту в одну строку. Эта несимметричность между строками и столбцами происходит из того, что точка (и метка) располагается между двух столбцов, но внутри строки.

`C-x r k` Уничтожить текст области-прямоугольника, сохраняя его содержимое в качестве “последнего уничтоженного прямоугольника” (`kill-rectangle`).

`C-x r d` Удалить текст области-прямоугольника (`delete-rectangle`).

`C-x r y` Восстановить последний уничтоженный прямоугольник, помещая его верхний левый угол в точке (`yank-rectangle`).

`C-x r o` Вставить пустое место, заполняя пространство области-прямоугольника (`open-rectangle`). Предыдущее содержимое области-прямоугольника выталкивается вправо.

`M-x clear-rectangle`
Очистить область-прямоугольник, заменяя ее содержимое пробелами.

`M-x delete-whitespace-rectangle`
Удалить пробельные знаки в каждой строке заданного прямоугольника, начиная с его самого левого столбца.

`C-x r t` *строка* ⟨RET⟩
Вставить *строку* в каждую строку области-прямоугольника (`string-rectangle`).

Операции, работающие с прямоугольниками, делятся на два класса: команды удаления и вставки прямоугольников и команды для пустых прямоугольников.

Существует два способа избавиться от текста в прямоугольнике: вы можете сбросить (удалить) его или записать его как "последний уничтоженный" прямоугольник. Для этого используются две команды `C-x r d` (`delete-rectangle`) и `C-x r k` (`kill-rectangle`). В обоих случаях часть каждой строки, которая попала внутрь границ прямоугольника, удаляется, заставляя последующий текст в строке (если он существует) сдвигаться влево.

Заметьте, что "уничтожение" прямоугольника не есть уничтожение в обычном понимании; этот прямоугольник хранится не в списке уничтожений, а в специальном месте, которое в состоянии записать только самое последнее уничтожение прямоугольника. Это происходит из-за того, что восстановление прямоугольника настолько отличается от восстановления линейного текста, что для этого должны использоваться другие команды восстановления, и в этом случае трудно приписать смысл команде, выдающей более ранние уничтожения.

Чтобы вставить последний уничтоженный прямоугольник, наберите `C-x r y` (`yank-rectangle`). Восстановление прямоугольника — это противоположность уничтожения. Левый верхний угол задается положением точки. Туда помещается первая строка прямоугольника, вторая строка прямоугольника помещается в позиции точки, но строкой ниже, и так далее. Число затронутых строк определяется высотой записанного прямоугольника.

Вы можете превратить списки из одной колонки в списки из двух колонок, используя уничтожение и восстановление прямоугольников; уничтожьте вторую половину списка как прямоугольник и затем восстановите его рядом с первой строчкой списка. См. Раздел 30.9 [Two-Column], с. 336, другой способ редактировать двухколоночный текст.

Прямоугольники также могут быть скопированы в регистры и из регистров с помощью `C-x r r` *r* и `C-x r i` *r*. См. Раздел 10.3 [Регистры для прямоугольников], с. 78.

Есть две команды для работы с пустыми прямоугольниками: `M-x clear-rectangle`, чтобы расписать пробелами существующий текст, и `C-x r o` (`open-rectangle`), чтобы вставить пустой прямоугольник. Очистка прямоугольника эквивалентна его удалению с последующей вставкой на его место пустого прямоугольника такого же размера.

Команда `M-x delete-whitespace-rectangle` удаляет горизонтальное пустое пространство, начиная с определенного столбца. Это относится к каждой строке в прямоугольнике, а столбец задается левым краем прямоугольника. Правый край прямоугольника не имеет значения для этой команды.

Команда `C-x r t` (`M-x string-rectangle`) замещает прямоугольник заданной строкой (вставляя ее один раз в каждую строку). Ширина строки не обязана совпадать с шириной прямоугольника. Если ширина строки меньше, текст после прямоугольника смещается влево; если строка шире прямоугольника, текст после него смещается вправо.

# 10 Регистры

*Регистры* Emacs — это места, куда вы можете записать текст или позиции для дальнейшего использования. Текст или прямоугольник, однажды записанный в регистр, может быть скопирован в буфер один или несколько раз; позицию, записанную в регистр, можно один или несколько раз использовать для передвижения к ней точки.

Каждый регистр имеет имя, состоящее из одиночного знака. Регистр может хранить кусок текста, позицию, прямоугольник, конфигурацию окна или имя файла, но только что-то одно из перечисленного в каждый конкретный момент. Всякий раз, когда вы сохраняете что-то в регистре, оно остается там до тех пор, пока вы не поместите в этот регистр что-то еще. Чтобы узнать, что содержит регистр *r*, используйте `M-x view-register`.

`M-x view-register` ⟨RET⟩ *r*
: Выдать описание того, что содержит регистр *r*.

## 10.1 Запись позиций в регистры

Сохранение позиции записывает местоположение в буфере, так что вы можете вернуться туда позже. Перенос в записанную позицию заново выбирает буфер и передвигает точку в это место.

`C-x r` ⟨SPC⟩ *r*
: Записать положение точки в регистр *r* (`point-to-register`).

`C-x r j` *r* Перейти в позицию, записанную в регистре *r* (`jump-to-register`).

Чтобы записать текущее положение точки в регистр, выберите имя *r* и наберите `C-x r` ⟨SPC⟩ *r*. Регистр *r* сохраняет записанное таким образом положение до тех пор, пока вы не запишете в этот регистр что-то другое.

Команда `C-x j` *r* передвигает точку в позицию, записанную в регистре *r*. Регистр не затрагивается, он продолжает хранить ту же самую позицию. Вы можете перейти в одну и ту же позицию, используя один и тот же регистр, любое число раз.

Если вы используете `C-x r j` для перехода к записанной позиции, но буфер, в котором она была записана, уже уничтожен, `C-x r j` пытается снова создать буфер, обращаясь к тому же файлу. Конечно, это работает только для буферов, которые обращались к файлам.

## 10.2 Запись текста в регистры

Если вы захотите вставить копию одного и того же куска текста несколько раз, использование списка уничтожений может оказаться неудобным, так как каждое последующее уничтожение сдвигает кусок текста все дальше вглубь списка. Альтернативой этому методу является сохранение текста в регистре с последующим его восстановлением.

`C-x r s` *r* Копировать область в регистр *r* (`copy-to-register`).

`C-x r i` *r* Вставить текст из регистра *r* (`insert-register`).

`C-x r s` *r* записывает копию текста области в регистр с именем *r*. Запущенная с числовым аргументом, `C-x r s` *r* кроме того удаляет текст из буфера.

`C-x r i` *r* вставляет в буфер текст из регистра *r*. Обычно эта команда оставляет точку перед текстом, а метку располагает после него, но с числовым аргументом (`C-u`), наоборот, точку ставит после текста, а метку перед ним.

## 10.3 Запись прямоугольников в регистры

Регистр может содержать вместо линейного текста прямоугольник. Прямоугольники представляются в виде списка строк. См. Раздел 9.4 [Прямоугольники], с. 74, для получения основной информации по прямоугольникам и о том, как они определяются в буфере.

`C-x r r` *r* Копирует область-прямоугольник в регистр *r* (`copy-region-to-rectangle`). С числовым аргументом еще и удаляет его.

`C-x r i` *r* Вставляет прямоугольник, который записан в регистре *r* (если тот содержит прямоугольник) (`insert-register`).

Команда `C-x r i` *r* вставляет строку текста, если регистр содержит строку, и вставляет прямоугольник, если регистр содержит прямоугольник.

Смотрите также команду `sort-columns`, которую можно рассматривать как команду сортировки прямоугольника. См. Раздел 30.7 [Сортировка], с. 333.

## 10.4 Запись конфигурации окон в регистры

Вы можете записать в регистр конфигурацию окон в выбранном фрейме или даже конфигурацию всех окон во всех фреймах и позднее восстановить эту конфигурацию.

`C-x r w` *r* Записать состояние окон выбранного фрейма в регистр *r* (`window-configuration-to-register`).

`C-x r f` *r* Записать состояние всех фреймов, включая все их окна, в регистр *r* (`frame-configuration-to-register`).

Используйте `C-x r j` *r* для восстановления конфигурации окон или фреймов. Это та же команда, что используется для восстановления позиции курсора. Когда вы восстанавливаете конфигурацию фреймов, все существующие фреймы, не входящие в эту конфигурацию, становятся невидимыми. Если вы вместо этого хотите удалить эти фреймы, используйте `C-u C-x r j` *r*.

## 10.5 Хранение чисел в регистрах

Существуют команды для записи числа в регистр, вставки его в буфер в десятичном виде и для увеличения этого числа. Эти команды могут быть полезны в макросах клавиатуры (см. Раздел 31.3 [Макросы клавиатуры], с. 353).

`C-u` *число* `C-x r n` *рег*
Записать *число* в регистр *рег* (`number-to-register`).

`C-u` *число* `C-x r +` *рег*
Увеличить число в регистре *рег* на заданное *число* (`increment-register`).

`C-x r g` *рег*
Вставить число из регистра *рег* в буфер.

`C-x r g` — это та же команда, что используется для вставки в буфер содержимого регистров другого типа.

## 10.6 Запись имен файлов в регистры

Если вы часто обращаетесь к определенным именам файлов, вам будет удобно поместить эти имена в регистры. Вот код на Лиспе, используемый для записи имени файла в регистр:

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

Например,

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

помещает показанное имя файла в регистр ‘z’.

Чтобы обратиться к файлу, чье имя находится в регистре *r*, наберите `C-x r j` *r*. (Это та же команда, что используется для перехода к записанной позиции или восстановления конфигурации фреймов.)

## 10.7 Закладки

*Закладки* отчасти похожи на регистры тем, что они записывают позиции, к которым вы можете перейти. В отличие от регистров, они имеют длинные имена и сохраняются автоматически от одного сеанса Emacs до другого. Шаблонное применение закладок — запись места “где вы читали” в различных файлах.

`C-x r m` ⟨RET⟩
: Заложить закладку в текущем файле в позиции точки.

`C-x r m` *закладка* ⟨RET⟩
: Заложить в точке закладку с именем *закладка* (`bookmark-set`).

`C-x r b` *закладка* ⟨RET⟩
: Перейти к закладке с именем *закладка* (`bookmark-jump`).

`C-x r l`
: Перечислить все закладки (`list-bookmarks`).

`M-x bookmark-save`
: Сохранить все текущие значения закладок в файле закладок, используемом по умолчанию.

Как правило закладки используются для записи одной текущей позиции в каждом из нескольких файлов. Таким образом, команда `C-x r m`, устанавливающая закладку, по умолчанию использует в качестве имени закладки имя файла, к которому вы обращаетесь в данный момент. Если вы называете каждую закладку по имени файла, на который она указывает, то сможете удобно повторно обратиться к любому из этих файлов с помощью `C-x r b` и в то же время переместиться к позиции закладки.

Чтобы отобразить список всех ваших закладок в отдельном буфере, наберите `C-x r l` (`list-bookmarks`). Если вы переключитесь в этот буфер, вы сможете отредактировать определения закладок или дать им описания. Наберите в этом буфере `C-h m` для получения более подробных сведений о специальных командах редактирования в нем.

Когда вы уничтожаете Emacs, он предложит вам сохранить значения закладок в вашем файле закладок по умолчанию, ‘`~/.emacs.bmk`’, если вы изменяли какие-либо из этих значений. Вы также можете сохранить закладки в любое время с помощью команды `M-x bookmark-save`. Команды, работающие с закладками, автоматически загружают ваш файл закладок по умолчанию. Благодаря этим сохранениям и загрузкам закладки переходят от одного сеанса Emacs к другому.

Если вы установите переменную `bookmark-save-flag` в значение 1, то каждая команда, устанавливающая закладку, также будет записывать ваши закладки; при этом вы не потеряете значения закладок, даже если Emacs останавливается аварийно. (Это значение, если оно является числом, говорит, сколько изменений закладок проходит между сохранениями.)

Значения позиций в закладках сохраняются вместе с окружающим контекстом, так что `bookmark-jump` может найти правильную позицию, даже если файл был слегка изменен. Переменная `bookmark-search-size` говорит, сколько знаков контекста по обе стороны от позиции закладки нужно записывать.

Вот несколько дополнительных команд для работы с закладками:

`M-x bookmark-load` ⟨RET⟩ *имя-файла* ⟨RET⟩
: Загрузить файл с именем *имя-файла*, содержащий список значений закладок. Вы можете использовать эту команду, как и команду `bookmark-write`, для работы с другими файлами закладок помимо вашего файла закладок по умолчанию.

`M-x bookmark-write` ⟨RET⟩ *имя-файла* ⟨RET⟩
: Сохранить все текущие значения закладок в файле *имя-файла*.

`M-x bookmark-delete` ⟨RET⟩ *закладка* ⟨RET⟩
: Удалить закладку с именем *закладка*.

`M-x bookmark-insert-location` ⟨RET⟩ *закладка* ⟨RET⟩
: Вставить в буфер имя файла, на который указывает закладка *закладка*.

`M-x bookmark-insert` ⟨RET⟩ *закладка* ⟨RET⟩
: Вставить в буфер *содержимое* файла, на который указывает закладка *закладка*.

# 11 Управление изображением

Так как в окне помещается только часть большого буфера, Emacs старается показывать ту часть, которая может быть интересна. Команды управления изображением позволяют вам указать, какой именно фрагмент текста вы хотите видеть, и как его отображать.

## 11.1 Прокрутка

Если буфер содержит текст, который не умещается в пределах окна, отображающего этот буфер, то Emacs показывает некую непрерывную часть этого текста. Показанный раздел всегда содержит точку.

*Прокрутка* означает передвижение текста вверх или вниз в окне так, что становятся видимыми различные его части. Прокрутка вперед означает, что текст двигается вверх, а новый текст появляется снизу. Прокрутка назад двигает текст вниз, а новый текст появляется сверху.

Прокрутка происходит автоматически, если вы сдвигаете точку за верхнюю или нижнюю границу окна. Вы можете также явно затребовать прокрутку при помощи команд из этого раздела.

`C-l` Очистить экран и восстановить изображение, сдвигая по вертикали выбранное окно к центральной точке в пределах окна (`recenter`).

`C-v`
⟨NEXT⟩ Прокрутить вперед (на одно окно или на определенное число строк) (`scroll-up`).

`M-v`
⟨PRIOR⟩ Прокрутить назад (`scroll-down`).

*арг* `C-l` Прокрутить таким образом, что точка оказывается на строке *арг* (`recenter`).

`C-M-l` Прокрутить эвристически, чтобы вывести на экран полезную информацию (`reposition-window`).

Основной командой прокрутки является `C-l` (`recenter`) без аргумента. Она полностью очищает экран и восстанавливает изображение во всех окнах. Кроме того, она прокручивает выбранное окно таким образом, что точка располагается посередине от начала окна.

Команды прокрутки `C-v` и `M-v` позволяют вам двигать весь текст в окне вверх или вниз на несколько строк. `C-v` (`scroll-up`) с аргументом показывает вам заданное количество строк, находящихся за нижней границей окна, сдвигая текст и точку вверх вместе, как `C-l`. `C-v` с отрицательным аргументом показывает вам больше строк, находящихся за верхней границей окна. `M-v` (`scroll-down`) подобна `C-v`, но двигает в обратном направлении. Функциональные клавиши ⟨NEXT⟩ и ⟨PRIOR⟩ эквивалентны `C-v` и `M-v`.

Имена команд прокрутки основаны на направлении, в котором движется текст в окне. Таким образом, команда прокрутки вперед называется `scroll-up`, потому что она перемещает текст вверх по экрану.

Чтобы прочитать буфер по целому окну, используйте `C-v` без аргумента. Эта команда берет последние две строки, находящиеся внизу окна, и ставит их в начало, и за ними далее следует почти все окно строк, невидимых ранее. Если точка находилась в тексте, ушедшем за верхний край окна, то она передвигается в новую вершину окна. `M-v` без аргумента двигает текст в обратном направлении с аналогичным перекрыванием. Количество перекрывающихся строк при `C-v` или `M-v` управляется переменной `next-screen-context-line`, по умолчанию оно равно двум.

Некоторым пользователям нравится, когда команды прокрутки всего окна сохраняют точку на одной строке экрана. Чтобы включить такое поведение, установите переменную

`scroll-preserve-screen-position` не равной `nil`. Такой режим удобен для просмотра файлов путем прокрутки по целому экрану; если вы вернетесь на экран, с которого начали, точка вернется на начальную строку. Однако, этот режим неудобен, когда вы перемещаетесь на следующий экран, чтобы передвинуть туда точку.

Другой способ выполнить прокрутку — использовать команду `C-l` с числовым аргументом. `C-l` с заданным аргументом не очищает экран, она только прокручивает выбранное окно. С положительным аргументом *n* она перемещает текст так, чтобы установить точку на *n* строк вниз от вершины. Аргумент, равный нулю, устанавливает точку на самую верхнюю строку. Точка не двигается по отношению к тексту; скорее, текст и точка двигаются на экране жестко связанные вместе. `C-l` с отрицательным аргументом устанавливает точку на заданное число строк выше от конца окна. Например, `C-u - 1 C-l` устанавливает точку на нижней строке, а `C-u - 5 C-l` устанавливает ее на пять строк выше конца окна. Просто `C-u` в качестве аргумента в `C-u C-l` прокручивает точку к центру экрана.

Команда `C-M-l` (`reposition-window`) прокручивает текущее окно эвристически, так, чтобы вывести на экран полезную информацию. Например, в файле с программой на Лиспе эта команда пытается разместить на экране все текущее определение функции, если это возможно.

Прокрутка происходит автоматически, если во время просмотра текста точка ушла из видимой его части. Обычно автоматическая прокрутка центрирует точку в окне по вертикали. Однако, если вы установите переменную `scroll-conservatively` равной маленькому числу *n*, тогда если вы сдвинете точку за экран лишь немного — меньше, чем на *n* строк, — Emacs прокрутит текст ровно на столько, чтобы вернуть точку на экран. По умолчанию значение `scroll-conservatively` равно нулю.

Переменная `scroll-margin` определяет, насколько близко может подойти точка к вершине или к низу окна. Ее значение — это число экранных строк; если точка подходит на заданное число строк к вершине или к низу окна, Emacs заново центрирует это окно. По умолчанию `scroll-margin` равна 0.

## 11.2 Горизонтальная прокрутка

*Горизонтальная прокрутка* — это сдвиг всех строк в окне в сторону, так что некоторый текст около левого края становится не виден совсем.

`C-x <` Прокручивает текст текущего окна влево (`scroll-left`).

`C-x >` Прокручивает вправо (`scroll-right`).

Когда окно прокручивается по горизонтали, строки текста становятся усеченными, а не продолженными. (см. Раздел 4.8 [Строки продолжения], с. 40); знак ‘`$`’ появляется в первом столбце, если есть текст, усеченный слева, и в последнем столбце, если существует текст, усеченный справа.

Команда `C-x <` (`scroll-left`) с аргументом *n* прокручивает выбранное окно влево на *n* столбцов. Она перемещает часть начала каждой строки за левый край окна. Без аргумента, она прокручивает почти на полную ширину окна (без двух столбцов, если быть точным).

`C-x >` (`scroll-right`) прокручивает аналогично, но только вправо. Окно не может быть прокручено дальше вправо, если оно отображено нормально (каждая строка начинается с левого края окна). Попытка сделать это не будет иметь результата. Это значит, что вы не обязаны точно вычислять аргумент для `C-x >`; любое достаточное большое число восстановит нормальное изображение.

Вы можете потребовать автоматической горизонтальной прокрутки, включив режим Hscroll. Когда этот режим задействован, Emacs горизонтально прокручивает окно всякий раз, когда это необходимо для сохранения точки видимой и расположенной не слишком

далеко от левого или правого края. Команда для включения и выключения этого режима — `M-x hscroll-mode`.

## 11.3 Режим Follow

*Режим Follow* — это второстепенный режим, который делает так, что два окна, показывающие один и тот же буфер, прокручиваются как одно большое "виртуальное окно". Чтобы использовать режим Follow, перейдите во фрейм с одним окном, разбейте его на два примыкающих окна с помощью `C-x 3` и затем напечатайте `M-x follow-mode`. Теперь вы можете редактировать буфер в любом из двух окон или прокручивать любое из них; второе окно будет следовать изменениям.

Чтобы выключить режим Follow, напечатайте `M-x follow-mode` второй раз.

## 11.4 Выборочный показ

Emacs обладает способностью прятать строки, смещенные вправо больше чем на определенное количество столбцов (вы сами задаете их число). Вы можете использовать это для получения обзора части программы.

Чтобы спрятать строки, наберите `C-x $` (`set-selective-display`) с числовым аргументом *n*. После этого строки, имеющие по крайней мере *n* столбцов отступа, пропадут с экрана. Показателем их существования являются только три точки ('`...`'), появляющиеся в конце каждой видимой строки, за которой следует одна или более невидимых.

Команды `C-n` и `C-p` перемещаются сквозь спрятанные строки, как если бы их не было.

Спрятанные строки по-прежнему присутствуют в буфере, и большинство команд редактирования видят их как обычные, так что можно установить точку посередине спрятанного текста. Когда это происходит, курсор появляется в конце предыдущей строки после трех точек. Если точка находится в конце видимой строки перед завершащим ее знаком перевода строки, курсор появляется перед тремя точками.

Чтобы сделать всё видимым снова, наберите `C-x $` без аргумента.

Если вы установите переменную `selective-display-ellipses` равной `nil`, то три точки не будут появляться в конце строки, которая предшествует спрятанным строкам. Тогда не будет никакого видимого указания на наличие скрытых строк. Эта переменная при установке автоматически становится локальной.

## 11.5 Дополнительные возможности строки режима

Когда включен режим Line Number, в строке режима появляется номер текущей строки, где находится точка. Используйте для переключения этого режима команду `M-x line-number-mode`; обычно он включен. Номер строки появляется перед *поз*, позицией в процентах от объема буфера, с буквой '`L`' для указания на то, что это такое. См. Раздел 31.1 [Второстепенные режимы], с. 341, для дальнейшей информации о второстепенных режимах и о том, как применять эту команду.

Если буфер очень большой (больше, чем значение переменной `line-number-display-limit`), то номер строки не выводится. Emacs не подсчитывает номера строк, если буфер велик, потому что это было бы слишком медленно. Если вы сузили буфер (см. Раздел 30.8 [Сужение], с. 335), отображаемый номер строки отчитывается относительно доступной части буфера.

Вы также можете показать номер текущего столбца, включив режим Column Number. Он отображает номер текущего столбца после буквы '`C`'. Для переключения этого режима напечатайте `M-x column-number-mode`.

Emacs может показывать время и загрузку системы во всех строках режима. Чтобы задействовать эту возможность, напечатайте `M-x display-time`. Эти сведения, добавляемые к строке режима, обычно появляются после имени буфера перед именами режимов. Это выглядит так:

чч:*мм*pm *з.зз*

Здесь чч и *мм* обозначают часы и минуты, за которыми всегда следуют ‘`am`’ или ‘`pm`’. *з.зз* — это среднее число процессов, запущенных во всей системе. (Некоторые поля могут быть пропущены, если ваша операционная система не поддерживает их.) Если вы предпочитаете видеть время в двадцатичетырехчасовом формате, установите переменную `display-time-24hr-format` в значение `t`.

Если для вас есть почта, которую вы еще не читали, после уровня загрузки появляется слово ‘`Mail`’.

## 11.6 Как отображается текст

Печатные знаки ASCII (с восьмиричными кодами от 040 до 0176) отображаются в буферах Emacs как их собственные графические представления. То же и для многобайтных печатных знаков, не входящих в ASCII (с восьмиричными кодами выше 0400).

Некоторые управляющие знаки ASCII отображаются особым образом. Знак новой строки (восьмиричный код 012) отображается как начало новой строки. Знак табуляции (восьмиричный код 011) показывается продвижением до следующей позиции табуляции (обычно через каждые 8 столбцов).

Другие управляющие знаки ASCII обычно отображается как шапочка (‘`^`’), за которой следует неуправляющая версия знака; таким образом, control-A показывается как ‘`^A`’.

Не-ASCII-знаки от 0200 до 0377 отображаются с помощью восьмиричных управляющих последовательностей; например, знак с кодом 0243 (восьмиричным) выводится как ‘`\243`’. Однако, если вы включите отображение европейских алфавитов, то большинство этих знаков станут печатными не-ASCII-знаками и будут отображаться с использованием их графических представлений (в предположении, что ваш терминал поддерживает это). См. Раздел 18.12 [Однобайтные европейские знаки], с. 172.

## 11.7 Переменные управления изображением

Этот раздел содержит информацию только для настройки. Начинающим пользователям стоит его пропустить.

Переменная `mode-line-inverse-video` управляет тем, отражается ли строка режима в инверсном виде (в предположении, что терминал поддерживает это); `nil` значит, что это не делается. См. Раздел 1.3 [Строка режима], с. 25. Если вы зададите цвет шрифта для начертания `modeline`, и `mode-line-inverse-video` не равна `nil`, то по умолчанию цветом фона для этого начертания будет цвет простого текста. См. Раздел 17.13 [Начертания], с. 155.

Если переменная `inverse-video` не равна `nil`, Emacs пытается инвертировать все строки дисплея из их обычного состояния.

Если переменная `visible-bell` не `nil`, Emacs пытается мерцать экраном, когда обычно он подает звуковой сигнал. Эта переменная не действует, если ваш терминал не умеет мерцать экраном.

Когда вы снова входите в Emacs после прерывания, Emacs обычно очищает экран и перерисовывает все изображение. На некоторых терминалах, имеющих более одной страницы памяти, можно сделать такую запись termcap, чтобы строки ‘`ti`’ и ‘`te`’ (выводимые терминалу, когда в Emacs входят и выходят, соответственно) переключали страницы памяти так, чтобы использовать одну для Emacs, а вторую для другого вывода. Затем, вы

можете по желанию установить переменную `no-redraw-on-reenter` не равной `nil`; это велит Emacs предполагать, что страница экрана после повторного входа все еще содержит то, что Emacs туда записал в последний раз.

Переменная `echo-keystrokes` управляет отражением многознаковых ключей. Значение этой переменной — это выраженная в секундах длина паузы, требуемой для вызова эхо; если оно равно нулю, это значит, что эхо не будет. См. Раздел 1.2 [Эхо-область], с. 24.

Если переменная `ctl-arrow` равна `nil`, то управляющие знак будут показаны в буфере с помощью восьмиричных управляющих последовательностей, все, кроме ограничителя новой строки и табуляции. Изменение значения переменной `ctl-arrow` делает ее локальной для текущего буфера, до этого же момента действует значение по умолчанию. Значение по умолчанию равно `t`. См. раздел “Display Tables” в *The Emacs Lisp Reference Manual*.

Обычно знак табуляции показывается в буфере как пробел, который простирается до следующей позиции табуляции дисплея; остановки по табуляции происходят через интервал, равный восьми пробелам. Число пробелов в табуляции управляется переменной `tab-width`, которая становится локальной при ее изменении, так же, как и `ctl-arrow`. Отметим, что способ отображения символа табуляции в буфере никак не влияет на определение ⟨TAB⟩, как команды. Переменная `tab-width` должна иметь значение между 1 и 1000, включительно.

Если переменная `truncate-lines` не равна `nil`, то каждая строка текста занимает ровно одну строку на экране; если строка текста слишком длинна, показывается только умещающаяся часть. Если `truncate-lines` равна `nil`, то длинные строки текста отображаются как несколько строк на экране, столько, сколько нужно, чтобы показать весь текст строки. См. Раздел 4.8 [Строки продолжения], с. 40. Изменение значения `truncate-lines` делает ее локальной для текущего буфера; до этого момента действует значение по умолчанию. Это значение по умолчанию изначально равно `nil`.

Если переменная `truncate-partial-width-windows` не равна `nil`, она принуждает усечение длинных строк вместо их продолжения в любом окне уже полной ширины фрейма, несмотря на значение переменной `truncate-lines`. Для получения информации о примыкающих окнах, смотрите Раздел 16.2 [Разделение окон], с. 142. Смотрите также раздел “Display” в *The Emacs Lisp Reference Manual*.

Переменная `baud-rate` содержит скорость вывода терминала, насколько ее знает Emacs. Установка этой переменной не меняет действительную скорость передачи данных, но ее значение используется для вычислений, например, наполнения. Она также влияет на принятие решения о том, нужно ли прокручивать часть экрана или перерисовывать ее — даже при использовании оконной системы. (Мы сделали так, несмотря на то, что оконная система не имеет действительной “скорости вывода”, чтобы дать вам возможность настроить эти параметры.)

Вы можете настроить способ вывода каждого конкретного знака с помощью таблицы отображения. См. раздел “Display Tables” в *The Emacs Lisp Reference Manual*.

# 12 Поиск и замена

Как и в других редакторах, в Emacs есть команды для поиска случаев появления какой-нибудь строки. Основная команда поиска необычна тем, что она является *наращиваемой*; она начинает поиск до того, как вы закончили набор строки поиска. Существуют также команды и для ненаращиваемого поиска, более похожие на аналогичные команды в других редакторах.

Кроме обычной команды `replace-string`, которая находит все случаи появления одной строки и заменяет их другой, Emacs имеет более сложную команду замены, названную `query-replace`, которая запрашивает в интерактивном режиме, в каких случаях надо произвести замену.

## 12.1 Наращиваемый поиск

Наращиваемый поиск начинается, как только вы набрали первый знак строки поиска. По мере того, как вы набираете строку поиска, Emacs показывает вам, где эта строка (в том виде, в каком вы ее уже набрали) может быть найдена. Когда вы набрали достаточно знаков, чтобы определить желаемое место, вы можете остановиться. В зависимости от того, что вы собираетесь делать потом, вам может понадобиться, а может и не понадобиться прекратить поиск явно с помощью ⟨RET⟩.

`C-s` Наращиваемый поиск вперед (`isearch-forward`).

`C-r` Наращиваемый поиск в обратном направлении (`isearch-backward`).

`C-s` начинает наращиваемый поиск. `C-s` считывает знаки с клавиатуры и располагает курсор в первом месте появления знаков, которые вы набрали. Если вы наберете `C-s` и затем `F`, то курсор встанет справа после первой найденной ‘`F`’. Наберите `O`, и увидите, что курсор встал за первой найденной ‘`FO`’. После еще одной `O` курсор встанет за первой ‘`FOO`’, находящейся за местом, с которого вы начали поиск. На каждом шаге текст буфера, совпадающий со строкой поиска, подсвечивается, если терминал может это сделать; текущая строка поиска обновляется на каждом шаге в эхо-области.

Если вы сделали ошибку в наборе строки поиска, то вы можете сбросить знаки с помощью ⟨DEL⟩. Каждый ⟨DEL⟩ отменяет последний знак строки поиска. Этого не происходит до тех пор, пока Emacs не будет готов считать следующий вводимый знак; сначала знак, который вы хотите сбросить, должен быть либо найден, либо нет. Если же вы не хотите ждать, пока это произойдет, используйте `C-g` так, как описано ниже.

Когда вы будете удовлетворены достигнутым местом, вы можете набрать ⟨RET⟩, что остановит поиск, оставляя курсор там, куда его поместила команда поиска. Любая команда, не имеющая специального значения при поиске, также останавливает поиск и затем выполняется сама. Таким образом, набор `C-a` привел бы к выходу из поиска и затем передвинул бы курсор в начало строки. ⟨RET⟩ необходим только в том случае, если следующая команда, которую вы хотите набрать, является печатным знаком, ⟨DEL⟩, ⟨RET⟩ или другим управляющим знаком, имеющим особое значение во время работы поиска (`C-q`, `C-w`, `C-r`, `C-s`, `C-y`, `M-y`, `M-r` или `M-s`).

Иногда вы ищете слово ‘`FOO`’ и находите его, но это не то, что вам нужно. Было второе ‘`FOO`’, о котором вы забыли, находящееся перед тем, которое вы ищете. В этом случае наберите `C-s` еще раз, чтобы продвинуться к следующему появлению строки поиска. Это можно проделывать неограниченное число раз. Если вы проскочили, то можете отменить некоторые число знаков `C-s` с помощью ⟨DEL⟩.

После выхода из поиска вы можете снова искать ту же самую строку, просто набрав `C-s C-s`: первый `C-s` — это ключ, который запускает наращиваемый поиск, а второй `C-s` означает “повтор поиска”.

Чтобы вы могли снова использовать более ранние строки поиска, существует *список поиска*. Команды `M-p` и `M-n` передвигают по списку, чтобы вы могли подобрать нужную строку для повторного поиска. Эти команды оставляют выбранную строку поиска в минибуфере, где вы можете ее отредактировать. Для завершения редактирования и начала поиска наберите `C-s` или `C-r`.

Если ваша строка вообще не найдена, то эхо-область говорит '`Failing I-Search`'. Курсор располагается после того места, где Emacs нашел из вашей строки всё, что смог. Таким образом, если вы ищете '`FOOT`', а такой строки нет, вы можете увидеть курсор после '`FOO`' в слове '`FOOL`'. С этого места вы можете сделать несколько вещей. Если ваша строка неправильно набрана, вы можете что-то стереть из нее и исправить. Если вы довольны найденным местом, вы можете набрать ⟨RET⟩ или любую другую команду Emacs, чтобы "принять то, что предложил этот поиск", или вы можете набрать `C-g`, что уничтожит из строки поиска знаки, которые не были найдены ('`T`' в '`FOOT`'), оставляя те, что нашлись ('`FOO`' в '`FOOT`'). Второй `C-g` в этом месте отменяет поиск полностью, возвращая точку туда, где она была, когда поиск начался.

Если строка поиска содержит заглавную букву, то поиск производится с учетом регистра. Если вы удалите заглавные буквы из строки поиска, эта особенность исчезает. См. Раздел 12.6 [Поиск и регистр], с. 95.

Если поиск был неудачным и вы просите повторить его, набирая `C-s` еще раз, то он начинается снова с начала буфера. Повторение неудачного поиска в обратном направлении при помощи команды `C-r` начинает новый поиск с конца. Такой поиск называется *круговым*. Как только это произошло, в подсказке поиска появляется слово '`Wrapped`'. Если вы пройдете через точку, где начался поиск, это слово заменяется на '`Overwrapped`', что означает, что вы снова проходите через уже виденные вами совпадения.

Знак "выхода" `C-g` поступает во время поиска особым образом. Что именно он делает, зависит от статуса поиска. Если поиск нашел то, что вы хотели, и ожидает ввода, то `C-g` полностью отменяет поиск. Курсор возвращается туда, откуда вы начали поиск. Если `C-g` набирается, когда в строке поиска есть ненайденные знаки — Emacs все еще ищет их, или он не смог их найти — тогда эти ненайденные знаки сбрасываются из строки поиска. Сброс этих знаков делает поиск успешным, и он ждет дальнейшего ввода, таким образом, второй `C-g` отменит поиск полностью.

Чтобы найти символ перевода строки, введите `C-j`. Для поиска другого управляющего знака, такого как control-S или возврат каретки, вы должны отменить их специальное значение, набирая перед ними `C-q`. Эта функция `C-q` аналогична ее назначению как команды для вставки (см. Раздел 4.1 [Вставка текста], с. 35): она заставляет трактовать следующий знак так, как в этом контексте трактовался бы любой "обычный" знак. Вы также можете задать знак по его восьмиричному коду: введите `C-q` и затем последовательность восьмиричных цифр.

Вы можете изменить направление поиска на обратное при помощи `C-r`. Вам следует поступить так, если поиск оказался неудачным, потому что место, с которого вы его начали, находилось слишком близко к концу файла. Повторение `C-r` продолжает поиск следующих случаев появления в обратном порядке, а `C-s` начинает поиск опять вперед. `C-r` в поиске может быть отменена при помощи ⟨DEL⟩.

Если вы заранее знаете, что вам нужно вести поиск в обратном порядке, то чтобы начать поиск, вы можете использовать `C-r` вместо `C-s`, так как `C-r` также является ключом, запускающим команду (`isearch-backward`) для поиска в обратном порядке. Обратный поиск находит совпадения, которые расположены перед начальной точкой, так же как поиск вперед находит совпадения, начинающиеся после точки, где поиск начался.

Знаки `C-y` и `C-w` могут использоваться в наращиваемом поиске для захвата текста из буфера в строку поиска. Это делает удобным поиск другого случая появления того текста, который находится в точке. `C-w` копирует слово после точки в строку поиска, продвигая точку вперед через это слово. Следующая команда `C-s` для повторения поиска будет затем

искать строку, включающую это слово. `C-y` подобна `C-w`, только копирует в строку поиска весь остаток текущей строки. И `C-y`, и `C-w` преобразуют копируемый текст к нижнему регистру, если поиск сейчас ведется без учета регистра; таким образом поиск остается регистронезависимым.

Команда `M-y` копирует в строку поиска текст из списка уничтожений. Она использует тот же текст, который был бы восстановлен командой `C-y`. См. Раздел 9.2 [Восстановление], с. 71.

Когда вы выходите из наращиваемого поиска, метка устанавливается в то место, где точка *была* до начала поиска. Это удобно для возврата к этому месту. В режиме Transient Mark наращиваемый поиск устанавливает метку, не активизируя ее, если только метка уже не активна.

Чтобы настроить специальные знаки, которые понимает наращиваемый поиск, измените их привязки в таблице ключей `isearch-mode-map`. Для получения перечня привязок посмотрите документацию на `isearch-mode` с помощью `C-h f isearch-mode` ⟨RET⟩.

### 12.1.1 Наращиваемый поиск на медленном терминале

Наращиваемый поиск на медленных терминалах использует модифицированный способ отображения, который разработан так, чтобы занимать как можно меньше времени. Вместо показа буфера в каждом месте, до которого добрался поиск, он создает новое окно, состоящее из одиночной строки, и использует его для показа найденной строки. Это окно из одной строки вступает в игру, как только точка выходит за пределы текста, который уже находится на экране.

Когда вы прерываете поиск, однострочное окно убирается. Только в этот момент Emacs перерисовывает окно, в котором производился поиск, чтобы отобразить новое положение точки.

Такой стиль отображения используется, когда скорость терминала в бодах меньше или равна значению переменной `search-slow-speed`, чье начальное значение равно 1200.

Количество строк, показываемых при поиске на медленном терминале, управляется переменной `search-slow-window-lines`. Ее обычное значение равно единице.

## 12.2 Ненаращиваемый поиск

В Emacs также есть удобные команды ненаращиваемого поиска, которые требуют от вас полностью набрать строку поиска до начала работы.

`C-s` ⟨RET⟩ *строка* ⟨RET⟩
: Поиск заданной *строки*.

`C-r` ⟨RET⟩ *строка* ⟨RET⟩
: Поиск *строки* в обратном направлении.

Чтобы начать ненаращиваемый поиск, наберите сначала `C-s` ⟨RET⟩. Эта команда входит в минибуфер для считывания строки поиска; ограничьте эту строку с помощью ⟨RET⟩, и поиск начнется. Если строка не будет найдена, команда поиска выдает ошибку.

Способ работы `C-s` ⟨RET⟩ заключается в следующем: `C-s` запускает наращиваемый поиск, который специально запрограммирован так, что запускает ненаращиваемый поиск, если заданный вами аргумент является пустым. (Такой пустой аргумент в других случаях был бы бесполезен). `C-r` ⟨RET⟩ работает аналогично.

Однако, запрошенный с помощью `C-s` ⟨RET⟩ ненаращиваемый поиск не запускает непосредственно `search-forward`. Первым делом проверяется, не будет ли следующим знаком `C-w`, что запустит поиск слов.

Прямой и обратный ненаращиваемый поиск осуществляются командами `search-forward` и `search-backward`. Эти команды могут быть привязаны к ключам обычным способом. Возможность их запуска через наращиваемый поиск имеет исторические причины и, помимо этого, существует для того, чтобы вам не нужно было находить для них подходящие последовательности ключей.

## 12.3 Поиск слов

Поиск по словам применяется для отыскания последовательности слов независимо от того, как эти слова разделены. Более подробно, вы набираете строку из нескольких слов, используя для их разделения одиночные пробелы, и эта строка может быть найдена, даже если в оригинале слова разделены несколькими пробелами, переводами строки, либо любыми знаками препинания.

Поиск слов полезен при редактировании печатных документов, подготовленных в программах для форматирования текста. Если вы редактируете, просматривая уже напечатанную, отформатированную версию, то вы не можете сказать, где прерывается строка в исходом файле. При помощи же поиска слова вы можете искать, не имея этой информации.

`C-s` ⟨RET⟩ `C-w` *слова* ⟨RET⟩
: Ищет *слова*, игнорируя пунктуацию между ними.

`C-r` ⟨RET⟩ `C-w` *слова* ⟨RET⟩
: Ищет *слова* в обратном направлении, игнорируя пунктуацию между ними.

Поиск слов — это специальный случай ненаращиваемого поиска, и он вызывается с помощью `C-s` ⟨RET⟩ `C-w`. За этим следует строка поиска, которая всегда должна быть ограничена ⟨RET⟩. Будучи ненаращиваемым, поиск не начинается до тех пор, пока аргумент не завершен. Этот поиск работает путем создания регулярного выражения и его поиска; смотрите Раздел 12.4 [Поиск регулярного выражения], с. 90.

Для обратного поиска слов используйте `C-r` ⟨RET⟩ `C-w`.

Прямой и обратный поиск слов реализован в командах `word-search-forward` и `word-search-backward`. Эти команды могут быть привязаны к ключам обычным способом. Возможность их запуска через наращиваемый поиск существует по историческим причинам и для того, чтобы вам не нужно было находить для них подходящие последовательности ключей.

## 12.4 Поиск регулярного выражения

*Регулярное выражение* (*regexp*, если кратко) — это образец, который обозначает набор строк, возможно, и неограниченный набор. В GNU Emacs вы можете искать следующее совпадение с регулярным выражением как наращиваемым способом, так и простым.

Наращиваемый поиск регулярного выражения производится набором `C-M-s` (`isearch-forward-regexp`). Эта команда считывает наращиваемую строку поиска, так же, как `C-s`, но трактует ее как регулярное выражение, а не ищет в тексте буфера точное совпадение. Каждый раз, когда вы добавляете текст в строку поиска, вы делаете регулярное выражение длиннее, и ищется уже новое регулярное выражение. Вызов `C-s` с префиксным аргументом (значение не играет роли) — это другой способ произвести прямой поиск регулярного выражения. Чтобы запустить поиск регулярного выражения в обратном направлении, используйте `C-M-r` (`isearch-backward-regexp`) или `C-r` с префиксным аргументом.

Все управляющие знаки, которые делают специальные вещи в рамках обыкновенного наращиваемого поиска, имеют те же самые функции и в наращиваемом поиске регулярного выражения. Набор `C-s` или `C-r` немедленно после начала поиска восстанавливает последнее регулярное выражение, использованное для наращиваемого поиска регулярного

выражения; это говорит о том, что наращиваемый поиск регулярного выражения и строки имеют независимые значения по умолчанию. Они также имеют раздельные списки поиска, доступ к которым вы можете получить с помощью `M-p` и `M-n`.

Если при наращиваемом поиске регулярного выражения вы наберете ⟨SPC⟩, он будет совпадать с произвольной последовательностью пробельных знаков, включая переводы строк. Если вам нужен только один пробел, введите `C-q` ⟨SPC⟩.

Обратите внимание, добавление знаков к регулярному выражению при наращиваемом поиске может вернуть курсор назад и начать поиск снова. Например, если вы искали '`foo`' и добавляете '`\|bar`', курсор вернется назад, если первый '`bar`' предшествовал первому '`foo`'.

Ненаращиваемый поиск регулярного выражения осуществляется функциями `re-search-forward` и `re-search-backward`. Вы можете запустить их с помощью `M-x`, или привязать их к ключам или вызывать через наращиваемый поиск регулярного выражения с помощью `C-M-s` ⟨RET⟩ и `C-M-r` ⟨RET⟩.

Если вы используете команды наращиваемого поиска регулярного выражения с префиксным аргументом, они производят обычный поиск строки, как `isearch-forward` и `isearch-backward`. См. Раздел 12.1 [Наращиваемый поиск], с. 87.

## 12.5 Синтаксис регулярных выражений

Регулярные выражения имеют синтаксис, в котором несколько знаков служат специальными конструкциями, а остальные — это *обыкновенные* знаки. Обыкновенный знак — это простое регулярное выражение, которое соответствует этому знаку и никакому больше. Специальными знаками являются '`$`', '`^`', '`.`', '`*`', '`+`', '`?`', '`[`', '`]`' и '`\`'. Любые другие знаки, появляющиеся в регулярном выражении, являются обыкновенными, если только им не предшествует '`\`'.

Например, '`f`' — это неспециальный знак, значит он обыкновенный, поэтому '`f`' — это регулярное выражение, которое соответствует строке '`f`' и никакой другой. (Оно *не* соответствует строке '`ff`'). Аналогично, '`o`' — это регулярное выражение, которое соответствует только '`o`'. (Когда различия в регистре игнорируются, эти регулярные выражения также совпадают с '`F`' и '`O`', но мы рассматриваем это как обобщение понятия "та же строка", а не как исключение.)

Любые два регулярных выражения *a* и *b* могут быть сцеплены. Результатом является регулярное выражение, совпадающее со строкой, в которой *a* соответствует некоторому началу этой строки, а *b* соответствует остатку строки.

В качестве простого примера мы можем сцепить регулярные выражения '`f`' и '`o`', чтобы получить регулярное выражение '`fo`', которое соответствует только строке '`fo`'. Пока все просто. Чтобы сделать что-то нетривиальное, вам необходимо использовать один из специальных знаков. Здесь представлен их перечень.

. (Точка) является специальным знаком, который соответствует любому одиночному знаку, за исключением перевода строки. Используя конкатенацию (сцепление), вы можете составить регулярное выражение, подобное '`a.b`', которое соответствует любой трехзнаковой строке, начинающейся с '`a`' и кончающейся на '`b`'.

* сама по себе не является конструкцией; это постфиксный оператор, который означает, что предыдущее регулярное выражение должно быть повторено столько раз, сколько это возможно. Таким образом, '`o*`' соответствует любому числу букв '`o`' (включая нуль).

'`*`' всегда относится к *наименьшему* возможному предыдущему выражению. Таким образом, '`fo*`' содержит повторяющуюся '`o`', а не '`fo`'. Оно совпадает с '`f`', '`fo`', '`foo`' и так далее.

Конструкция ‘*’ обрабатывается путем сопоставления с наибольшим количеством повторений, которое сразу может быть найдено. Затем продолжается сравнение с остатком шаблона. Если оно прошло неудачно, то происходит перебор с возвратом. Некоторые из совпадений с конструкцией с модификатором ‘*’ сбрасываются, чтобы дать возможность поиска соответствия для остатка структуры. Например, сравнивая ‘ca*ar’ со строкой ‘caaar’, ‘a*’ сначала ставится в соответствие со всеми тремя ‘a’, но остаток шаблона — это ‘ar’, а в этом случае для подбора остается только ‘r’, поэтому эта попытка неудачна. Следующий вариант — это поставить в соответствие с ‘a*’ только две буквы ‘a’. При таком выборе остаток регулярного выражения успешно соответствует строке.

`+` это такой же постфиксный оператор, как и ‘*’, за исключением того, что он требует, чтобы предшествующее ему выражение сопоставлялось по крайней мере один раз. Так например, ‘ca+r’ будет соответствовать строкам ‘car’ и ‘caaar’, но не строке ‘cr’, тогда как ‘ca*r’ соответствует всем трем строкам.

`?` постфиксный оператор, как и ‘*’, но он может соответствовать предшествующему выражению либо один раз, либо ни одного. Например, ‘ca?r’ будет соответствовать ‘car’ или ‘cr’ и ничему больше.

`[ ... ]` это *набор знаков*, который начинается ‘[’ и завершается ‘]’. В простейшем случае совпадающий набор формируют знаки между этими скобками.

Таким образом, ‘[ad]’ соответствует либо одной ‘a’, либо одному ‘d’, а ‘[ad]*’ соответствует любой строке, составленной просто из ‘a’ и ‘d’ (включая пустую строку), из всего этого следует, что ‘c[ad]*r’ соответствует ‘cr’, ‘car’, ‘cdr’, ‘caddaar’ и так далее.

Вы также можете включить в множество знаков интервалы, написав два знака, разделенные ‘-’; таким образом, ‘[a-z]’ соответствует любой строчной букве ASCII. Интервалы могут быть свободно перемешаны с отдельными знаками, как в ‘[a-z$%.]’, что соответствует любой строчной букве ASCII, или ‘$’, или ‘%’ или точке.

Заметим, что специальные знаки регулярных выражений внутри такого множества больше не являются специальными. Внутри знакового множества существуют совершенно другой набор специальных знаков: ‘]’, ‘-’ и ‘^’.

Чтобы включить в знаковый набор ‘]’, вы должны поставить его первым. Например, ‘[]a]’ соответствует ‘]’ или ‘a’. Чтобы включить ‘-’, напишите ‘-’ первым или последним знаком в наборе или поместите его после указания интервала. Таким образом, ‘[]-]’ соответствует ‘]’ и ‘-’.

Чтобы включить в набор знак ‘^’, пишите его где угодно, но не первым.

Если вы задаете интервал при поиске без учета регистра, вы должны либо написать оба конца интервала заглавными буквами, либо оба строчными, либо оба они не должны быть буквами. Поведение интервала с концами, заданными в разных регистрах, определено плохо и может быть изменено в будущих версиях Emacs.

`[^ ... ]` ‘[^’ начинает *дополнительный набор знаков*, который соответствует любому знаку, исключая описанные в нем. Таким образом, ‘[^a-z0-9A-Z]’ соответствует всем знакам, *исключая* буквы и цифры.

‘^’ не является специальным в наборе знаков, если он не стоит первым. Знак, следующий за ‘^’, трактуется так, как если бы он был первым (иными словами, ‘-’ и ‘]’ здесь не являются специальными).

Дополнительный набор знаков может соответствовать знаку новой строки, если он не упоминается как один из несовпадающих знаков. Это противоречит способу обработки регулярных выражений в таких программах, как `grep`.

`^` это специальный знак, который соответствует пустой строке, но только в начале строки сопоставляемого текста. В противном случае, сравнение не удастся. Таким образом, ‘`^foo`’ соответствует ‘`foo`’, которая встречена в начале строки.

`$` подобен ‘`^`’, но сравнение происходит только в конце строки. Таким образом, ‘`xx*$`’ соответствует строке из одного или более ‘`x`’ в конце строки.

`\` имеет две функции: отменяет особый смысл специальных знаков (включая ‘`\`’) и вводит дополнительные специальные конструкции.

Так как ‘`\`’ отменяет особый смысл специальных знаков, то ‘`\$`’ — это регулярное выражение, которое соответствует только ‘`$`’, а ‘`\[`’ — регулярное выражение, которое соответствует только ‘`[`’, и так далее.

Замечание: для исторической совместимости специальные знаки трактуются как обычные знаки, если они находятся в контексте, в котором их специальный смысл не имеет значения. Например, ‘`*foo`’ трактует ‘`*`’ как обыкновенный, так как не существует предыдущего выражения, на которое может подействовать ‘`*`’. Плохо быть зависимым от этого правила; лучше всегда явно отменять особый смысл специальных знаков независимо того, где они находятся.

В большинстве случаев ‘`\`’, за которым следует любой знак, соответствует только этому знаку. Однако, существует несколько исключений: двухзнаковые последовательности, начинающиеся с ‘`\`’, имеющие особый смысл. Второй знак в такой последовательности всегда обычный, когда встречается сам по себе. Здесь представлена таблица конструкций с ‘`\`’.

`\|` описывает альтернативу. Два регулярных выражения *a* и *b* с ‘`\|`’ между ними формируют выражение, которое соответствует любому из них в отдельности: либо *a*, либо *b*. Это работает так: сначала пробуется *a*, и если соответствие не найдено, пробуется *b*.

Таким образом, ‘`foo\|bar`’ соответствует либо ‘`foo`’, либо ‘`bar`’, но не другой строке.

‘`\|`’ применяется к самым большим охватывающим выражениям. Только охватывающие скобки ‘`\( ... \)`’ могут ограничить группирующую силу ‘`\|`’.

Существует возможность полного обратного восстановления для обработки многократных использований ‘`\|`’.

`\( ... \)` группирующая конструкция, которая служит для трех целей:

1. Чтобы отделить набор альтернатив ‘`\|`’ от других операций. Таким образом, ‘`\(foo\|mar\)x`’ соответствует либо ‘`foox`’, либо ‘`marx`’.
2. Чтобы ограничить сложное выражение для действия постфиксных операторов ‘`*`’, ‘`+`’ и ‘`?`’. Таким образом, ‘`ba\(na\)*`’ соответствует ‘`bananana`’ и так далее с любым (нулевым или большим) числом строк ‘`na`’.
3. Чтобы отметить соответствующую подстроку для будущей ссылки.

Это последнее применение не является следствием идеи ограничения группы; это отдельное свойство, которое определено как второе значение той же самой конструкции ‘`\( ... \)`’. На практике между этими двумя значениями не оказывается противоречий.

`\`*n* соответствует тексту, совпавшему с *n*-ным появлением конструкции ‘`\( ... \)`’.

После конца конструкции ‘`\( ... \)`’ сопоставление запоминает начало и конец текста, совпавшего с этой конструкцией. Затем, позднее в регулярном выражении, вы можете использовать ‘`\`’, за которым следует цифра *n*, чтобы сказать: “сопоставить с том же текстом, который совпал с *n*-ным появлением конструкции ‘`\( ... \)`’”.

Строкам, соответствующим первым девяти конструкциями ‘\( ... \)’, появляющимся в регулярном выражении, присваиваются номера от 1 до 9 в том порядке, в каком в регулярном выражении появились открывающие скобки. Конструкции от ‘\1’ до ‘\9’ могут использоваться для ссылки на текст конструкции ‘\( ... \)’ с этим номером.

Например, ‘\(.*\)\1’ соответствует любой строке, не содержащей знаков перевода строки, которая состоит из двух одинаковых половин. ‘\(.*\)’ соответствует первой половине, которая может быть любой, но ‘\1’, что идет следом, должна соответствовать точно такому же тексту.

Если для какой-нибудь конструкции ‘\( ... \)’ найдено более одного соответствия (что может легко произойти, если за ней следует ‘*’), то запоминается только последнее совпадение.

\‘ соответствует пустой строке, но только в начале буфера или строки, где происходит поиск.

\’ соответствует пустой строке, но только в конце буфера или строки, где происходит поиск.

\= соответствует пустой строке, но только в точке.

\b соответствует пустой строке, если эта конструкция находится в начале или конце слова. Таким образом, ‘\bfoo\b’ соответствует любому появлению ‘foo’ как отдельного слова. ‘bballs?\b’ соответствует ‘ball’ или ‘balls’ как отдельным словам.

‘\b’ находит соответствие в начале или конце буфера, независимо от того, какой текст идет далее.

\B соответствует пустой строке, если только она находится *не* в начале или конце слова.

\< соответствует пустой строке, если она находится в начале слова. ‘\<’ находит соответствие в начале буфера, но только если затем идет знак, являющийся частью слова.

\> соответствует пустой строке, если она находится в конце слова. ‘\>’ находит соответствие в конце буфера, но только если буфер завершается знаком, являющимся частью слова.

\w соответствует любому знаку, являющемуся частью слова. Какие именно это знаки, определяет синтаксическая таблица. См. Раздел 31.6 [Синтаксис], с. 366.

\W соответствует любому знаку, не являющемуся частью слова.

\s*c* соответствует любому знаку, чей синтаксис определяется кодом *c*. Здесь *c* — это знак, который представляет собой синтаксический код, например, это ‘w’ для части слова, ‘-’ для пробельных знаков, ‘(’ для открывающей скобки, и так далее. Вы можете обозначить пробельный знак (который может быть переводом строки) либо как ‘-’, либо одним пробелом.

\S*c* соответствует любому знаку, чей синтаксис не определяется кодом *c*.

Конструкции, имеющие отношение к словам и синтаксису, управляются установками в синтаксической таблице (см. Раздел 31.6 [Синтаксис], с. 366).

Далее представлено сложное регулярное выражение, используемое Emacs для распознавания конца предложения вместе с любыми пробельными знаками, которые идут следом. Оно дано в синтаксисе Лиспа, чтобы дать вам возможность отличить пробелы от знаков табуляции. В синтаксисе Лиспа, константная строка начинается и заканчивается двойными кавычками. ‘\"’ обозначает двойные кавычки как часть регулярного выражения, ‘\\’

обозначает обратную косую черту, ‘\t’ обозначает знак табуляции, а ‘\n’ — знак новой строки.

```
"[.?!][]\"')]*\\($\\|\t\\|  \\)[ \t\n]*"
```

Здесь последовательно содержатся четыре части: набор знаков, соответствующий точке, ‘?’ или ‘!’; набор знаков, соответствующий парным квадратным скобкам, кавычкам или круглым скобкам, повторяемым любое число раз; альтернатива, заключенная в скобки с обратными косыми чертами, которая соответствует концу строки, табуляции или двум пробелам; и набор знаков, соответствующий любым пробельным знакам, повторяющимся любое число раз.

Чтобы ввести это регулярное выражение интерактивно, вы напечатали бы ⟨TAB⟩, чтобы получить знак табуляции, и C-j, чтобы получить знак перевода строки. Вы также печатали бы одиночные обратные косые черты как есть, а не дублировали бы их в соответствии с синтаксисом Лиспа.

## 12.6 Поиск и регистр букв

Все виды наращиваемого поиска в Emacs обычно игнорируют регистр текста, в котором происходит поиск, если вы задали текст в нижнем регистре. Таким образом, если вы запросили поиск ‘foo’, то совпадениями считаются и ‘Foo’, и ‘foo’ . Регулярные выражения, и в частности наборы знаков, также включаются в это правило: ‘[aB]’ соответствовало бы ‘a’, или ‘A’, или ‘b’ или ‘B’.

Заглавная буква в любом месте строки наращиваемого поиска делает этот поиск регистрозависимым. Таким образом, поиск ‘Foo’ не найдет ‘foo’ или ‘FOO’. Это применяется также и к поиску регулярного выражения. Этот эффект исчезает, если вы удалили заглавные буквы из строки поиска.

Если вы установите переменную case-fold-search равной nil, все буквы должны будут совпадать точно, включая регистр. Эта переменная своя для каждого буфера; ее изменение затрагивает только текущий буфер, но существует значение по умолчанию, которое вы тоже можете изменить. См. Раздел 31.2.4 [Локальные переменные], с. 350. Эта переменная применяется также и к ненаращиваемому поиску, включая те его разновидности, которые осуществляются командами замены (см. Раздел 12.7 [Замена], с. 95) и командами поиска в истории минибуфера (см. Раздел 5.4 [История минибуфера], с. 49).

## 12.7 Команды замены

Глобальные команды поиска и замены не нужны в Emacs так часто, как в других редакторах[1], но они доступны. Кроме простой команды M-x replace-string, которая аналогична такой же команде в большинстве редакторов, существует команда M-x query-replace, которая для каждого появления образца спрашивает вас, надо ли его заменять.

Команды замены обычно работают с текстом от точки до конца буфера; однако, в режиме Transient Mark они действуют на область, когда метка активна. Все команды замены заменяют одну строку (или регулярное выражение) одной строкой замены. Можно выполнить параллельно несколько замен, используя команду expand-region-abbrevs (см. Раздел 24.3 [Расшифровка сокращений], с. 258).

### 12.7.1 Безусловная замена

M-x replace-string ⟨RET⟩ *строка* ⟨RET⟩ *новая-строка* ⟨RET⟩
: Заменяет каждое вхождение *строки* на *новую-строку*.

[1] В некоторых редакторах операции поиска и замены — это единственный удобный способ сделать одно изменение в тексте.

`M-x replace-regexp` ⟨RET⟩ *regexp* ⟨RET⟩ *новая-строка* ⟨RET⟩
Заменяет каждое совпадение с *regexp* на *новую-строку*.

Чтобы заменить каждый случай вхождения ‘`foo`’ после точки на ‘`bar`’, используется команда `M-x replace-string` с двумя аргументами ‘`foo`’ и ‘`bar`’. Замещение происходит только в тексте после точки, так, если вы хотите охватить весь буфер, вы должны сначала отправиться в его начало. Все экземпляры вплоть до конца буфера будут заменены; чтобы ограничиться заменой в части буфера, сузьте его до этой части перед выполнением замены (см. Раздел 30.8 [Сужение], с. 335). В режиме Transient Mark, если область активна, замена ограничена этой областью (см. Раздел 8.2 [Transient Mark], с. 64).

Когда вы выходите из `replace-string`, точка остается на месте последней замены. Значение точки в момент, когда была запущена команда `replace-string`, запоминается в списке пометок. `C-u C-`⟨SPC⟩ перемещает вас обратно.

Числовой аргумент ограничивает замену совпадениями, которые окружены ограничителями слов. Значение аргумента роли не играет.

### 12.7.2 Замена регулярных выражений

Команда `M-x replace-string` заменяет точные совпадения с одиночной строкой. Аналогичная команда `replace-regexp` замещает любое совпадение с заданным образцом.

В `replace-regexp`, *новая-строка* не обязательно должна быть константой: она может ссылаться на все или часть того, что соответствует регулярному выражению *regexp*. ‘`\&`’ в *новой-строке* означает полный замещаемый текст. ‘`\`*n*’, где *n* — это цифра, означает то, что было поставлено в соответствие *n*-ной заключенной в скобки группе в регулярном выражении *regexp*. Чтобы включить в новый текст знак ‘`\`’, вы должны ввести ‘`\\`’. Например,

`M-x replace-regexp` ⟨RET⟩ `c[ad]+r` ⟨RET⟩ `\&-safe` ⟨RET⟩

заменит (например) ‘`cadr`’ на ‘`cadr-safe`’ и ‘`cddr`’ на ‘`cddr-safe`’.

`M-x replace-regexp` ⟨RET⟩ `\(c[ad]+r\)-safe` ⟨RET⟩ `\1` ⟨RET⟩

делает обратное преобразование.

### 12.7.3 Команды замены и регистр букв

Если первый аргумент в команде замены набран в нижнем регистре, во время поиска вхождений для замены регистр игнорируется — при условии, что `case-fold-search` не равна `nil`. Если `case-fold-search` установлена в значение `nil`, регистр учитывается во всех типах поиска.

Кроме того, когда аргумент *новая-строка* весь или частично написан строчными буквами, команды замены пытаются сохранить образец использования регистра в каждом вхождении. Таким образом, команда

`M-x replace-string` ⟨RET⟩ `foo` ⟨RET⟩ `bar` ⟨RET⟩

заменяет ‘`foo`’ в нижнем регистре на ‘`bar`’ в нижнем регистре, ‘`FOO`’ в верхнем регистре на ‘`BAR`’, а ‘`Foo`’ с первой заглавной буквой на ‘`Bar`’. (Три эти альтернативы: все строчные буквы, все заглавные и первая заглавная — единственные варианты, которые может распознать `replace-string`.)

Если в строке подстановки использованы буквы верхнего регистра, то они остаются такими при каждой вставке этого текста. Если буквы верхнего регистра используются в первом аргументе, то второй аргумент всегда вставляется в том виде, в котором он дан, без изменения регистра. Аналогично, если переменная `case-replace` или `case-fold-search` установлена равной `nil`, замещение происходит без изменения регистра.

### 12.7.4 Замена с подтверждением

`M-%` *строка* ⟨RET⟩ *новая-строка* ⟨RET⟩
`M-x query-replace` ⟨RET⟩ *строка* ⟨RET⟩ *новая-строка* ⟨RET⟩
Заменяет некоторые вхождения *строки* на *новую-строку*.

`C-M-%` *regexp* ⟨RET⟩ *новая-строка* ⟨RET⟩
`M-x query-replace-regexp` ⟨RET⟩ *regexp* ⟨RET⟩ *новая-строка* ⟨RET⟩
Заменяет некоторые совпадения с *regexp* на *новую-строку*.

Если вы хотите заменить только некоторые экземпляры '`foo`' на '`bar`', но не все, вы не можете использовать обыкновенную `replace-string`. Вместо этого используется `M-%` (`query-replace`). Эта команда находит экземпляры '`foo`' один за другим, отображает каждый экземпляр и спрашивает вас, надо ли его заменять. Числовой аргумент говорит `query-replace`, что нужно рассматривать лишь те экземпляры, которые окружены знаками-разделителями слов. Эта команда сохраняет регистр так же, как и `replace-string`, при условии, что `case-replace` не равна `nil`, как это обычно и бывает.

За исключением запроса подтверждения, `query-replace` работает точно так же, как `replace-string`, а `query-replace-regexp` — как `replace-regexp`. Эта команда запускается при помощи `C-M-%`.

Когда вам показывают вхождение *строки* или совпадение с регулярным выражением *regexp*, вы можете набрать следующее:

⟨SPC⟩ чтобы заменить это вхождение на *новую-строку*.

⟨DEL⟩ чтобы перейти к следующему вхождению, не заменяя это.

`,` (Запятая)
чтобы заменить это вхождение и показать результат. Затем у вас запрашивают ввод еще одного знака, чтобы узнать, что делать дальше. Так как замена уже произведена, то ⟨DEL⟩ и ⟨SPC⟩ в этой ситуации эквивалентны; обе переходят к следующему вхождению.

Вы можете набрать в этом месте `C-r` (смотрите ниже), чтобы изменить замененный текст. Вы можете также набрать `C-x u`, чтобы отменить сделанную замену; эта команда выходит из `query-replace`, так что если вы хотите делать дальнейшие замены, вы должны использовать `C-x` ⟨ESC⟩ ⟨ESC⟩ ⟨RET⟩, чтобы запустить замену заново (см. Раздел 5.5 [Повтор], с. 50).

⟨RET⟩ чтобы выйти без осуществления дальнейших замен.

`.` (Точка) чтобы заменить этот экземпляр и затем выйти без продолжения поиска следующих вхождений.

`!` чтобы заменить все оставшиеся экземпляры без повторных запросов.

`^` чтобы вернуться к положению предыдущего вхождения (или к тому, что им было), если вы изменили его по ошибке. Это делается при помощи выталкивания из списка пометок. Можно использовать только один '`^`' подряд, так как во время работы `query-replace` хранится только одна предыдущая позиция замены.

`C-r` чтобы войти в новый уровень рекурсивного редактирования, в том случае, когда экземпляр нуждается скорее в редактировании, чем просто в замене его *новой-строкой*. Когда вы сделаете это, выйдите из этого уровня рекурсивного редактирования, набрав `C-M-c`, чтобы перейти к следующему вхождению. См. Раздел 30.12 [Рекурсивное редактирование], с. 338.

`C-w` чтобы удалить это вхождение и потом войти в новый уровень рекурсивного редактирования, как в `C-r`. Используйте рекурсивное редактирование для вставки текста и замены удаленного вхождения *строки*. Когда вы закончите, выйдите из этого уровня рекурсивного редактирования с помощью `C-M-c`, чтобы перейти к следующему вхождению.

`C-l` чтобы восстановить изображение экрана. Потом вы должны набрать еще один знак, чтобы указать, что делать с этим вхождением.

`C-h` чтобы просмотреть сообщение, резюмирующее эти варианты. Потом вы должны набрать еще один знак, чтобы указать, что делать с этим вхождением.

Некоторые другие знаки являются синонимами перечисленных выше: `y`, `n` и `q` эквивалентны ⟨SPC⟩, ⟨DEL⟩ и ⟨RET⟩.

Кроме этих знаков, любой другой выходит из `query-replace` и снова считывается как часть последовательности ключей. Таким образом, если вы напечатаете `C-k`, она выйдет из `query-replace` и уничтожит текст до конца строки.

Чтобы перезапустить `query-replace`, когда вы уже из нее вышли, используйте `C-x` ⟨ESC⟩ ⟨ESC⟩, которая повторит `query-replace`, так как она использовала минибуфер для чтения аргументов. См. Раздел 4.11 [Повторение], с. 43.

Смотрите также Раздел 28.9 [Преобразование имен файлов], с. 297, чтобы узнать о командах Dired для переименования, копирования или создания ссылок на файлы путем замены в их именах совпадений с регулярным выражением.

## 12.8 Другие команды поиска в цикле

Здесь представлены некоторые другие команды, которые находят совпадения с регулярными выражениями. Все они действуют от точки до конца буфера, и все они игнорируют при сопоставлении регистр, если образец не содержит заглавных букв, а `case-fold-search` отлична от `nil`.

`M-x occur` ⟨RET⟩ *regexp* ⟨RET⟩
: Выводит перечень, показывающий каждую строку буфера, которая содержит совпадение с *regexp*. Числовой аргумент задает число строк контекста, которые должны быть напечатаны перед и после каждой сравниваемой строки; значений по умолчанию — не печатать контекст. Чтобы ограничить поиск частью буфера, сузьтесь до этой части (см. Раздел 30.8 [Сужение], с. 335).

  Буфер ‘`*Occur*`’, в который записывается вывод, служит в качестве меню для поиска вхождений в их оригинальном контексте. Щелкните `Mouse-2` на вхождении, перечисленном в ‘`*Occur*`’, или поместите там точку и нажмите ⟨RET⟩; это переключит в буфер, где делался поиск, и переместит точку к оригиналу выбранного вхождения.

`M-x list-matching-lines`
: Синоним для `M-x occur`.

`M-x count-matches` ⟨RET⟩ *regexp* ⟨RET⟩
: Печатает число совпадений с *regexp* после точки.

`M-x flush-lines` ⟨RET⟩ *regexp* ⟨RET⟩
: Удаляет каждую строку, следующую после точки и содержащую совпадение с *regexp*.

`M-x keep-lines` ⟨RET⟩ *regexp* ⟨RET⟩
: Удаляет каждую строку, следующую после точки и *не* содержащую совпадение с *regexp*.

Кроме того, вы можете использовать из Emacs программу `grep` для поиска совпадений с регулярным выражением в группе файлов, а затем обратиться к найденным совпадениям последовательно или в произвольном порядке. См. Раздел 23.2 [Поиск с grep], с. 248.

# 13 Команды для исправления опечаток

В этой главе мы описываем команды, которые особенно полезны в тех случаях, когда вы обнаружили в вашем тексте ошибку сразу после того, как вы ее допустили, или если вы изменили ваши намерения во время сочинения текста.

Наиболее фундаментальная команда для исправления ошибочного редактирования — это команда отмены, `C-x u` или `C-_`. Эта команда отменяет одну команду (как правило), часть команды (в случае `query-replace`) или несколько самовставляющихся знаков подряд. Последовательное повторение команд `C-_` или `C-x u` отменяет все более ранние изменения до конца доступной информации отмены. См. Раздел 4.4 [Отмена], с. 37, для получения дальнейших сведений.

## 13.1 Уничтожение ваших ошибок

⟨DEL⟩ Удалить последний знак (`delete-backward-char`).

`M-`⟨DEL⟩ Уничтожить последнее слово (`backward-kill-word`).

`C-x` ⟨DEL⟩ Уничтожить все до начала предложения (`backward-kill-sentence`).

Символ ⟨DEL⟩ (`delete-backward-char`) — наиболее важная команда исправления. Она удаляет знак перед точкой. Когда ее используют после самовставляющейся знаковой команды, ее можно рассматривать как отмену этой команды. Но избегайте ошибочной мысли, что ⟨DEL⟩ — это универсальный способ отменить команду!

Когда ваша ошибка длиннее, чем пара знаков, то может оказаться более удобным использовать `M-`⟨DEL⟩ или `C-x` ⟨DEL⟩. `M-`⟨DEL⟩ уничтожает все знаки назад до начала последнего слова, а `C-x` ⟨DEL⟩ уничтожает назад до начала последнего предложения. `C-x` ⟨DEL⟩ особенно полезна в случае, когда вы меняете свое мнение о формулировке текста, который вы пишите. `M-`⟨DEL⟩ и `C-x` ⟨DEL⟩ записывают уничтоженный текст для восстановления с помощью `C-y` и `M-y`. См. Раздел 9.2 [Восстановление], с. 71.

`M-`⟨DEL⟩ часто полезна, даже когда вы набрали всего несколько ошибочных знаков, если вы знаете, что запутались и не уверены точно, что именно вы набрали. В этом случае вы не можете исправлять с помощью ⟨DEL⟩, не изучая экран, чтобы увидеть, что же вы сделали. Часто уничтожение всего слова и его повторный набор потребует меньших раздумий.

## 13.2 Перестановка текста

`C-t` Переставить два знака (`transpose-chars`).

`M-t` Переставить два слова (`transpose-words`).

`C-M-t` Переставить два сбалансированных выражения (`transpose-sexps`).

`C-x C-t` Переставить две строки (`transpose-lines`).

Распространенная ошибка перестановки двух знаков может быть исправлена, если они являются соседними, с помощью команды `C-t` (`transpose-chars`). Обычно `C-t` переставляет два знака, стоящие по обе стороны от точки. Когда она дается в конце строки, то вместо того, чтобы поменять местами последний знак символ и перевод строки, что было бы бесполезно, `C-t` переставляет последние два знака на строке. Так, если вы отловили вашу ошибку перестановки сразу, вы можете исправить ее просто с помощью `C-t`. Если вы отловили ее не так быстро, вы должны перевести курсор назад и установить его между двумя переставленными знаками. Если вы переставили пробел и последний знак в слове перед ним, то команды движения по словам служат удобным способом добраться до этого

места. Однако, обратный поиск (`C-r`) часто бывает самым лучшим способом. См. Глава 12 [Поиск], с. 87.

`M-t` (`transpose-words`) переставляет слово перед точкой со словом после нее. Она двигает точку вперед через слово, также перемещая вперед слово, предшествующее точке или содержащее ее. Знаки пунктуации между словами не двигаются. Например, '`FOO, BAR`' превращается в '`BAR, FOO`', а не в '`BAR FOO,`'.

`C-M-t` (`transpose-sexps`) — аналогичная команда для перестановки двух выражений (см. Раздел 22.2 [Списки], с. 206), а `C-x C-t` (`transpose-lines`) меняет местами строки. Они работают так же, как `M-t`, за исключением определения деления текста на синтаксические единицы.

Числовой аргумент для команд перестановки работает как счетчик повторений: он указывает команде перестановки передвинуть знак (слово, строку, выражение) перед точкой или содержащий ее через несколько других знаков (слов, строк, выражений). Например, `C-u 3 C-t` передвигает знак перед точкой вперед через три других знака. Это эквивалентно трехкратному повторению `C-t`. `C-u - 4 M-t` двигает слово перед точкой назад через четыре слова. `C-u - C-M-t` отменила бы действие простой `C-M-t`.

Числовому аргументу, равному нулю, придается специальное значение (так как в противном случае команды с нулевым счетчиком повторов должны были бы ничего не делать): переставить знак (слово, выражение, строку), оканчивающийся после точки, со знаком (словом, выражением, строкой), оканчивающимся после метки.

## 13.3 Изменение регистра

`M- M-l` Перевести последнее слово в нижний регистр. Отметим, что `Meta-` означает Meta-минус.

`M- M-u` Перевести последнее слово в верхний регистр.

`M- M-c` Перевести последнее слово в нижний регистр с первой заглавной буквой.

Набор слова в неправильном регистре — очень распространенная ошибка. Поэтому команды изменения регистра `M-l`, `M-u` и `M-c` обладают специальным свойством, когда используются с отрицательным аргументом: они не передвигают курсор. Как только вы увидите, что последнее слово набрано неправильно, вы можете просто изменить в нем регистр и продолжать набор. См. Раздел 21.6 [Регистр], с. 189.

## 13.4 Поиск и исправление орфографических ошибок

Этот раздел описывает команды для проверки правописания отдельного слова или части буфера. Эти команды работают с программой проверки правописания Ispell, которая не является частью Emacs.

`M-x flyspell-mode`
Включает режим Flyspell, который выделяет все неправильно написанные слова.

`M-$` Проверяет и исправляет написание слова в точке (`ispell-word`).

`M-`⟨TAB⟩ Завершает слово перед точкой, основываясь на орфографическом словаре (`ispell-complete-word`).

`M-x ispell-buffer`
Проверяет и исправляет написание всех слов в буфере.

`M-x ispell-region`
Проверяет и исправляет написание всех слов в области.

`M-x ispell-message`
: Проверяет и исправляет написание всех слов в черновике почтового сообщения, за исключением цитируемого материала.

`M-x ispell-change-dictionary` ⟨RET⟩ *словарь* ⟨RET⟩
: Перезапускает процесс Ispell, используя *словарь* в качестве словаря.

`M-x ispell-kill-ispell`
: Уничтожает подпроцесс Ispell.

Режим Flyspell предоставляет полностью автоматический способ проверить правописание во время редактирования в Emacs. Он работает путем проверки слов по мере того, как вы изменяете или вставляете их. Когда он находит нераспознанное слово, он выделяет его. Это не влияет на ваше редактирование, но когда вы видите выделенное слово, вы можете переместиться к нему и исправить. Чтобы включить этот режим в текущем буфере, наберите `M-x flyspell-mode`.

Когда режим Flyspell выделяет неправильно написанное слово, вы можете щелкнуть на этом слове `Mouse-2`, чтобы получить меню возможных исправлений и действий. Вы также можете исправить слово, отредактировав его вручную любым способом, который вам нравится.

Другие возможности Emacs по проверке правописания проверяют или ищут слова, когда вы даете явную команду для этого. Проверка всего или части буфера полезна, когда у вас есть текст, который был написан не в данном сеансе Emacs и может содержать любое число ошибок.

Для проверки орфографии в слове вокруг точки или после нее и, возможно, для его исправления, используйте команду `M-$` (`ispell-word`). Если слово написано неправильно, эта команда предложит вам несколько вариантов действий.

Чтобы проверить правописание во всем текущем буфере, запустите команду `M-x ispell-buffer`. Используйте `M-x ispell-region` для проверки только текущей области. Чтобы проверить орфографию в почтовом сообщении, которое вы пишите, используйте `M-x ispell-message`; она проверяет весь буфер, но не трогает материал, в котором сделан отступ, или который процитирован из других сообщений.

Каждый раз, когда эти команды встречают неправильно написанное слово, они спрашивают вас о дальнейших действиях. Они показывают список вариантов, обычно включающий несколько “похожих слов” — которые близки по написанию с проверяемым словом. Затем вы должны напечатать один знак. Вот допустимые варианты ответа:

⟨SPC⟩
: Пропустить это слово — продолжать считать это слово неправильным, но не изменять его здесь.

`r` *новое* ⟨RET⟩
: Заменить это слово (только в этом месте) на *новое*.

`R` *новое* ⟨RET⟩
: Заменить это слово на *новое* и выполнить `query-replace`, чтобы вы могли заменить его во всех других местах буфера, если хотите.

*цифра*
: Заменить это слово (только в этом месте) на одно из показанных похожих слов. Каждое похожее слово перечисляется с цифрой; чтобы выбрать его, наберите эту цифру.

`a`
: Принять неправильное слово — считать его правильным, но только в этом сеансе редактирования.

`A`
: Принять неправильное слово — считать его правильным, но только в этом сеансе редактирования и для этого буфера.

`i` Вставить это слово в ваш личный файл словаря, чтобы Ispell отныне считал его правильным даже в будущих сеансах.

`u` Вставить это слово в нижнем регистре в ваш личный файл словаря.

`m` Как `i`, но вы также можете указать сведения о завершении для словаря.

`l` *слово* ⟨RET⟩
Поискать в словаре слова, сопоставляющиеся со *словом*. Эти слова становятся новым списком “похожих слов”; вы можете выбрать для замены одно из них, набрав цифру. Вы можете использовать в *слове* знак ‘`*`’ для описания шаблона.

`C-g` Выйти из интерактивной проверки правописания. Вы можете перезапустить ее позже с помощью `C-u M-$`.

`X` То же, что и `C-g`.

`x` Выйти из интерактивной проверки правописания и переместить точку назад, где она была, когда вы запустили проверку.

`q` Выйти из интерактивной проверки правописания и уничтожить процесс Ispell.

`C-l` Перерисовать экран.

`C-z` Этот ключ имеет свое обычное значение (приостановить Emacs или минимизировать этот фрейм).

Команда `ispell-complete-word`, которая привязана к ключу `M-`⟨TAB⟩ в режиме Text и родственных с ним режимах, показывает список завершений, основываясь на исправлении орфографии. Вставьте начало слова, а затем напечатайте `M-`⟨TAB⟩; эта команда отобразит окно со списком завершений. Чтобы выбрать одно из перечисленных завершений, щелкните на нем `Mouse-2` или переместите к нему курсор и нажмите ⟨RET⟩. См. Раздел 21.7 [Режим Text], с. 190.

Однажды будучи запущенным, подпроцесс Ispell продолжает работать (ожидать какой-либо работы), чтобы последующие команды проверки правописания завершались быстрее. Если вы хотите избавиться от процесса Ispell, воспользуйтесь `M-x ispell-kill-ispell`. Обычно это не так необходимо, поскольку этот процесс не занимает время, когда вы не проводите проверку правописания.

Ispell использует два словаря: стандартный и ваш личный. Переменная `ispell-dictionary` задает имя используемого файла стандартного словаря. Значение `nil` говорит, что нужно использовать словарь по умолчанию. Команда `M-x ispell-change-dictionary` устанавливает эту переменную и затем перезапускает подпроцесс Ispell, чтобы он использовал другой словарь.

# 14 Работа с файлами

Операционная система хранит постоянные даные в именованных *файлах*. Поэтому большая часть текста, который вы редактируете в Emacs, приходит из файлов и в конечном итоге записывается в файл.

Чтобы редактировать файл, вы должны велеть Emacs считать его и подготовить буфер, содержащий копию текста файла. Это называется *обращением* к файлу. Команды редактирования применяются непосредственно к тексту в буфере, то есть к копии внутри Emacs. Ваши изменения появляются в самом файле, только когда вы *сохраните* буфер в файле.

Кроме обращения к файлам и их сохранения Emacs может удалять, копировать, переименовывать и добавлять в файлы и работать с каталогами файлов.

## 14.1 Имена файлов

Большинство команд Emacs, которые оперируют с файлами, требуют от вас указания имени файла. (Запись и восстановление являются исключением; буферу известно, какое имя файла используется для них.) Имена файлов задаются с использованием минибуфера (см. Глава 5 [Минибуфер], с. 45). Вы можете использовать *завершение* для облегчения написания длинных имен файлов. См. Раздел 5.3 [Завершение], с. 47.

Для большинства операций существует *имя файла по умолчанию*, которое будет использовано, если вы наберете просто ⟨RET⟩, вводя пустой аргумент. Обычно имя файла по умолчанию — это имя файла, находящегося в текущем буфере, что упрощает действия над этим файлом с помощью любых файловых команд Emacs.

Каждый буфер имеет свой каталог по умолчанию, обычно тот же самый, что и каталог файла, к которому обращается этот буфере. Когда вы вводите имя файла, не указывая каталог, он использует каталог по умолчанию. Если вы зададите каталог в относительной форме, с помощью имени, которое не начинается с косой черты, оно интерпретируется по отношению к каталогу по умолчанию. Каталог по умолчанию хранится в переменной `default-directory`, которая имеет свое собственное значение в каждом буфере.

Например, если по умолчанию имя файла — ‘`/u/rms/gnu/gnu.tasks`’, то каталогом по умолчанию будет ‘`u/rms/gnu/`’. Если вы наберете просто ‘`foo`’, не описывая каталог, то это будет сокращением для ‘`/u/rms/gnu/foo`’. ‘`../.login`’ будет соответствовать ‘`/u/rms/.login`’. ‘`new/foo`’ обозначает файл с именем ‘`/u/rms/gnu/new/foo`’.

Команда `M-x pwd` печатает каталог по умолчанию для текущего буфера, а команда `M-x cd` устанавливает его (значение считывается в минибуфере). Каталог по умолчанию в буфере изменяется только тогда, когда используется команда `cd`. Каталог по умолчанию для буфера, обращающегося к файлу, инициализируется по каталогу файла, к которому он обратился. Если буфер создается произвольным образом с помощью `C-x b`, его каталог по умолчанию копируется из того буфера, который был текущим в тот момент.

Каталог по умолчанию фактически появляется в минибуфере, когда минибуфер становится активным для чтения имени файла. Это служит двум целям: чтобы *показать* вам, что имеется по умолчанию, так что вы можете набрать соответствующее имя файла и с определенностью узнать, что это будет значить, и чтобы позволить вам *отредактировать* каталог по умолчанию и задать другой каталог. Эта подстановка каталога по умолчанию не делается, если переменная `insert-default-directory` установлена равной `nil`.

Заметим, что законно набирать полное имя файла после того, как вы войдете в минибуфер, игнорируя присутствие имени каталога по умолчанию как части текста. Окончательное содержание минибуфера может казаться неправильным, но на самом деле это не так. Например, если сначала минибуфер содержал ‘`/usr/tmp/`’, и вы добавили ‘`/x1/rms/foo`’,

вы получите ‘/usr/tmp//x1/rms/foo’; но Emacs игнорирует все до того места, где встречены две косые черты подряд; в результате получается ‘/x1/rms/foo’. См. Раздел 5.1 [Минибуфер Файл], с. 45.

Литера ‘$’ в имени файла используется для подстановки переменных среды. Например, если вы применили команду оболочки ‘export FOO=rms/hacks’ для установки переменной среды с именем FOO, то вы можете использовать ‘/u/$FOO/test.c’ или ‘/u/${FOO}/test.c’ в качестве сокращения для ‘/u/rms/hacks/test.c’. Имя переменной среды состоит из всех букв и цифр после ‘$’; или оно может быть заключено в фигурные скобки после ‘$’. Заметьте, что команды оболочки, которые устанавливают переменные среды, повлияют на Emacs, только если они были выполнены до запуска Emacs.

Чтобы получить доступ к файлу, содержащему в имени знак ‘$’, напечатайте ‘$$’. Эта пара превращается в один ‘$’, в то время как для одиночного ‘$’ делается подстановка переменной. Или вы можете отменить особый смысл всех знаков в имени файла с помощью последовательности ‘/:’ (см. Раздел 14.13 [Буквальные имена файлов], с. 134).

Лисповская функция, которая производит подстановку, называется substitute-in-file-name. Подстановка выполняется только для имен файлов, считываемых как таковые с использованием минибуфера.

Вы можете включать в имена файлов знаки, не входящие в ASCII, если установите переменную file-name-coding-system в отличное от nil значение. См. Раздел 18.9 [Задание кодирования], с. 168.

## 14.2 Обращение к файлам

C-x C-f Обратиться к файлу (find-file).

C-x C-r Обратиться к файлу для просмотра, не допуская его изменения (find-file-read-only).

C-x C-v Обратиться к другому файлу, вместо последнего посещенного (find-alternate-file).

C-x 4 f Обратиться к файлу в другом окне (find-file-other-window). Не меняет текущее окно.

C-x 5 f Обратиться к файлу в другом фрейме (find-file-other-frame). Не изменяет то, что отображается в выбранном фрейме.

M-x find-file-literally
Обратиться к файлу без преобразования его содержимого.

*Обращение* к файлу означает копирование его содержимого в буфер Emacs, где вы можете его редактировать. Emacs создает новый буфер для каждого файла, к которому вы обращаетесь. Мы говорим, что этот буфер обращается к файлу, для хранения которого он был создан. Emacs создает имя буфера из имени файла, отбрасывая каталог и сохраняя просто собственно имя. Например, файл с именем ‘/usr/rms/emacs.tex’ получит буфер с именем ‘emacs.tex’. Если буфер с таким именем уже существует, то создается уникальное имя путем добавления ‘<2>’, ‘<3>’ и так далее; при этом используется наименьший номер, создающий еще не используемое имя.

Строка режима каждого окна показывает имя буфера, который отражен в этом окне, таким образом вы всегда можете сказать, какой буфер редактируете.

Все изменения, сделанные при помощи команд редактирования, делаются в буфере Emacs. Они не влияют на файл, к которому вы обратились, или на любое другое постоянное место, пока вы не *сохранили* буфер. Сохранение буфера означает, что Emacs записывает текущее содержимое этого буфера в файл, к которому он обращался. См. Раздел 14.3 [Сохранение], с. 108.

Если буфер содержит изменения, которые не были сохранены, то о буфере говорится, что он *изменен* (или *модифицирован*). Это важно, так как это подразумевает, что какие-то изменения потеряются, если буфер не будет сохранен. Если буфер изменен, около левого края строка режима показываются две звездочки.

Чтобы обратиться к файлу, используйте команду `C-x C-f` (`find-file`). После нее введите имя файла, который вы хотите посетить, ограниченное ⟨RET⟩.

Имя файла считывается с использованием минибуфера (см. Глава 5 [Минибуфер], с. 45) с использованием значений по умолчанию и завершением стандартным способом (см. Раздел 14.1 [Имена файлов], с. 105). Пока вы находитесь в минибуфере, вы можете прервать выполнение `C-x C-f`, набрав `C-g`.

Подтверждением того, что `C-x C-f` завершилась удачно, служит появление нового текста на экране и нового имени буфера в строке режима. Если указанный файл не существует и не может быть создан или считан, тогда возникает ошибка. Сообщение о ней печатается в эхо-области.

Если вы обращаетесь к файлу, который уже существует в Emacs, `C-x C-f` не создает другой копии. Она выбирает существующий буфер, содержащий этот файл. Однако, перед этим она проверяет, не изменился ли сам файл с тех пор, как вы обратились к нему или записали его в прошлый раз. Если файл был изменен, то будет напечатано предостерегающее сообщение. См. Раздел 14.3.2 [Защита от одновременного редактирования], с. 112.

А что если вы захотите создать новый файл? Просто обратитесь к нему. Emacs печатает в эхо-области ‘`(New File)`’, но в других отношениях ведет себя так же, как, если бы вы обратились бы к существующему пустому файлу. Если вы сделаете любые изменения и запишете их, то файл будет создан.

Emacs узнаёт из содержимого файла, какое соглашение используется в нем для разделения строк — перевод строки (используемый в GNU/Linux и Unix), возврат каретки и перевод строки (используемые в системах Microsoft) или просто возврат каретки (используемый на Macintosh) — и автоматически преобразует содержимое к обычному формату Emacs, в котором строки разделяются знаками перевода строки. Это часть общего средства преобразования системы кодирования (см. Раздел 18.7 [Системы кодирования], с. 165), и это позволяет редактировать файлы, перенесенные с различных операционных систем с одинаковым удобством. Если вы изменили текст и сохранили файл, Emacs производит обратное преобразование, заменяя переводы строки на возврат каретки и перевод строки или просто возврат каретки, если это необходимо.

Если указанный вами файл фактически является каталогом, то `C-x C-f` вызывает Dired, броузер каталогов Emacs, чтобы вы могли “редактировать” содержимое этого каталога (см. Глава 28 [Dired], с. 291). Dired предоставляет удобный способ удаления, просмотра или обработки файлов в каталоге. Однако, если переменная `find-file-run-dired` равна `nil`, попытка обратиться к каталогу будет считаться ошибкой.

Если заданное вами имя файла содержит символы подстановки, Emacs обращается ко всем файлам, соответствующим этому имени. См. Раздел 14.13 [Буквальные имена файлов], с. 134, если вы хотите обратиться к файлу, чье имя в самом деле содержит символы подстановки.

Если операционная система не позволяет вам изменять файл, к которому вы обращаетесь, Emacs делает буфер доступным только для чтения, так что у вас не получится внести изменения, которые будет проблематично впоследствии сохранить. Вы можете сделать буфер доступным для записи с помощью `C-x C-q` (`vc-toggle-read-only`). См. Раздел 15.3 [Другие операции с буферами], с. 136.

Иногда вы можете захотеть обратиться к файлу в режиме только чтения, чтобы защититься от случайного внесения изменений; делайте это, обращаясь к файлу с помощью команды `C-x C-r` (`find-file-read-only`).

Если вы обратились к несуществующему файлу неумышленно (так как вы набрали неправильное имя файла), используйте команду `C-x C-v` (`find-alternate-file`), чтобы обратиться к файлу, который вам нужен на самом деле. `C-x C-v` подобна `C-x C-f`, но уничтожает текущий буфер (после того, как сначала предложит записать его, если он изменен). При считывании нового имени файла она вставляет полное имя текущего файла, оставляя точку сразу после имени каталога; это удобно, если вы сделали небольшую ошибку, когда вводили имя файла.

Если вы обращаетесь к файлу, который существует, но не может быть прочитан, `C-x C-f` выдает ошибку.

`C-x 4 f` (`find-file-other-window`) похожа на `C-x C-f`, но буфер, содержащий описанный файл, выбирается в другом окне. Окно, которое было выбранно до вызова `C-x 4 f`, продолжает показывать тот же буфер, который уже показывался. Если эта команда используется, когда существует только одно окно, то это окно делится на два, одно из которых продолжает показывать то же, что и прежде, а другое показывает новый затребованный файл. См. Глава 16 [Окна], с. 141.

`C-x 5 f` (`find-file-other-frame`) аналогична этой команде, но открывает новый фрейм или делает видимым любой существующий фрейм, показывающий искомый файл. Это возможно, только если вы пользуетесь оконной системой. См. Глава 17 [Фреймы], с. 147.

Если вы хотите отредактировать файл как последовательность знаков без особой перекодировки или преобразования, используйте команду `M-x find-file-literally`. Она обращается к файлу, как `C-x C-f`, но не делает преобразования формата (см. Раздел 21.11 [Форматированный текст], с. 198), преобразования знаковых кодов (см. Раздел 18.7 [Системы кодирования], с. 165) или автоматической распаковки (см. Раздел 14.11 [Сжатые файлы], с. 133). Если вы уже обращаетесь к этому же файлу обычным способом (не буквально), эта команда спросит вас, обратиться ли к нему в этот раз буквально.

Две переменные-ловушки позволяют расширениям изменять способ обращения к файлам. Обращение к несуществуещему файлу запускает функции в списке `find-file-not-found-hooks`; значение этой переменной — список функций, и эти функции вызываются одна за другой до тех пор, пока одна из них не вернет отличное от `nil` значение. Любое обращение к файлу, существующему или нет, предполагает, что `find-file-hooks` содержит список функций, и вызывает их все одну за другой. В обоих случаях функции вызываются без аргументов. Первой применяется переменная `find-file-not-found-hooks`. Эти переменные *не* нормальные ловушки, и для обозначения этого их имена заканчиваются на ‘`-hooks`’, а не на ‘`-hook`’. См. Раздел 31.2.3 [Ловушки], с. 349.

Есть несколько способов автоматически указать основной режим для редактирования файла (см. Раздел 19.1 [Выбор режима], с. 175) и установить локальные переменные, определенные для этого файла (см. Раздел 31.2.5 [Переменные файла], с. 351).

## 14.3 Сохранение файлов

*Сохранение* буфера в Emacs означает запись его содержимого обратно в файл, к которому этот буфер обращается.

`C-x C-s` Сохранить текущий буфер в файле, к которому этот буфер обращается (`save-buffer`).

`C-x s` Сохранить некоторые или все буферы в их соответствующих файлах (`save-some-buffers`).

`M-~` Забыть, что текущий буфер был изменен (`not-modified`).

`C-x C-w` Сохранить текущий буфер в заданный файл (`write-file`).

`M-x set-visited-file-name`
Изменить имя файла, под которым текущий буфер будет сохраняться.

Если вы хотите сохранить файл и сделать его изменения постоянными, наберите `C-x C-s` (`save-buffer`). После того, как запись закончится, `C-x C-s` печатает сообщение подобное этому:

```
Wrote /u/rms/gnu/gnu.tasks
```

Если же выбранный буфер не модифицирован (в нем не было сделано изменений с тех пор, как он был создан или последний раз записан), его запись не делается, так как это не имеет смысла. Вместо этого, `C-x C-s` печатает такое сообщение в эхо-области:

```
(No changes need to be saved)
```

Команда `C-x s` (`save-some-buffers`) предлагает записать какие-либо или все измененные буферы. Она спрашивает вас, что нужно сделать с каждым буфером. Возможные варианты ответа аналогичны вариантам для `query-replace`:

`y` Сохранить этот буфер и спрашивать об остальных.

`n` Не сохранять этот буфер, но спрашивать об остальных.

`!` Сохранить этот буфер и все остальные без дальнейших вопросов.

⟨RET⟩ Прервать `save-some-buffers` и больше ничего не записывать.

`.` Сохранить этот буфер и выйти из `save-some-buffers`, не спрашивая об остальных буферах.

`C-r` Просмотреть буфер, о котором вас спросили в данный момент. Когда вы выйдите из режима View, вы снова попадаете в `save-some-buffers`, которая продолжает задавать вам вопросы.

`C-h` Показать справочное сообщение о этих вариантах ответа.

`C-x C-c`, последовательность ключей для выхода из Emacs, вызывает `save-some-buffers` и, следовательно, задает эти же вопросы.

Если вы изменили буфер, но не хотите записывать изменения, вы должны предпринять некоторые действия, чтобы предотвратить это. Иначе каждый раз, когда вы используете `C-x s` или `C-x C-c`, вы можете по ошибке его записать. С одной стороны, вы можете набрать `M-~` (`not-modified`), что сбрасывает показатель измененности буфера. Если вы сделаете так, то ни одна из команд записи не будет думать, что буфер нужно сохранять. (‘~’ часто используется как математический символ для обозначения отрицания, таким образом, последовательность `M-~` — это мета-отрицание). Вы могли бы также использовать `set-visited-file-name` (смотрите ниже), чтобы пометить буфер, как обратившийся к файлу с другим именем, который не использовался для чего-то важного. С другой стороны, вы можете отменить все изменения, сделанные с тех пор, когда к файлу обратились, или он был записан, с помощью повторного считывания текста из файла. Это называется *возвращением* к прежнему состоянию. См. Раздел 14.4 [Возвращение], с. 113. Вы могли бы также отменить все изменения, повторяя команду отмены `C-x u` достаточно долго; но возвращение проще.

`M-x set-visited-file-name` заменяет имя файла, к которому обращается текущий буфер. Она считывает новое имя файла, используя минибуфер. Затем соответственно изменяется имя буфера (если новое имя уже не используется). `set-visited-file-name` не записывает буфер в новый вызванный файл, она просто меняет записи внутри Emacs на случай последующего сохранения. Она так же помечает буфер как “измененный”, так что `C-x C-s` *будет* его сохранять.

Если вы хотите пометить буфер как обращающийся к другому файлу и сразу его записать, используйте `C-x C-w` (`write-file`). Это совершенно точный эквивалент `set-visited-file-name`, за которым следует `C-x C-s`. Использование `C-x C-s` в буфере, который не обращался к файлу, имеет то же самое действие, что и `C-x C-w`; то есть, она считывает имя файла, метит буфер как обращающийся к этому файлу и записывает его

туда. По умолчанию имя файла в буфере, который не обращался к файлу, составляется из имени буфера и каталога по умолчанию для этого буфера.

Если новое имя файла подразумевает основной режим, то `C-x C-w` в большинстве случаев переключает в этот режим. Команда `set-visited-file-name` ведет себя так же. См. Раздел 19.1 [Выбор режима], с. 175.

Если Emacs собирается записать файл и видит, что дата последней версии на диске не соответствует тому, что он последний раз читал или записывал, то он ставит вас в известность об этом факте, так как это, возможно, выявляет проблему, вызванную одновременным редактированием, и требует вашего незамедлительного внимания. См. Раздел 14.3.2 [Защита от одновременного редактирования], с. 112.

Если переменная `require-final-newline` не равна `nil`, Emacs ставит ограничитель строки в конец каждого файла, который не заканчивается им, каждый раз, когда файл сохраняется или записывается. По умолчанию эта переменная равна `nil`.

### 14.3.1 Резервные файлы

В большинстве операционных систем переписывание файла автоматически разрушает все сведения о том, что этот файл содержал раньше. Таким образом, запись файла из Emacs отбрасывает старое содержимое файла — или может отбросить, если перед фактической записью Emacs предусмотрительно не скопирует старое содержимое в другой файл, называемый *резервным*.

Для большинства файлов решение о создании резервных копий определяется переменной `make-backup-files`. На большинстве операционных систем ее значение по умолчанию равно `t`, что велит Emacs создавать резервные файлы.

Для файлов, находящихся под контролем системы управления версиями (см. Раздел 14.7 [Управление версиями], с. 116), это определяется переменной `vc-make-backup-files`. По умолчанию она равна `nil`, так как резервные копии излишни, раз вы записываете все предыдущие версии в системе управления версиями. См. Раздел 14.7.9.2 [Обработка рабочих файлов в VC], с. 130.

Существующее по умолчанию значение переменной `backup-enable-predicate` запрещает записывать резервные копии для файлов из каталога ‘`/tmp`’.

По вашему выбору Emacs может сохранять либо только один резервный файл, либо несколько нумерованных резервных файлов для каждого файла, который вы редактировали.

Emacs создает резервный файл только первый раз, когда файл записывается из одного буфера. Вне зависимости от того, сколько раз вы записывали файл, его резервная копия продолжает содержать то, что было в файле перед обращением. Обычно это означает, что резервный файл содержит то, что было в файле перед текущим сеансом редактирования; однако, если вы уничтожите буфер, а затем обратитесь к файлу снова, то при следующем сохранении будет создан новый резервный файл.

Вы также можете явно запросить создание еще одной резервной копии из буфера, даже если буфер был уже сохранен хотя бы раз. Если вы сохраните буфер с помощью `C-u C-x C-s`, записанная таким способом версия станет резервной, если вы сохраните буфер снова. `C-u C-u C-x C-s` сохраняет буфер, но сначала переносит старое содержимое файла в новый резервный файл. `C-u C-u C-u C-x C-s` делает и то, и другое: она создает резервную копию старого содержимого и готовится сделать еще одну из вновь сохраненного содержимого, если вы сохраните буфер опять.

#### 14.3.1.1 Одиночные или нумерованные резервные файлы

Если вы решили держать единственный резервный файл (что принимается по умолчанию), то его имя составляется путем добавления ‘`~`’ к имени редактируемого файла, таким образом, резервный файл для ‘`eval.c`’ назывался бы ‘`eval.c~`’.

Если вы захотите иметь серию пронумерованных резервных файлов, то их имена создаются путем добавления '`.~`', номера и другой '`~`' к исходному имени файла. Таким образом, резервные копии файла '`eval.c`' будут называться '`eval.c.~1~`', '`eval.c.~2~`' и так далее, проходя через такие имена, как '`eval.c.~259~`' и выше.

Если защита запрещает вам записывать резервные файлы под обычными именами, то они записываются как '`%backup%~`' в вашем начальном каталоге. Может существовать только один такой файл, поэтому доступна только резервная копия, сделанная самой последней.

Выбор единственного резервного файла или нескольких управляется переменной `version-control`. Ее возможные значения:

`t` Создавать нумерованные резервные файлы.

`nil` Создавать нумерованные резервные файлы для файлов, которые уже имеют нумерованные файлы. Иначе создавать один резервный файл.

`never` Никогда не создавать нумерованные файлы, всегда делать одиночный резервный файл.

Вы можете установить `version-control` локально в отдельном буфере, для управления созданием резервных копий файла этого буфера. Например, режим Rmail локально устанавливает `version-control` на `never`, чтобы быть уверенным, что для Rmail-файла существует только один резервный файл. См. Раздел 31.2.4 [Локальные переменные], с. 350.

Если вы установите переменную среды `VERSION_CONTROL`, чтобы указать различным утилитам GNU, что делать с резервными файлами, Emacs также подчиняется ей, устанавливая соответственно во время запуска переменную Лиспа `version-control`. Если значение этой переменной среды равно '`t`' или '`numbered`', то `version-control` становится равной `t`; если это значение равно '`nil`' или '`existing`', то `version-control` становится `nil`; если это '`never`' или '`simple`', то `version-control` устанавливается в значение `never`.

#### 14.3.1.2 Автоматическое удаление резервных файлов

Чтобы предотвратить неограниченное потребление пространства на диске, Emacs может удалять пронумерованные резервные версии файлов автоматически. Обычно Emacs хранит только несколько первых и несколько последних резервных файлов, уничтожая все находящиеся между ними. Это происходит каждый раз, когда создается новый резервный файл.

Двумя переменными, контролирующими удаление, являются `kept-old-versions` и `kept-new-versions`. Их значения — это, соответственно, номер самой старой резервной копии файла (наименьший номер), которая должна быть сохранена, и номер самой последней копии (наибольший номер), которая должна сохраняться каждый раз, когда создается новая копия. Помните, что эти значения используются сразу после того, как создастся новая резервная копия; вновь созданная копия включается в счетчик `kept-new-version`. По умолчанию обе переменные равны 2.

Если `delete-old-versions` не равна `nil`, то излишек средних версий уничтожается безропотно. Если же она `nil`, как по умолчанию, тогда вас спрашивают, должен ли быть уничтожен излишек промежуточных версий.

Команда Dired `.` (точка) также может быть использована для удаления старых версий. См. Раздел 28.3 [Удаление в Dired], с. 291.

#### 14.3.1.3 Копирование vs. переименование

Резервные файлы могут быть созданы с помощью копирования старого файла или с помощью его переименования. Эти варианты различаются, когда старый файл имеет

несколько имен. Если старый файл переименовывается в резервный, тогда очередные имена становятся именами для резервного файла. Если вместо этого старый файл копируется, то очередные имена остаются именами для файла, который вы редактируете, и содержание, доступное по этим именам, будет новым содержанием.

Метод создания резервных файлов также может затронуть владельцев и группы владельцев файлов. Если используется копирование, то они не изменяются. Если используется переименование, то вы становитесь владельцем файла, и устанавливается группа по умолчанию (различные операционные системы используют различные значения по умолчанию для группы).

Изменение владельца обычно является хорошей идеей, поскольку тогда всегда видно, кто последним редактировал файл. Кроме того, владельцы резервных копий показывают, кто сделал эти версии. Иногда существует файл, чей владелец не должен изменяться; хорошая идея для таких файлов — включить локальные списки переменных для установки `backup-by-copying-when-mismatch` (см. Раздел 31.2.5 [Переменные файла], с. 351).

Выбор переименования или копирования управляется тремя переменными. По умолчанию делается переименование. Если переменная `backup-by-coping` — не `nil`, то используется копирование. В противном случае, если переменная `backup-by-copying-when-linked` не равна `nil`, то делается копирование для файлов, которые имеют несколько имен, но может все же делаться переименование, когда редактируемый файл имеет только одно имя. Если переменная `backup-by-copying-when-mismatch` — не `nil`, тогда, если переименование привело бы к изменению владельца файла или группы, то делается копирование. `backup-by-copying-when-mismatch` по умолчанию равна `t`, если вы запустили Emacs как привилегированный пользователь.

Когда файл находится под управлением системы контроля версий (см. Раздел 14.7 [Управление версиями], с. 116), Emacs обычно не создает резервных копий как обычно. Но извлечение и фиксирование отчасти подобны созданию резервных копий. Они похожи, к сожалению, и тем, что как правило разрушают жесткие ссылки, разъединяя имя файла, к которому вы обратились, и все другие имена этого же файла. Это не вина Emacs — это делает система управления версиями.

### 14.3.2 Защита от одновременного редактирования

Одновременное редактирование случается, когда два пользователя обращаются к одному и тому же файлу, оба делают изменения и затем оба сохраняют их. Если никого не проинформировали о том, что это случилось, то пользователь, сохранивший свои изменения первым, может позднее обнаружить, что его изменения пропали.

В некоторых системах Emacs сразу замечает, если второй пользователь начинает изменять файл, и выдает немедленное предостережение. На всех системах Emacs делает проверку, когда вы записываете файл, и выдает предупреждение, если вы собираетесь затереть изменения другого пользователя. Вы можете предотвратить потерю чужой работы, предприняв необходимые действия вместо сохранения файла.

Когда вы делаете первую модификацию в буфере Emacs, который обращается к файлу, Emacs записывает, что вы *захватили* этот файл. (Он делает это, создавая символьную ссылку с другим именем в том же каталоге.) Захват отменяется, когда вы запишете изменения. Идея состоит в том, что файл захвачен всегда, когда в буфере, который к нему обращается, есть несохраненные изменения.

Если вы начнете изменять буфер, когда файл, к которому он обращается, захвачен кем-то еще, это приведет к *столкновению*, и Emacs спросит вас, что делать, вызвав лисповскую функцию `ask-user-about-lock`. Вы можете переопределить эту функцию для своих нужд. Стандартное определение этой функции задает вам вопрос и принимает три возможных ответа:

`s` Перехватить захват. Тот пользователь, кто уже редактировал файл, теряет захват, а вы его приобретаете.

`p` Продолжать. Идти дальше и редактировать файл, несмотря на то, что он кем-то захвачен.

`q` Выйти. Это приводит к ошибке (`file-locked`), а изменения, которые вы пытались сделать в буфере, в действительности не будут иметь места.

Заметим, что захват работает на основе имени файла; если файл имел несколько имен, Emacs не осознает, что два имени — это один и тот же файл, и не может предупредить двух пользователей о попытке редактирования одного и того же файла под разными именами. Однако, основание захвата на именах означает, что Emacs может блокировать редактирование новых файлов, которые фактически не существуют, пока их не запишут.

Некоторые системы не сконфигурированы так, чтобы позволить Emacs сделать захваты. В таких случаях Emacs не может определить опасность заранее, но он по-прежнему может обнаружить столкновение, когда вы пытаетесь сохранить файл и затереть чьи-то чужие изменения.

Если в Emacs или в операционной системе случается фатальный сбой, это может оставить файлы захвата, которые уже потеряли актуальность. Поэтому вы можете иногда получить предупреждение о мнимых столкновениях. Когда вы обнаружите, что столконение ложно, просто используйте `p`, чтобы велеть Emacs продолжать.

Каждый раз, когда Emacs записывает буфер, он сначала сверяет дату последней модификации файла, существующего на диске, чтобы увидеть, что она не изменялась с тех самых пор, как к файлу обращались или его записывали последний раз. Если дата не совпадает, то это означает, что изменения были произведены в файле каким-то другим способом, и что эти изменения могут быть потеряны, если Emacs сохранит буфер на самом деле. Чтобы предотвратить это, Emacs печатает предостерегающее сообщение и запрашивает перед записью подтверждение. Иногда вы знаете, почему файл был изменен, и знаете, что это не имеет значения; в этом случае вы можете ответить `yes` и продолжить редактирование. В противном случае, вы должны отменить запись с помощью `C-g` и исследовать ситуацию.

Первое, что вы должны сделать, когда пришло извещение об одновременном редактировании, — распечатать каталог с помощью `C-u C-x C-d` (см. Раздел 14.8 [Каталоги], с. 131). Это покажет вам текущего автора. Вы должны будете попытаться связаться и предупредить его, чтобы он не продолжал редактирование. Чаще всего, следующий шаг — записать содержимое вашего буфера Emacs под другим именем и использовать `diff`, чтобы сравнить два файла.

## 14.4 Возвращение буфера

Если вы сделали обширные изменения в файле, а затем изменили ваше мнение о них, то вы можете от них избавиться, считав предыдущую версию этого файла. Чтобы сделать это, используйте `M-x revert-buffer`, она действует в текущем буфере. Так как от безусловного возврата буфера может пропасть много работы, вы должны подтвердить эту команду вводом `yes`.

`revert-buffer` сохраняет точку на том же самом расстоянии (измеренном в знаках) от начала файла. Если файл был отредактирован только слегка, то после возвращения вы очутитесь примерно в той же части текста, в которой и были. Если вы сделали кардинальные изменения, то то же самое значение точки в старом файле может ссылаться на совершенно другой кусок текста.

Возвращение помечает буфер как “неизмененный” до тех пор, пока не будут сделаны другие изменения.

Некоторые виды буферов, чье содержимое отражает отличные от файлов данные, такие как буферы Dired, также могут быть возвращены. Для них возвращение означает повторное считываение их содержимого из соответствующей базы данных. Буферы, созданные явно с помощью `C-x b`, не могут быть возвращены; `revert-buffer` сообщает об ошибке, когда ее просят сделать это.

Когда вы редактируете файл, который изменяется часто и автоматически — например, протокол вывода от еще работающего процесса — может оказаться удобным, если бы Emacs возвращал файл без подтверждения всякий раз, когда вы снова обращаетесь к этому файлу с помощью `C-x C-f`.

Чтобы запросить такое поведение, установите переменную `revert-without-query` равной списку регулярных выражений. Когда имя файла соответствует одному из этих регулярных выражений, `find-file` и `revert-buffer` будут возвращать его автоматически, если он изменился — при условии, что сам буфер не был модифицирован. (Если вы редактировали текст, сбрасывать ваши изменения будет нехорошо.)

## 14.5 Самосохранение: защита от гибели

Emacs время от времени (основываясь на подсчете нажатых вами клавиш) записывает все посещенные файлы без запроса. Это называется *самосохранением*. Оно убережет вас от потери большого количества работы, если система рухнет.

Когда Emacs определяет, что пришло время для самосохранения, то каждый буфер рассматривается и записывается, если для него включено самосохранение, и он изменялся с тех пор, как последний раз был самосохранен. Во время самосохранения в эхо-области отображается сообщение '`Auto-saving...`', если какой-либо файл действительно сохраняется. Ошибки, появляющиеся во время самосохранения, отлавливаются так, что они не мешают выполнению набранных вами команд.

### 14.5.1 Файлы для самосохранения

Самосохранение обычно не записывает в файлы, к которым вы обратились, так как может быть очень нежелательно записывать программу, которая находится в несогласованном состоянии, когда вы сделали половину планируемых изменений. Вместо этого самосохранение делается в другой файл, который называется *файлом для самосохранения*, а посещенный файл изменяется только тогда, когда вы явно потребуете записать его (например, с помощью `C-x C-s`).

Обычно имя файла для самосохранения создается добавлением знака '`#`' перед и после имени файла, к которому вы обратились. Таким образом, буфер, обращающийся к файлу '`foo.c`', будет самосохранен в файл '`#foo.c#`'. Большинство буферов, которые не обращались к файлам, самосохраняются, только если вы явно потребуете этого; когда они автоматически записываются, имя файла для самосохранения создается добавлением к имени буфера знаков '`#%`' в начале и '`#`' в конце. Например, буфер '`*mail*`', в котором вы составляете отправляемые сообщения, самосохраняется в файл с именем '`#%*mail*#`'. Имена файлов для самосохранения конструируются таким образом, если вы не перепрограммируете часть Emacs, чтобы делалось что-то иное (функции `make-auto-save-file-name` и `auto-save-file-name-p`). Имя файла, которое будет использоваться для самосохранения в буфере, составляется, когда в этом буфере включается самосохранение.

Когда вы удаляете значительную часть текста большого буфера, самосохранение в нем временно выключается. Это делается по той причине, что если вы удалили текст неумышленно, самосохраненный файл может оказаться для вас полезнее, если удаленный текст все еще остается в нем. Чтобы после этого снова разрешить самосохранение, запишите буфер командой `C-x C-s` или используйте `C-u 1 M-x auto-save`.

Если вы хотите, чтобы в файле, к которому вы обратились, выполнялось самосохранение, установите переменную `auto-save-visited-file-name` в отличное от `nil` значение. В этом режиме в действительности нет различий между самосохранением и явным сохранением.

Файл самосохранений удаляется, когда вы записываете содержимое буфера в файл, к которому он обращался. Чтобы воспрепятствовать этому, установите переменную `delete-auto-save-files` равной `nil`. Изменение имени посещенного файла с помощью `C-x C-w` или `set-visited-file-name` переименовывает файл для самосохранения, чтобы он соответствовал имени нового посещенного файла.

### 14.5.2 Управление самосохранением

Каждый раз, когда вы обращаетесь к файлу, для буфера этого файла включается самосохранение, если переменная `auto-save-default` не равна `nil` (но не в пакетном режиме; см. Глава 3 [Вход в Emacs], с. 33). По умолчанию эта переменная равна `t`, поэтому включенное самосохранение — это обычное состояние буферов, работающих с файлами. Самосохранение может включаться или выключаться для любого существующего буфера с помощью команды `M-x auto-save-mode`. Подобно другим командам второстепенных режимов, `M-x auto-save-mode` включает самосохранение при положительном аргументе и выключает, когда аргумент равен нулю или отрицателен; без аргумента она переключает режим.

Emacs производит самосохранение периодически, основываясь на подсчете числа знаков, набранных вами с того момента, как самосохранение было сделано в последний раз. Переменная `auto-save-interval` определяет, сколько знаков приходится между двумя самосохранениями. По умолчанию она равна 300.

Самосохранение также производится, когда вы перестаете печатать на некоторое время. Переменная `auto-save-timeout` говорит, сколько секунд должен ждать Emacs то того, как сделать самосохранение (а также, возможно, и сборку мусора). (Действительный период времени больше, если теущий буфер велик; это эвристика, цель которой — не мешать вам, когда вы редактируете длинные буферы, самосохранение в которых занимает заметное время.) Самосохранение во время периодов бездействия выполняет две задачи: во-первых, оно гарантирует, что ваша работа сохраняется, если вы отошли на некоторое время от терминала; во-вторых, оно может позволить избежать самосохранения в то время, когда вы на самом деле печатаете.

Emacs также выполняет самосохранение, когда получает фатальную ошибку. Это включает уничтожение задания Emacs с помощью команды оболочки, как ‘`kill %emacs`’, или в результате разъединения телефонной линии или связи по сети.

Вы можете явно запросить самосохранение с помощью команды `M-x do-auto-save`.

### 14.5.3 Восстановление данных из самосохранения

Вы можете использовать содержимое файла самосохранения для восстановления потерянных данных, запустив команду `M-x recover-file` ⟨RET⟩ *файл* ⟨RET⟩. Эта команда обращается к *файлу* и затем (после вашего подтверждения) переписывает содержание из его самосохраненного файла ‘`#`*файл*`#`’. Вы можете потом сохранить этот буфер при помощи `C-x C-s`, чтобы поместить восстановленный текст в сам *файл*. Например, чтобы восстановить файл ‘`foo.c`’ из его файла для самосохранения ‘`#foo.c#`’, сделайте следующее:

```
M-x recover-file ⟨RET⟩ foo.c ⟨RET⟩
yes ⟨RET⟩
C-x C-s
```

Перед тем как запросить у вас подтверждение, `M-x recover-file` показывает распечатку каталога, описывающую заданный файл и файл самосохранения, так что вы можете

сравнить их размеры и даты. Если файл для самосохранения старше, то `M-x recover-file` не предлагает его считывать.

Если Emacs или комьютер потерпели крах, вы можете восстановить все файлы, которые вы редактировали, из их самосохраненных файлов при помощи команды `M-x recover-session`. Она сначала показывает перечень записанных прерванных сеансов. Переместите точку к нужной вам и наберите `C-c C-c`.

Затем `recover-session` спрашивает о каждом файле, который редактировался во время этого сеанса, нужно ли его восстанавливать. Если вы отвечаете `y`, она вызывает `recover-file`. Эта команда работает обычным способом: показывает даты оригинального файла и его самосохраненной версии и спрашивает еще раз, нужно ли его восстанавливать.

Когда `recover-session` завершается, все файлы, которые вы решили восстановить, присутствуют в буферах Emacs. Теперь вам нужно их сохранить. Только это — их сохранение — обновляет сами файлы.

Прерванные сеансы записываются для последующего восстановления в файлах с именами '`~/.saves-`*pid*`-`*машина*'. Часть '`~/.saves`' этих имен получается из знаечения `auto-save-list-file-prefix`. Вы можете сделать так, чтобы записи о сеансах держались в другом месте, устанавливая эту переменную в вашем файле '`.emacs`', но вам также придется переопределить `recover-session`, чтобы она искала в новом месте. Если в файле '`.emacs`' вы установите `auto-save-list-file-prefix` равной `nil`, сеансы не будут записываться для восстановления.

## 14.6 Псевдонимы файлов

Символьные ссылки и жесткие ссылки позволяют одному и тому же файлу иметь несколько имен. Жесткие ссылки — это альтернативные имена, ссылающиеся непосредственно на файл; все имена одинаково правильны, и ни одно из них не является предпочтительным. Напротив, символьные ссылки — это вид определенных псевдонимов: когда файл '`foo`' является символьной ссылкой на '`bar`', вы можете использовать оба имени, но действительным именем будет '`bar`', тогда как '`foo`' — это просто псевдоним. Более сложные ситуации возникают, когда символьные ссылки указывают на каталоги.

Если вы обращаетесь к одному и тому же файлу по двум именам, Emacs обычно создает два разных буфера, но предупреждает вас об этой ситуации.

Если вы хотите избежать обращения к одному и тому же файлу в двух буферах под разными именами, установите переменную `find-file-existing-other-name` в отличное от `nil` значение. Тогда `find-file` использует существующий буфер, обращающийся к этому файлу, независимо от того, какое имя вы зададите.

Если переменная `find-file-visit-truename` не равна `nil`, то для буферов записывается *истинные имена* файлов (получаемые заменой всех символьных ссылок на их целевые имена), а не имена, заданные вами. Установка `find-file-visit-truename` также подразумевает действие `find-file-existing-other-name`.

## 14.7 Управление версиями

*Системы управления версиями* — это пакеты, которые могут записывать несколько версий исходного файла, обычно сохраняя неизменившиеся части этого файла только один раз. Системы управления версиями также записывают сведения об истории, такие как время создания каждой версии, имя ее создателя и описание изменений в этой версии.

Интерфейс Emacs для управления версиями называется VC. Его команды работают с тремя системами управления версиями — RCS, CVS и SCCS. Проект GNU рекомендует RCS и CVS, которые являются свободными программами, и их можно получить от Фонда Свободного Программного Обеспечения.

### 14.7.1 Введение в управление версиями

VC позволяет вам использовать системы управления версиями из Emacs, хорошо интегрируя операции по управлению версиями и редактирование. VC предоставляет обобщенный интерфейс к управлению версиями, так что вы можете использовать его одним методом независимо от того, какую систему вы применяете.

Этот раздел предоставляет общий обзор управления версиями и описывает системы управления версиями, которые поддерживает VC. Вы можете пропустить этот раздел, если знакомы с системой управления версиями, которую хотите использовать.

#### 14.7.1.1 Поддерживаемые системы управления версиями

На данный момент VC работает с тремя разными системами управления версиями или "постпроцессорами": RCS, CVS и SCCS.

RCS — это свободная система управления версиями, ее можно получить от Фонда Свободного Программного Обеспечения. Вероятно, это наиболее развитый из поддерживаемых постпроцессоров, и команды VC концептуально ближе всего к RCS. Почти все, что вы можете делать с RCS, можно сделать через VC.

CVS построена поверх RCS и расширяет возможности RCS, позволяя более сложное управление выпусками и разработку многими пользователями. VC поддерживает основные операции редактирования под CVS, но для некоторых менее частых задач вам все же понадобится вызывать CVS из командной строки. Заметьте, что до использования CVS вы должны настроить репозиторий, но это слишком сложная тема, чтобы ее здесь рассматривать.

SCCS — это несвободная, но широко используемая система управления версиями. По возможностям это самая слабая из трех систем, поддерживаемых VC. VC компенсирует отсутствие некоторых средств в SCCS (снимков, например), реализуя их сама, но некоторые другие возможности VC, такие как множественные ветви, недоступны при использовании SCCS. Вам стоит применять SCCS, только если по какой-то причине вы не можете воспользоваться RCS.

#### 14.7.1.2 Концепции управления версиями

Когда файл помещен под контроль системы управления версиями, мы говорим, что он *зарегистрирован* в этой системе. Для каждого зарегистрированного файла есть соответствующий *мастер-файл*, который представляет текущее состояние файла и историю его изменений — достаточную для реконструкции текущей или любой более ранней версии. Обычно в мастер-файле также сохранены *журнальные записи* для каждой версии, описывающие словами, что было изменено в этой версии.

Файл, сопровождаемый управлением версий, иногда называется *рабочим файлом*, соответствующим его мастер-файлу. Вы редактируете рабочий файл и делаете в нем изменения, как вы делали бы йдля обычного файла. (В SCCS и RCS вы должны *блокировать* файл перед тем, как начать его редактировать.) После того, как вы сделали некоторые изменения, вы *фиксируете* этот файл, что записывает эти изменения в мастер-файле вместе с журнальной записью для них.

В CVS обычно бывает много рабочих файлов, соответствующих одному мастер-файлу — часто у каждого пользователя есть своя копия. Таким способом можно использовать и RCS, но это не обычный метод ее применения.

В системе управления версиями как правило есть некий механизм для координирования пользователей, которые хотят редактировать один и тот же файл. Один из способов — *блокирование* (аналогичное блокированию, которое Emacs применяет для отслеживания

попыток одновременного редактирования файла, но отличающееся от него). Другой метод — объединение ваших изменений с изменениями другого человека при их фиксировании.

При управлении версиями с блокированием, рабочие файлы обычно доступны только для чтения, так что вы не можете их изменить. Вы просите систему управления версиями сделать файл записываемым, блокируя его; в одно время это может сделать только один пользователь. Когда вы фиксируете ваши изменения, это разблокирует файл, и он снова становится доступным только для чтения. Это позволяет другим пользователям заблокировать этот файл и делать дальнейшие изменения. SCCS всегда использует блокирование, и RCS обычно тоже.

Для RCS есть другая альтернатива — позволить каждому пользователю изменять рабочий файл в любое время. В таком режиме в блокировании нет нужды, но оно позволяется; запись новой версии по-прежнему производится путем фиксирования.

CVS обычно позволяет каждому пользователю изменять свою собственную копию рабочего файла в любое время, но требует объединения с версиями других пользователей во время фиксирования. Однако, CVS тоже можно настроить так, чтобы она требовала блокирования. (см. Раздел 14.7.9.1 [Параметры постпроцессора], с. 130).

### 14.7.2 Управление версиями и строка режима

Когда вы обращаетесь к файлу, который находится под контролем системы управления версиями, Emacs показывает это в строке режима. Например, ‘`RCS-1.3`’ говорит, что для этого файла используется RCS, а текущая версия — 1.3.

Знак между именем постпроцессора и номером версии показывает статус этого файла в системе управления версиями. Дефис ‘`-`’ говорит, что рабочий файл не заблокирован (если блокирование используется) или не изменен (если блокирование не используется). Знак ‘`:`’ показывает, что файл блокирован или изменен. Если файл блокировал какой-то другой пользователь (скажем, ‘`jim`’), это отображается как ‘`RCS:jim:1.3`’.

### 14.7.3 Основы редактирования с управлением версиями

Основная команда VC — это команда общего назначения, которая либо блокирует, либо фиксирует файл в зависимости от ситуации.

`C-x C-q`
`C-x v v` Делает следующую логическую операцию управления версиями для этого файла.

Строго говоря, команда для этого называется `vc-next-action`, она привязана к `C-x v v`. Однако, обычное значение `C-x C-q` — делать буферы, предназначенные только для чтения, доступными для записи или наоборот; мы расширили ее так, чтобы она правильно делала это же для файлов, находящихся под контролем системы управления версиями, производя подходящие операции. Когда вы набираете `C-x C-q` в зарегистрированнм файле, она ведет себя, как `C-x v v`.

Точное действие этой команды зависит от состояния файла и от того, использует ли система управления версиями блокирование или нет. SCCS и RCS обычно используют блокирование; CVS обычно не использует.

#### 14.7.3.1 Основы управления версиями с блокированием

Если для файла применяется блокирование (как в в случае с SCCS и RCS в режиме по умолчанию), `C-x C-q` может либо блокировать файл, либо зафиксировать его:

- Если файл не заблокирован, `C-x C-q` блокирует его и делает доступным для записи, чтобы вы могли его изменять.

- Если файл заблокирован вами и содержит изменения, `C-x C-q` фиксирует эти изменения. Для этого она сначала считывает журнальную запись новой версии. См. Раздел 14.7.3.3 [Буфер журнала], с. 119.
- Если файл заблокирован вами, но не содержит изменений с тех пор, как вы его блокировали, `C-x C-q` снимает блокировку и делает файл опять доступным только для чтения.
- Если файл заблокирован кем-то еще, `C-x C-q` спрашивает вас, хотите ли вы "украсть блокировку" у этого пользователя. Если вы ответите да, то файл становится блокированным вами, но человеку, который раньше заблокировал его, посылается сообщение, чтобы проинформировать о случившемся.

Эти правила применимы также, когда вы используете CVS в блокирующем режиме, за исключением того, что там нет такого понятия, как перехват блокировки.

### 14.7.3.2 Основы управления версиями без блокирования

Когда блокирования нет — по умолчанию в CVS — рабочие файлы всегда доступны для записи; вам не нужно ничего делать перед тем, как начать редактирование. Индикатором статуса в строке режима служит '`-`', если файл не изменен; он заметяется на '`:`', как только вы сохраняете любые изменения в рабочем файле.

Вот что делает `C-x C-q` при использовании CVS:

- Если какой-то другой пользователь зафиксировал свои изменения в мастер-файле, Emacs спрашивает вас, хотите ли вы влить эти изменения в ваш рабочий файл (см. Раздел 14.7.6.3 [Объединение], с. 124). Вы обязаны сделать это до того, как сможете зафиксировать свои собственные изменения.
- Если в мастер-файле нет изменений, но вы модифицировали ваш рабочий файл, `C-x C-q` фиксирует ваши изменения. Для этого она сначала считывает журнальную запись для новой версии. См. Раздел 14.7.3.3 [Буфер журнала], с. 119.
- Если файл не изменен, `C-x C-q` ничего не делает.

Эти правила применимы также и в том случае, если вы используете RCS в режиме, когда она не требует блокирования, за исключением того, что автоматическое объединение с мастер-файлом не реализовано. К сожалению, это означает, что вас не информируют, если еще один пользователь зафиксировал изменения в том же файле после того, как вы начали редактирование, и когда это происходит, его изменения в результате исчезнут в вашей зафиксированной версии (хотя они останутся в мастер-файле, так что не окажутся полностью потерянными). Поэтому вы должны убедиться, что текущая версия осталась неизмененной перед тем как фиксировать свои изменения. Мы надеемся устранить этот риск и предоставить автоматическое объединение для RCS в будущей версии Emacs.

Кроме того, даже в этом режиме RCS блокирование возможно, хоть и необязательно; `C-x C-q` в неизмененном файле блокирует этот файл, так же, как с RCS в обычном (блокирующем) режиме.

### 14.7.3.3 Буфер журнальной записи

Когда вы фиксируете изменения, `C-x C-q` сначала считывает журнальную запись. Она поднимает буфер с именем '`*VC-Log*`', в котором вы вводите журнальную запись. Когда вы завершили, нажмите `C-c C-c` в буфере '`*VC-Log*`'. Только тогда происходит действительное фиксирование.

Чтобы прервать фиксирование, просто **не** набирайте `C-c C-c` в этом буфере. Вы можете переключать буферы и делать другое редактирование. Пока вы не пытаетесь зафиксировать другой файл, запись, которую вы редактировали, остается в буфере '`*VC-Log*`', и вы можете в любое время вернуться в этот буфер и завершить фиксирование.

Если вы модифицируете несколько исходных файлов с одной целью, часто бывает удобно указать одинаковую журнальную запись для многих файлов. Чтобы сделать так, используйте историю предыдущих журнальных записей. Предназначенные для этого команды `M-n`, `M-p`, `M-s` и `M-r` работают так же, как команды истории минибуфера (за исключением того, что они применяются вне минибуфера).

Каждый раз, когда вы фиксируете файл, буфер журнальной записи помещается в режим VC Log, что влечет запуск двух ловушек: `text-mode-hook` и `vc-log-mode-hook`. См. Раздел 31.2.3 [Ловушки], с. 349.

### 14.7.4 Просмотр и сравнение старых версий

Одна из удобных возможностей систем управления версиями — возможность просмотра любой версии файла или сравнения двух версий.

`C-x v ~` *версия* ⟨RET⟩
: Показывает заданную *версию* файла, к которому вы обратились, в отдельном буфере.

`C-x v =`
: Сравнивает текущее содержимое буфера с последней зафиксированной версией этого файла.

`C-u C-x v =` *файл* ⟨RET⟩ *старая-вер* ⟨RET⟩ *новая-вер* ⟨RET⟩
: Сравнивает две заданные версии *файла*.

`C-x v g`
: Выводит результат команды CVS annotate с использованием разных цветов.

Чтобы просмотреть старую версию in toto, обратитесь к файлу и наберите `C-x v ~` *версия* ⟨RET⟩ (`vc-version-other-window`). Это поместит текст указанной *версии* в файл с именем ‘*имя-файла*`.~`*версия*`~`’ и обратится к нему в новом буфере в отдельном окне. (В RCS, вы также можете выбрать старый буфер и создать из него новую ветвь. См. Раздел 14.7.6 [Ветви], с. 123.)

Но обычно более удобно сравнивать две версии файла с помощью команды `C-x v =` (`vc-diff`). Просто `C-x v =` сравнивает текущее содержимое буфера (сохраняя его в файл, если требуется) с последней зафиксированной версией этого файла. `C-u C-x v =` с числовым аргументом считывает имя файла и номера двух версий, а затем сравнивает эти версии указанного файла.

Если вы зададите вместо зарегистрированного файла имя каталога, эта команда сравнивает заданные версии всех зарегистрированных файлов в этом каталоге и его подкаталогах.

Вы можете задать зафиксированную версию числом; пустой ввод обозначает текущее содержимое рабочего файла (оно может отличаться от всех зафиксированных версий). Вы также можете задать вместо одного или обоих номеров версий имя снимка (см. Раздел 14.7.7 [Снимки], с. 126).

Эта команда работает путем запуска утилиты `diff`, передавая ей аргументы из переменной `diff-switches`. Она показывает вывод в особом буфере в другом окне. В отличие от команды `M-x diff`, `C-x v =` не пытается определить позиции различий в старой и новой версиях. Так делается, потому что как правило одна или обе версии не существуют в момент сравнения в виде файлов; они существуют только в записях в мастер-файле. См. Раздел 14.9 [Сравнение файлов], с. 132, для получения большей информации о `M-x diff`.

Для файлов, управляемых CVS, вы можете посмотреть результат команды CVS annotate с использованием разных цветов для улучшения внешнего вида. Используйте для этого команду `M-x vc-annotate`. Красным обозначается новое, синим — старое, а промежуточные цвета обозначают промежуточные версии. Префиксный аргумент *n* задает коэффициент растяжения по временной шкале; он говорит, что каждый цвет покрывает в *n* раз больший период времени.

### 14.7.5 Второстепенные команды VC

Этот раздел объясняет второстепенные команды VC; те, что вы могли бы использовать раз в день.

#### 14.7.5.1 Регистрирование файла для управления версиями

Вы можете поместить любой файл под контроль системы управления версиями, просто обратясь к нему и набрав затем `C-x v i` (`vc-register`).

`C-x v i` Регистрирует файл в системе управления версиями.

Чтобы зарегистрировать файл, Emacs должен выбрать, какую систему управления версиями для него использовать. Вы можете указать свой выбор явно, устанавливая `vc-default-back-end` в значения `RCS`, `CVS` или `SCCS`. Иначе, если есть подкаталог с именем '`RCS`', '`SCCS`' или '`CVS`', Emacs использует соответствующую систему управления версиями. В отсутствие каких-либо указаний, по умолчанию выбирается RCS, если в системе установлена RCS, иначе SCCS.

Если применяется блокирование, `C-x v i` оставляет файл неблокированным и запрещает в запись. Наберите `C-x C-q`, если вы хотите начать в нем редактирование. После регистрирования с использованием CVS вы должны зафиксировать изменения, набрав `C-x C-q`.

Первоначальной версии вновь зарегистрированного файла присваивается номер 1.1, по умолчанию. Вы можете задать другое значение по умолчанию, устанавливая переменную `vc-default-init-version`, или вы можете дать `C-x v i` числовой аргумент; тогда она считывает номер версии для этого конкретного файла в минибуфере.

Если `vc-initial-comment` отлична от `nil`, `C-x v i` считывает начальный комментарий, описывающий предназначение данного исходного файла. Начальный комментарий считывается так же, как журнальные записи (см. Раздел 14.7.3.3 [Буфер журнала], с. 119).

#### 14.7.5.2 Команды VC для выяснения статуса файла

`C-x v l` Показывает состояние файла в системе управления версиями и историю изменений.

Чтобы просмотреть подробную информацию о статусе версии и историю файла, наберите `C-x v l` (`vc-print-log`). Это покажет историю изменений текущего файла, в том числе текст журнальных записей. Вывод появляется в другом окне.

#### 14.7.5.3 Отмена действий над версиями

`C-x v u` Возвращает буфер и его файл к последней зафиксированной версии.

`C-x v c` Удаляет последнее внесенное изменение из мастер-файла для файла, к которому вы обратились. Это отменяет ваше последнее фиксирование.

Если вы хотите сбросить ваши текущие изменения и вернуть файл к его последней версии, используйте `C-x v u` (`vc-revert-buffer`). Файл остается неблокированным; если используется блокирование, до того, как сможете изменять этот файл, вы должны сначала снова его заблокировать. `C-x v u` требует подтверждения, если только она не видит, что вы не вносили изменения со времени последего фиксирования.

`C-x v u` также нужна для разблокирования файла, когда вы блокировали его, а потом решили не изменять.

Чтобы отменить уже зафиксированное изменение, используйте `C-x v c` (`vc-cancel-version`). Эта команда сбрасывает все записи о последней фиксированной версии. `C-x v c` также предлагает вернуть ваш рабочий файл и буфер к предыдущей версии (к той, что предшествовала удаляемой).

Если вы отвечаете `no`, VC сохраняет ваши изменения в буфере и блокирует файл. Такая возможность полезна, когда вы зафиксировали изменение, а затем обнаружили в нем тривиальную ошибку; вы можете отменить фиксирование, исправить ошибку и снова зафиксировать файл.

Когда `C-x v c` не возвращает буфер, она вместо этого сворачивает все заголовки версий (см. Раздел 14.7.8.3 [Заголовки версии], с. 128). Это делается по той причине, что буфер больше не соответствует никакой существующей версии. Если вы снова зафиксируете его, заголовки будут раскрыты правильно относительно нового номера версии.

Однако, автоматически свернуть заголовок RCS '`$Log$`' невозможно. Если вы пишете этот заголовок, вы должны свернуть его вручную — удалив вхождение для версии, которую вы отменили.

Будьте осторожны при вызове `C-x v c`, так как при этом легко потерять много работы. Чтобы помочь вам не допустить ошибки, эта команда всегда требует подтверждения с `yes`. Обратите внимание также на то, что эта команда выключена под CVS, поскольку там отмена версий очень опасна и не рекомендуется.

### 14.7.5.4 Dired под VC

Когда вы работаете над большой программой, часто бывает полезно узнать, какие файлы были изменены в пределах целого дерева каталогов, или просмотреть статус всех файлов с управлением версиями одновременно и произвести какие-то операции над версиями для набора файлов. Вы можете использовать команду `C-x v d` (`vc-directory`), чтобы создать распечатку каталога, который включает только файлы, имеющие отношение к системе управления версиями.

`C-x v d` создает буфер, использующий режим VC Dired. Он выглядит как обычный буфер Dired (см. Глава 28 [Dired], с. 291); однако, как правило в нем показаны только стоящие упоминания файлы (блокированные или необновленные). Это называется *сжатым показом*. Если вы установите переменную `vc-dired-terse-display` равной `nil`, то VC Dired показывает все относящиеся к делу файлы — те, что находятся под контролем системы управления версиями, плюс все подкаталоги (*полный показ*). Команда `v t` в буфере VC Dired переключает между сжатым и полным показом (см. Раздел 14.7.5.5 [Команды VC Dired], с. 123).

По умолчанию VC Dired производит рекурсивую распечатку заслуживающих упоминания или релевантных файлов в заданном каталоге и ниже. Вы можете изменить это, устанавливая переменную `vc-dired-recurse` равной `nil`; тогда VC Dired показывает только файлы текущего каталога.

Строка для отдельного файла показывает состояние версии на месте числа жестких ссылок, владельца, группу и размер файла. Если файл не изменен, синхронизирован с мастер-файлом, статус версии пуст. Иначе он состоит из текста в круглых скобках. Под RCS и SCCS показывается имя пользователя, блокировавшего этот файл; под CVS используется сокращенная версия вывода '`cvs status`'. Вот пример с использованием RCS:

```
/home/jim/project:

-rw-r-r- (jim)      Apr  2 23:39 file1
-r-r-r-            Apr  5 20:21 file2
```

Файлы '`file1`' и '`file2`' находятся под управлеием RCS, '`file1`' блокировал пользователь jim, а '`file2`' неблокирован.

Вот пример с использованием CVS:

```
/home/joe/develop:

-rw-r-r- (modified) Aug  2  1997 file1.c
-rw-r-r-            Apr  4 20:09 file2.c
-rw-r-r- (merge)    Sep 13  1996 file3.c
```

Здесь ‘`file1.c`’ изменен по отношению к репозиторию, а ‘`file2.c`’ не изменен. ‘`file3.c`’ изменен, но в репозитории были также зафиксированы другие изменения — вам нужно объединить их в рабочем файле перед фиксированием.

Когда VC Dired показывает подкаталоги (в режиме “полного” показа), он опускает некоторые из них, которые никогда не могут содержать файлов под контролем системы управления версиями. По умолчанию это включает каталоги, создаваемые этими системами, такие как ‘`RCS`’ и ‘`CVS`’; вы можете настраивать это, устанавливая переменную `vc-directory-exclusion-list`.

Вы можете подобрать подходящий формат VC Dired, набрав `C-u C-x v d` — как в обычном Dired, что позволяет вам указывать дополнительные ключи для команды `ls`.

#### 14.7.5.5 Команды VC Dired

Все обычные команды Dired работают как всегда и в режиме VC Dired, за исключением `v`, которая переопределена как префикс управления версиями. Вы можете вызывать команды VC, такие как `vc-diff` и `vc-print-log`, набирая `v =` или `v l` и так далее. Большинство этих команд применяются к имени файла на текущей строке.

Команда `v v` (`vc-next-action`) обрабатывает все помеченные файлы, так что вы можете заблокировать или зафиксировать несколько файлов одновременно. Если она работает более чем с одним файлом, то обрабатывает каждый файл в соответствии с его статусом; таким образом, она может блокировать один файл, но зафиксировать другой. Возможно, это смутит вас; но вы вольны избежать путаницы, помечая набор файлов с одним и тем же статусом.

Если какой-либо файл требует фиксирования, `v v` считывает единственную журнальную запись и использует ее для всех фиксируемых файлов. Это удобно для одновременного регистрирования или фиксирования нескольких файлов как частей одного изменения.

Вы можете в любое время переключаться между сжатым показом (только блокированные или необновленные файлы) и полным показом, набирая `v t vc-dired-toggle-terse-mode`. Есть также особая команда `* l` (`vc-dired-mark-locked`), которая помечает все заблокированные в данный момент файлы (или, для CVS, все необновленные). Таким образом, набор `* l t k` — это другой способ удалить из буфера все файлы, кроме тех, что сейчас заблокированы.

### 14.7.6 Множество ветвей файла

Одно из применений управления версиями — сопровождение нескольких “текущих” версий файла. Например, у вас могло бы быть несколько разных версий программы, в которой вы постепенно добавляли различные незавершенные новые возможности. Каждая независимая линия разработки называется *ветвью*. VC позволяет вам создавать ветви, переключаться между разными ветвями и вливать изменения из одной ветви в другую. Пожалуйста, заметьте однако, что такие ветви на данный момент поддерживаются только для RCS.

Главная линия развития файла обычно называется *стволом*. Версии ствола обычно нумеруются как 1.1, 1.2, 1.3, etc. На любой из этих версий вы можете начать независимую ветвь. Ветвь, начинающаяся на версии 1.2 имела бы номер 1.2.1.1, а последующие версии этой ветви имели бы номера 1.2.1.2, 1.2.1.3, 1.2.1.4 и так далее. Если есть вторая ветвь, также начинающаяся на версии 1.2, она состояла бы из версий 1.2.2.1, 1.2.2.2, 1.2.2.3, etc.

Если вы опускаете последний компонент в номере версии, это называется *номером ветви*. Он ссылается на самую верхнюю существующую версию этой ветви — ее *головную версию*. Ветви в примере выше имеют номера 1.2.1 и 1.2.2.

### 14.7.6.1 Переключение между ветвями

Чтобы переключиться между ветвями, введите `C-u C-x C-q` и укажите номер версии, который вы хотите выбрать. Тогда к этой версии делается обращение в *неблокированном* (защищенном от записи) режиме, так что вы можете просмотреть ее перед блокированием. Переключение ветвей таким способом допускается только тогда, когда файл блокирован.

Вы можете опустить номер второстепенной версии, задавая таким образом только номер ветви; это переносит вас к головной версии выбранной ветви. Если вы наберете просто ⟨RET⟩, Emacs переходит к самой верхней версии ствола.

После переключения на любую ветвь (включая главную), вы остаетесь в ней с точки зрения всех последующих команд VC, пока явно не выберете какую-то другую ветвь.

### 14.7.6.2 Создание новых ветвей

Чтобы создать новую ветвь из головной версии (последней в своей ветви), сначала выберите эту версию, если необходимо, заблокируйте ее командой `C-x C-q` и делайте нужные вам изменения. Затем, когда вы фиксируете изменения, используйте `C-u C-x C-q`. Это позволит вам задать номер для новой версии. Вы должны задать подходящий номер для ветви, отходящей от текущей версии. Например, если текущая версия — 2.5, то номер ветви должен быть 2.5.1, 2.5.2 и так далее в зависимости от номеров существующих ветвей в этой точке.

Чтобы создать новую ветвь от более старой версии (той, что уже не является головной в ветви), сначала выберите эту версию (см. Раздел 14.7.6.1 [Переключение ветвей], с. 124), затем заблокируйте ее с помощью `C-x C-q`. Когда вы блокируете старую версию, вас попросят подтвердить, что вы действительно имели в виду создание новой ветви, — если вы ответите нет, вам предоставят возможность блокировать вместо этого последнюю версию.

Потом вносите ваши изменения и снова наберите `C-x C-q`, чтобы зафиксировать новую версию. Это автоматически создаст новую ветвь от выбранной версии. Вам не нужно специально запрашивать создание новой ветви, поскольку это единственный способ добавить новую версию в точке, которая уже не находится в голове ветви.

После того как ветвь создана, вы "остаетесь" в ней. Это означает, что последующие фиксирования создают новые версии в этой ветви. Чтобы покинуть ветвь, вы должны явно выбрать другую версию с помощью `C-u C-x C-q`. Чтобы перенести изменения из одной ветви в другую, используйте команду объединения, описанную в следующем разделе.

### 14.7.6.3 Объединение ветвей

Когда вы закончили изменения в конкретной ветви, вам часто будет нужно внести их в главную линию разработки файла (ствол). Это нетривиальная процедура, потому что в стволе тоже могло идти развитие, поэтому вы должны *объединить* изменения с файлом, который уже был изменен иначе. VC позволяет вам сделать это (и другие вещи) при помощи команды `vc-merge`.

`C-x v m (vc-merge)`
: Вливает изменения в рабочий файл.

`C-x v m` (`vc-merge`) берет набор изменений и вливает их в текущую версию рабочего файла. Сначала она спрашивает у вас номер ветви или пару номеров версий в минибуфере. Затем она находит отличия от этой ветви или между двумя заданными версиями и объединяет их в текущей версии текущего файла.

В качестве примера предположим, что вы завершили некоторое добавление в ветви 1.3.1. Тем временем разработка ствола продвинулась до версии 1.5. Чтобы влить изменения в ствол, сначала перейдите в головную версию ствола, набрав `C-u C-x C-q RET`. Версия 1.5 теперь стала текущей. Если для этого файла используется блокирование, наберите `C-x C-q` для блокирования версии 1.5, чтобы вы могли ее изменять. Затем наберите `C-x v m 1.3.1 RET`. Это возьмет весь набор изменений в ветви 1.3.1 (относительно версии 1.3, где ветвь была начата, и до самой последней версии этой ветви) и вливает их в текущую версию рабочего файла. Теперь вы можете зафиксировать измененный файл, создавая таким образом версию 1.6, содержащую изменения из ветви.

После объединения можно делать дальнейшее редактирование до следующего фиксирования. Но обычно мудрее зафиксировать объединенную версию, затем блокировать ее и только тогда продолжать редактирование. Это сохранит лучшую запись истории изменений.

Когда вы вливаете изменения в файл, который сам был модифицирован, различия могут перекрываться. Мы называем такую ситуацию *конфликтом*, а согласование различий называется *разрешением конфликта*.

Когда во время объединения возникают конфликты, VC замечает их, говорит вам о них в эхо-области и спрашивает, хотите ли вы помочь в объединении. Если вы отвечаете да, VC запускает сеанс Ediff (см. раздел “Ediff” в *The Ediff Manual*).

Если вы говорите нет, в файл вставляются оба конфликтующих изменения, окруженные *маркерами конфликта*. Пример ниже показывает, как выглядят конфликтующие области; файл называется ‘**имя-файла**’, а номер текущей версии в мастер-файле, где находятся с изменения пользователя Б — 1.11.

```
<<<<<<< имя-файла
  Версия пользователя А
=======
  Версия пользователя Б
>>>>>>> 1.11
```

Теперь вы можете разрешить конфликт, редактируя файл вручную. Или вы можете напечатать `M-x vc-resolve-conflicts` после обращения к файлу. Это запускает сеанс Ediff, как описано выше.

### 14.7.6.4 Многопользовательские разветвления

Часто нескольким разработчикам бывает полезно работать одновременно над различными ветвями файла. CVS позволяет это по умолчанию; в RCS это возможно, если вы создадите несколько исходных каталогов. Каждый исходный каталог должен иметь ссылку с именем ‘`RCS`’, которая указывает на общий каталог с мастер-файлами RCS. Тогда каждый исходный каталог может хранить собственный набор выбранных версий, но все они разделяют одни общие записи RCS.

Этот метод работает надежно и автоматически, при условии, что исходные файлы содержат заголовки RCS о версии (см. Раздел 14.7.8.3 [Заголовки версии], с. 128). Эти заголовки позволяют Emacs всегда точно знать номер версии, присутсвующей в рабочем файле.

Если в файлах нет заголовков версии, вы должны в каждом сеансе явно говорить Emacs, над какой ветвью вы работаете. Чтобы сделать так, сначала обратитесь к файлу, затем наберите `C-u C-x C-q` и укажите правильный номер версии. Это должно гарантировать, что Emacs знает, какая ветвь используется во время конкретного сеанса редактирования.

### 14.7.7 Снимки

*Снимок* — это именованный набор версий файлов (одна для каждого зарегистрированного файла), с которыми вы можете обращаться как с одним целым. Один важный вид снимка назвается *выпуском*, это (теоретически) стабильная версия системы, готовая к распространению среди пользователей.

#### 14.7.7.1 Создание и использование снимков

Есть две основные команды для работы со снимками; одна создает снимок с заданным именем, а вторая извлекает именованный снимок.

`C-x v s` *имя* ⟨RET⟩
: Определяет последние сохраненные версии каждого зарегистрированного файла в текущем каталоге или ниже него как снимок с заданным *именем* (`vc-create-snapshot`).

`C-x v r` *имя* ⟨RET⟩
: Для всех зарегистрированных файлов на уровне текущего каталога или ниже выбирает версии, соответствующие снимку с заданным *именем* (`vc-retrieve-snapshot`).

  Эта команда сообщает об ошибке, если в текущем каталоге или ниже есть заблокированные файлы, и ничего не изменяет; это делается для предотвращения перезаписи редактируемых в данный момент файлов.

Снимок занимает очень небольшой объем ресурсов — ровно столько, сколько нужно для запоминания списка имен файлов и принадлежищих снимку версий. Поэтому нужно не колебаться и создавать снимки всегда, когда они могут быть полезными.

Вы можете предоставить в качестве аргумента для `C-x v =` или `C-x v ~` имя снимка (см. Раздел 14.7.4 [Старые версии], с. 120). Таким образом, вы можете использовать это для сравнения снимка с текущими файлами, или двух снимков друг с другом или снимка с заданной версией.

#### 14.7.7.2 Опасные места при работе со снимками

Работа со снимками в VC смоделирована на основе поддежки именованных конфигураций в RCS. Для нее используются встроенные средства RCS, поэтому снимки, сделанные под VC с использованием RCS, видны, даже когда вы обходите VC.

Для SCCS, VC реализует снимки сама. Используемые ей файлы содержат тройки имя/файл/номер-версии. Такие снимки видны только через VC.

Снимок — это набор зафиксированных версий. Поэтому при создании снимка вы должны убедиться, что все файлы зафиксированы и неблокированы.

Переименование и удаление файлов может создать некоторые трудности со снимками. Эта проблема не специфична для VC, но является общим вопросом в реализации систем управления версиями, который никем еще не решен хорошо.

Если вы переименовываете зарегистрированный файл, вам нужно переименовать и его мастер-файл (команда `vc-rename-file` делает это автоматически). Если вы пользуетесь SCCS, вы должны также обновить записи о снимках, чтобы они ссылались на этот файл по новому имени (`vc-rename-file` делает и это тоже). Старый снимок, ссылающийся на мастер-файл, который больше не существует под записанным именем, уже не корректен; VC больше не может извлечь его. Достаточное углубление в подробности об RCS и SCCS для объяснения процесса ручного обновления снимков вышло бы за рамки данного руководства.

Использование `vc-rename-file` сохраняет корректность снимка для извлечения, но не решает всех проблем. Например, некоторые файлы в программе вероятно ссылаются на другие файлы по именам. По самой меньшей мере, переименованный вами файл упомянут в Make-файле. Если вы извлекаете старый снимок, переименованный файл получает свое новое имя, а не то, которое ожидает Make-файл. Поэтому на самом деле программа не заработает в том виде, в каком ее извлекли.

### 14.7.8 Различные команды и возможности VC

Этот раздел рассказывает о других возможностях VC, применяемых не столь часто.

#### 14.7.8.1 Журналы изменений и VC

Если вы используете для программы RCS или CVS и также сопровождаете файл журнала ее изменений (см. Раздел 22.12 [Change Log], с. 224), вы можете автоматически генерировать вхождения для него из журнальных записей системы управления версиями:

`C-x v a` Обращается к журнальному файлу текущего каталога и создает для зарегистрированных файлов в этом каталоге новые вхождения для версий, зафиксированных позже последнего вхождения в этом журнальном файле (`vc-update-change-log`).

Эта команда работает только с RCS или CVS, но не с SCCS.

`C-u C-x v a`
Как выше, но находит вхождения только для файла текущего буфера.

`M-1 C-x v a`
Как выше, но находит вхождения для всех файлов, к которым вы обращаетесь, и которые сопровождаются системой управления версиями. Это работает только с RCS и также помещает все вхождения в журнал для каталога по умолчанию, что может не подходить.

Для примера предположим, что первая строка в ‘`ChangeLog`’ датирована 1999-04-10, и что с тех пор случилось только фиксирование, сделанное Натениэлом Боудичем для ‘`rcs2log`’ 1999-05-22 с журнальной записью ‘`Ignore log messages that start with ‘#’.`’. Тогда `C-x v a` обращается к ‘`ChangeLog`’ в вставляет подобный текст:

```
1999-05-22  Nathaniel Bowditch  <nat@apn.org>

        * rcs2log: Ignore log messages that start with ‘#’.
```

Теперь вы можете еще отредактировать новое вхождение в журнал по своему желанию.

К сожалению, метки в файлах ChangeLog сообщают только даты, поэтому некоторые новые журнальные записи могут продублировать то, что уже есть в ChangeLog. Вам придется удалить дублирования вручную.

Обычно вхождение в журнале для файла ‘`foo`’ отображается как ‘`* foo:` *текст вхождения*’. Знак ‘`:`’ после ‘`foo`’ опускается, если текст вхождения начинается со строки ‘`(`*имя-функции*`): `’. Например, если вхождение для ‘`vc.el`’ такое: ‘`(vc-do-command): Check call-process status.`’, то текст в ‘`ChangeLog`’ выглядит как:

```
1999-05-06  Nathaniel Bowditch  <nat@apn.org>

        * vc.el (vc-do-command): Check call-process status.
```

Когда `C-x v a` добавляет несколько вхождений одновременно, она группирует связанные между собой журнальные записи вместе, если все они зафиксированы одним автором

примерно в одно время. Если вхождения для нескольких таких файлов имеют одинаковый текст, она объединяет их в одно вхождение. Например, предположим, что последние фиксирования были с такими журнальными записями:

- Для ‘vc.texinfo’: ‘Fix expansion typos.’
- Для ‘vc.el’: ‘Don’t call expand-file-name.’
- Для ‘vc-hooks.el’: ‘Don’t call expand-file-name.’

В ‘ChangeLog’ они появятся так:

```
1999-04-01  Nathaniel Bowditch  <nat@apn.org>

        * vc.texinfo: Fix expansion typos.

        * vc.el, vc-hooks.el: Don't call expand-file-name.
```

Обычно C-x v a разделяет журнальные записи пустой строкой, но вы можете сделать так, чтобы несколько связанных записей сцеплялись вместе (без промежуточной пустой строки), начиная текст каждой из связанных журнальных записей с метки в форме ‘{*имя-сцепки*} ’. Сама метка не копируется в ‘ChangeLog’. Например, предположим, что есть такие журнальные записи:

- Для ‘vc.texinfo’: ‘{expand} Fix expansion typos.’
- Для ‘vc.el’: ‘{expand} Don’t call expand-file-name.’
- Для ‘vc-hooks.el’: ‘{expand} Don’t call expand-file-name.’

Тогда текст в ‘ChangeLog’ выглядит так:

```
1999-04-01  Nathaniel Bowditch  <nat@apn.org>

        * vc.texinfo: Fix expansion typos.
        * vc.el, vc-hooks.el: Don't call expand-file-name.
```

Журнальные записи, чей текст начинается с ‘#’, не копируются в ‘ChangeLog’. Например, если вы просто поправили грамматические ошибки в комментариях, вы можете сделать в журнале запись, начинающуюся с ‘#’, чтобы в ‘ChangeLog’ не заносились столь тривиальные вещи.

### 14.7.8.2 Переименование файлов под VC

Когда вы переименовываете зарегистрированный файл, вы должны переименовать также соответствующий мастер-файл, чтобы получить правильный результат. Используйте vc-rename-file, чтобы переименовать исходный файл, как вы укажете, и соответственно его мастер-файл. Это также обновит все снимки (см. Раздел 14.7.7 [Снимки], с. 126), которые упоминают данный файл, так что они будут использовать новое имя; несмотря на это, измененный таким образом снимок может не работать (см. Раздел 14.7.7.2 [Снимки Опасные места], с. 126).

Вы не можете использовать vc-rename-file для файла, который блокирован кем-то еще.

### 14.7.8.3 Вставка заголовков версий

Иногда удобно помещать строки для идентификации версии прямо в рабочие файлы. Некоторые особые строки, называемые *заголовками версии*, заменяются в каждой версии на ее номер.

Если вы пользуетесь RCS, и в ваших рабочих файлах присутствуют заголовки версий, Emacs может использовать их для определения текущей версии и состояния блокировки

этих файлов. Это более надежно, чем обращение к мастер-файлам, которое делается, если заголовков версий нет. Заметьте, что в среде с несколькими ветвями заголовки версий необходимы для корректной работы VC (см. Раздел 14.7.6.4 [Многопользовательские ветви], с. 125).

Переменная `vc-consult-headers` управляет поиском заголовков версий. Если она отлична от `nil`, Emacs производит поиск заголовков, чтобы узнать номер версии, которую вы редактируете. Установка этой переменной в `nil` выключает это средство.

Для вставки подходящей строки заголовка вы можете использовать команду `C-x v h` (`vc-insert-headers`).

`C-x v h` Вставляет в файл заголовки для использования с вашей системой управления версиями.

Стока заголовка по умолчанию — это '`$Id$`' для RCS и '`%W%`' для SCCS. Вы можете указать другие вставляемые заголовки, устанавливая переменную `vc-header-alist`. Ее значение — это список элементов в форме (*программа* . *строка*), где *программа* — это `RCS` или `SCCS`, а *строка* — это используемая строка.

Вместо одной строки вы можете задать список строк; тогда каждая строка из списка вставится как отдельный заголовок на отдельной строке.

Часто необходимо применять "излишние" обратные косые черты, когда вы пишете строки для этой переменной. Это нужно для того, чтобы такие строки не интерпретировались в константах как заголовки версий, если сам файл на Emacs Lisp находится под контролем системы управления версиями.

Каждый заголовок вставляется в точке на новой строке, в окружении знаков табуляции внутри ограничителей комментария. Как правило, используются обычные для текущего режима строки для начала и завершения комментария, но в некоторых режимах для этой цели есть особые ограничители комметариев; их определяет переменная `vc-comment-alist`. Каждый элемент в этом списке имеет форму (*режим начало конец*).

Переменная `vc-static-header-alist` указывает, какие еще строки должны добавляться в зависимости от имени буфера. Ее значение должно быть списком элементов в форме (*regexp* . *формат*). Когда *regexp* соответствует имени буфера, как часть загловка вставляется *формат*. Строка загловка вставляется для каждого элемента, совпадающего с именем буфера, и для каждой строки, указанной в `vc-header-alist`. Строка загловка получается путем форматирования строки из `vc-header-alist` с форматом, взятым из элемента `vc-static-header-alist`. По умолчанию `vc-static-header-alist` имеет следующее значение:

```
(("\\.c$" .
  "\n#ifndef lint\nstatic char vcid[] = \"\%s\";\n\
#endif /* lint */\n"))
```

Это определяет вставку текста в такой форме:

```
#ifndef lint
static char vcid[] = "строка";
#endif /* lint */
```

Заметьте, что текст выше начинается с пустой строки.

Если вы хотите использовать в файле более одного заголовка версии, помещайте их рядом. Механизм сохранения меток в `revert-buffer` может не обработать метки, расположенные между двумя заголовками.

### 14.7.9 Настройка VC

Есть много способов настройки VC. Параметры, которые вы можете установить, разделяются на четыре катагории, описанные в последующих разделах.

#### 14.7.9.1 Параметры для постпроцессора VC

Вы можете сказать RCS и CVS, должны ли они использовать для файла блокирование или не должны (см. Раздел 14.7.1.2 [Концепции VC], с. 117, для получения описания блокирования). VC автоматически распознает, что вы выбрали, и ведет себя соответственно.

В RCS по умолчанию применяется блокирование, но есть режим, называемый *нестрогим блокированием*, в котором вы можете фиксировать изменения без предварительного блокирования файла. Используйте '`rcs -U`' для переключения к нестрогому блокированию для некоторого файла, подробности смотрите в документе Man '`rcs`'.

Под CVS блокирование по умолчанию не применяется; каждый может изменять рабочий файл в любое время. Однако, есть способы ограничить это, дающие в результате поведение, похожее на блокирование.

С одной стороны, вы можете установить переменную среды `CVSREAD` в произвольное значение. Если эта переменная определена, CVS делает ваши рабочие файлы доступными только для чтения по умолчанию. В Emacs вы должны набрать `C-x C-q`, чтобы сделать файл доступным для записи, так что редактирование рабочих файлов фактически похоже на редактирование с применением блокирования. Заметьте однако, что фактически блокировки не происходит, поэтому несколько пользователей могут одновременно сделать свои рабочие файлы записываемыми. Когда вы устанавливаете `CVSREAD` первый раз, обязательно заново извлеките все ваши модули, чтобы защита файлов была правильно выставлена.

Другой способ достичь чего-то похожего на блокировку — воспользоваться средством *наблюдения* в CVS. Если над файлом установлено наблюдение, CVS по умолчанию делает его доступным только для чтения, и вы также должны использовать в Emacs `C-x C-q`, чтобы сделать его записываемым. Чтобы сделать файл доступным для записи, VC вызывает `cvs edit`, и CVS заботится о том, чтобы другие разработчики были оповещены о вашем намерении изменить этот файл. Подробности об использовании средства наблюдения смотрите в документации по CVS.

Вы можете подавить использование VC для файлов, управляемых CVS, установив переменную `vc-handle-cvs` в `nil`. Если вы сделаете так, Emacs считает эти файлы незарегистритованными, и команды VC становятся в них недоступными. Тогда вы должны делать все операции CVS вручную.

#### 14.7.9.2 Управление рабочими файлам в VC

Обычно Emacs не сохраняет резервные копии для исходных файлов, которые находятся под контролем системы управления версиями. Если вы хотите делать резервные копии даже для таких файлов, установите переменную `vc-make-backup-files` в отличное от `nil` значение.

Как правило рабочие файлы существуют всегда, блокированы они или нет. Если вы установите `vc-keep-workfiles` в `nil`, то при фиксировании новой версии с помощью `C-x C-q` рабочий файл будет удаляться; но любая попытка обратиться к этому файлу из Emacs создаст его снова. (Под CVS рабочие файлы остаются всегда.)

Редактирование файла в управлением версиями через символьную ссылку может быть опасным. Это обходит систему управления версиями — вы можете редактировать файлы, не блокируя их, и у вас может не получиться зафиксировать свои изменения. Также, ваши изменения могут перезаписать версию другого пользователя. Чтобы предоставить защиту от этого, VC проверяет каждую символьную ссылку, к которой вы обращаетесь, чтобы узнать, не указывает ли она на файл под контролем системы управления версиями.

Переменная `vc-follow-symlinks` говорит, что нужно делать, когда символьная ссылка указывает на файл с управлением версиями. Если она равна `nil`, VC лишь показывает предупреждающее сообщение. Если это `t`, VC автоматически следует по ссылке и обращается к настоящему файлу, сообщая вам об этом в эхо-области. Если это значение равно `ask` (по умолчанию), VC всякий раз спрашивает вас, нужно ли проследовать по ссылке.

#### 14.7.9.3 Как VC узнает статус файла

При вычислении статуса блокировки файла, VC сначала ищет в нем строку заголовка версии RCS (см. Раздел 14.7.8.3 [Заголовки версии], с. 128). Если строки заголовка нет, или если вы пользуетесь SCCS, VC обычно смотрит на разрешения, установленные для рабочего файла; это быстро. Но могут быть ситуации, когда разрешениям нельзя доверять. В таком случае делается обращение к мастер-файлу, что довольно дорого. Кроме того, мастер-файл может *только* сказать, заблокирован ли файл кем-то, но не то, что ваш рабочий файл содержит именно эту блокированную версию.

Вы можете указать VC не использовать заголовки версии для определения статуса блокировки, установив `vc-consult-headers` в `nil`. Тогда VC всегда использует для этого биты разрешений для файла (если она может им доверять) или проверяет мастер-файл.

Вы можете задать критерий, по которому следует доверять разрешениям для файла, с помощью переменной `vc-mistrust-permissions`. Ее значением может быть `t` (никогда не доверять битам разрешений и всегда проверять мастер-файл), `nil` (всегда доверять разрешениям) или функция с одним аргументом, которая принимает решение. Аргумент — это имя подкаталога '`RCS`', '`CVS`' или '`SCCS`'. Отличное от `nil` значение из этой функции говорит, что битам разрешений доверять нельзя. Если вы обнаружили, что разрешения для рабочего файла выстановлены ошибочно, установите `vc-mistrust-permissions` в `t`. Тогда VC всегда будет проверять мастер-файл для определения статуса файла.

#### 14.7.9.4 Выполнение команд в VC

Если `vc-suppress-confirm` отлична от `nil`, то `C-x C-q` и `C-x v i` могут сохранять текущий буфер без запроса, и `C-x v u` также работает без подтверждения. (Эта переменная не влияет на `C-x v c`; данная операция насколько значительна, что для нее всегда нужно спрашивать подтверждение.)

Режим VC делает большую часть работы, запуская команды оболочки для RCS, CVS и SCCS. Если `vc-command-messages` отлична от `nil`, VC выводит сообщения, показывающие, какие команды оболочки запускаются, и дополнительные сообщения, когда команда завершается.

Вы можете задать дополнительные каталоги для поиска программ управления версиями, устанавливая переменную `vc-path`. Эти каталоги просматриваются перед обычным путем поиска. Но обычно правильные файлы находятся автоматически.

### 14.8 Каталоги файлов

Файловая система группирует файлы по *каталогам*. *Распечатка каталога* — это список всех файлов в каталоге. Emacs предоставляет команды для создания и удаления каталогов и для выдачи распечатки каталогов в кратком формате (только имена файлов) и в подробном формате (включающем размеры, даты и авторов). Есть также броузер каталогов, называемый Dired; смотрите Глава 28 [Dired], с. 291.

`C-x C-d` *кат-или-образец* ⟨RET⟩
: Выводит краткую распечатку каталога (`list-directory`).

`C-u C-x C-d` *кат-или-образец* ⟨RET⟩
: Выводит подробную распечатку каталога.

`M-x make-directory` ⟨RET⟩ *имя-кат* ⟨RET⟩
: Создает новый каталог с именем *имя-кат*.

`M-x delete-directory` ⟨RET⟩ *имя-кат* ⟨RET⟩
: Удаляет каталог с именем *имя-кат*. Он должен быть пуст, иначе вы получите ошибку.

Команда для вывода распечатки каталога — это `C-x C-d` (`list-directory`). Она считывает, используя минибуфер, имя файла, который является либо каталогом, который нужно распечатать, либо шаблоном имен файлов, которые нужно перечислить. Например,

```
C-x C-d /u2/emacs/etc ⟨RET⟩
```

перечисляет все файлы в каталоге '`/u2/emacs/etc`'. Вот пример описания образца имен файлов:

```
C-x C-d /u2/emacs/src/*.c ⟨RET⟩
```

Обычно `C-x C-d` печатает краткий перечень каталога, содержащий только имена файлов. Числовой аргумент (независимо от значения) велит показывать подробную распечатку (как '`ls -l`').

Текст распечатки каталога получается от `ls`, запущенной в подчиненном процессе. Две переменные Emacs управляют ключами, передаваемыми `ls`: `list-directory-brief-switches` — это строка, дающая ключи для использования в кратких распечатках (`"-CF"` по умолчанию), и `list-directory-verbose-switches` — ключи для подробной распечатки (`"-l"` по умолчанию).

## 14.9 Сравнение файлов

Команда `M-x diff` сравнивает два файла, показывая различия в буфере Emacs с именем '`*Diff*`'. Она запускает программу `diff`, используя ключи, получаемые из переменной `diff-switches`, чье значение должно быть строкой.

Буфер '`*Diff*`' имеет в качестве основного режим Compilation, поэтому вы можете использовать `C-x '`, чтобы последовательно обратиться к изменившимся местам в двух исходных файлах. Вы также можете перейти к конкретному ломтю изменений и нажать ⟨RET⟩ или `C-c C-c`, или щелкнуть на нем `Mouse-2`, чтобы перейти к соответствующей позиции в исходном тексте. Вы также можете использовать другие особые команды режима Compilation: ⟨SPC⟩ и ⟨DEL⟩ для прокрутки и `M-p` и `M-n` для передвижения курсора. См. Раздел 23.1 [Компиляция], с. 247.

Команда `M-x diff-backup` сравнивает заданный файл с его самой последней резервной копией. Если вы задали имя резервного файла, `diff-backup` сравнивает его с исходным файлом.

Команда `M-x compare-windows` сравнивает текст в текущем окне с текстом следующего окна. Сравнение начинается от точки в каждом окне, и обе начальные позиции вталкиваются в список пометок соответствующего буфера. Затем точка перемещается в каждом окне вперед по одному знаку, пока не будет найдено несовпадение. Тогда эта команда останавливается. Для получения большей информации об окнах в Emacs смотрите Глава 16 [Окна], с. 141.

С числовым аргументом, `compare-windows` игнорирует изменения в пропусках. Если переменная `compare-ignore-case` не `nil`, она игнорирует также и различия в регистре букв.

Смотрите также Раздел 22.14 [Emerge], с. 231, описание удобных средств для слияния двух похожих файлов.

## 14.10 Разнообразные действия над файлами

В Emacs есть команды для произведения многих других операций над файлами. Все они действуют на один файл; они не воспринимают имена файлов с шаблонами.

`M-x view-file` позволяет вам просмотреть или прочитать файл по целым экранам. Она считывает имя файла, используя минибуфер. После загрузки файла в буфер Emacs, `view-file` показывает его начало. Теперь вы можете нажать ⟨SPC⟩, чтобы прокрутить вперед на

целое окно, или ⟨DEL⟩ для прокрутки назад. Предоставляются и другие различные команды для перемещения по файлу, но не для его изменения; наберите во время просмотра `?`, чтобы получить их перечень. Это практически те же обычные команды Emacs для передвижения курсора. Чтобы выйти из просмотра, наберите `q`. Команды просмотра определяются особым основным режимом, называемом режимом View.

Есть родственная команда, `M-x view-buffer`, для просмотра буфера, уже существующего в Emacs. См. Раздел 15.3 [Буферы Разное], с. 136.

`M-x insert-file` вставляет копию содержимого заданного файла в текущий буфер в позиции точки, оставляя точку неизменной перед вставленным и метку после него.

`M-x write-region` — это обращение `M-x insert-file`; она копирует содержимое области в указанный файл. `M-x append-to-file` добавляет текст области в конец заданного файла. См. Раздел 9.3 [Накопление текста], с. 73.

`M-x delete-file` удаляет указанный файл, как команда оболочки `rm`. Если вы хотите удалить много файлов в одном каталоге, может оказаться удобнее воспользоваться Dired (см. Глава 28 [Dired], с. 291).

`M-x rename-file` считывает, используя минибуфер, два имени файла, *старый* и *новый*, а затем переименоывает файл *старый* в *новый*. Если файл с именем *новый* уже существует, вы должны подтвердить переименование вводом `yes`, или переименование не производится; это сделано, потому что такое переименование приведет к потере старого значения имени *новый*. Если *старый* и *новый* находятся на разных файловых системах, файл *старый* копируется и удаляется.

Похожая команда `M-x add-name-to-file` используется для добавления еще одного имени существующему файлу без удаления старого имени. Новое имя обязано принадлежать той же файловой системе, где находится сам файл.

`M-x copy-file` считывает файл *старый* и записывает новый файл с именем *новый* с тем же содержимым. Если файл с именем *новый* уже существует, требуется подтверждение, потому что копирование затирает старое содержимое файла *новый*.

`M-x make-symbolic-link` считывает два имени файла, *цель* и *имя-ссылки*, а затем создает символьную ссылку с именем *имя-ссылки*, указывающую на *цель*. Это проявится в том, что будущие попытки открыть файл *имя-ссылки* получат тот файл, который называется *цель* во время открытия, или получат ошибку, если имя *цель* в это время не используется. Эта команда не раскрывает аргумент *цель*, поэтому она позволяет вам указать относительное имя в качестве файла назначения ссылки.

Если *имя-ссылки* занято, требуется подтверждение. Обратите внимание, не все системы поддерживают символьные ссылки.

## 14.11 Доступ к сжатым файлам

Emacs поставляется с библиотекой, которая автоматически распаковывает сжатые файлы, когда вы к ним обращаетесь, и автоматически сжимает их снова, если вы их изменили и сохраняете. Чтобы задействовать эту возможность, наберите команду `M-x auto-compression-mode`.

Когда включено автоматическое сжатие (что подразумевает и автоматическую распаковку), Emacs распознает сжатые файлы по именам. Имена файлов, завершающиеся на ‘`.gz`’, указывают, что этот файл сжат программой `gzip`. Другие окончания обозначают другие методы сжатия.

Автоматическая распаковка и сжатие применяются ко всем операциям, в которых Emacs использует содержимое файлов. Это включает обращение, сохранение, вставку содержимого в буфер, загрузку и байт-компиляцию.

## 14.12 Удаленные файлы

Вы можете сослаться на файлы на других машинах, используя особый синтаксис имен:

*/машина:имя-файла*
*/пользователь@машина:имя-файла*

Когда вы делаете так, Emacs использует для чтения и записи файлов на указанной машине программу FTP. Он заходит через FTP, используя ваше пользовательское имя или имя *пользователь*. Он может спрашивать у вас пароль время от времени; это используется для захода на *машину*.

Обычно, если вы не задали имя пользователя в имени удаленного файла, это означает, что нужно использовать ваше собственное пользовательское имя. Но если вы установите переменную `ange-ftp-default-user` равной какой-то строке, то будет использоваться эта строка. (Пакет Emacs, который реализует доступ к файлам по FTP, называется `ange-ftp`.)

Вы можете полностью выключить обработку имен FTP-файлов, установив переменную `file-name-handler-alist` в значение `nil`.

## 14.13 Буквальные имена файлов

Вы можете *отменить особый смысл* абсолютного имени файла, чтобы блокировать действие специальных символов и синтаксиса. Это можно сделать, добавив '`/:`' в начале.

Например, вы можете отменить особый смысл имени локального файла, который выглядит как удаленный, чтобы предотвратить его трактовку как имени удаленного файла. Таким образом, если у вас есть каталог с именем '`/foo:`' и в нем файл с именем '`bar`', вы можете сослаться в Emacs на этот файл как на '`/:/foo:/bar`'.

'`/:`' также может предотвратить понимание '`~`' как специального символа, обозначающего начальный каталог этого пользователя. Например, '`/:/tmp/~hack`' ссылается на файл с именем '`~hack`' в каталоге '`/tmp`'.

Аналогично, использование '`/:`' — это один из способов ввести в минибуфере имя файла, содержащее '`$`'. Однако, '`/:`' обязана быть в начале этого буфера, чтобы блокировать эффект '`$`'.

С помощью '`/:`' вы также можете подавить эффект символов подстановки при обращении к файлам. Например, '`/:/tmp/foo*bar`' обращается к файлу '`/tmp/foo*bar`'. Однако, чаще всего вы просто можете ввести символы подстановки как есть. Например, если единственное имя файла в '`/tmp`', которое начинается на '`foo`' и завершается на '`bar`' — это '`foo*bar`', то указав '`/tmp/foo*bar`' вы обратитесь только к '`/tmp/foo*bar`'.

# 15 Использование множества буферов

Текст, который вы редактируете в Emacs, находится в объекте, называемом *буфером*. Каждый раз, когда вы обращаетесь к файлу, для хранения его текста создается буфер. Каждый раз, когда вы запускаете Dired, создается буфер, содержащий список каталога. Если вы посылаете сообщение с помощью `C-x m`, то для текста этого сообщения используется буфер с именем '`*mail*`'. Когда вы запрашиваете документацию команды, она появится в буфере с именем '`*Help*`'.

В любой момент один и только один буфер является *выбранным*. Он также называется *текущим буфером*. Часто мы говорим, что команда действует в "буфере", как если бы он был только один; но на самом деле это означает, что команда действует в выбранном буфере (большинство команд так и делают).

Когда Emacs создает множество окон, каждое окно имеет свой выбранный буфер, но в любой момент времени только одно из окон является выбранным, и его буфер — это выбранный буфер. Строка режима каждого окна показывает имя буфера, который в нем отображен (см. Глава 16 [Окна], с. 141).

У каждого буфера есть имя, которое может быть произвольной длины, и вы можете выбрать любой буфер по имени. Большинство буферов создаются при обращении к файлам, и их имена производятся из имени файла. Но вы можете также создать пустой буфер с любым именем, каким захотите. Только что запущенный Emacs несет один буфер с именем '`*scratch*`', который может быть использован для вычисления выражений Лиспа в Emacs. В именах буферов имеет значение различие между верхним и нижним регистрами.

Каждый буфер записывает отдельно, к какому файлу он обращается, изменен ли он, и какие основной и второстепенные режимы в нем действуют (см. Глава 19 [Основные режимы], с. 175). Любая переменная Emacs может быть сделана *локальной* для конкретного буфера; имеется в виду, что ее значение в этом буфере может отличаться от ее значения в других буферах. См. Раздел 31.2.4 [Локальные переменные], с. 350.

## 15.1 Создание и выбор буферов

`C-x b` *буфер* ⟨RET⟩
: Выбрать или создать буфер с именем *буфер* (`switch-to-buffer`).

`C-x 4 b` *буфер* ⟨RET⟩
: Аналогично, но выбирает *буфер* в другом окне (`switch-to-buffer-other-window`).

`C-x 5 b` *буфер* ⟨RET⟩
: Аналогично, но выбирает *буфер* в другом фрейме (`switch-to-buffer-other-frame`).

Чтобы выбрать буфер с именем *имя-буфера*, наберите `C-x b` *имя-буфера* ⟨RET⟩. Это запустит команду `switch-to-buffer` с аргументом *имя-буфера*. Вы можете применить завершение сокращенного имени желаемого буфера (см. Раздел 5.3 [Завершение], с. 47). Пустой аргумент для `C-x b` задает последний выбранный буфер, который не отображен ни в одном окне.

Большинство буферов создаются при обращении к файлам или же командами Emacs, которые хотят показать некоторый текст, но вы также можете явно создать буфер, набрав `C-x b` *имя-буфера* ⟨RET⟩. Эта команда создает новый, пустой буфер, который не обращается ни к какому файлу, и выберет его для редактирования. Такие буферы создаются для заметок. Если вы попытаетесь сохранить этот буфер, то у вас спросят имя файла для записи. Основной режим в новом буфере определяется значением переменной `default-major-mode` (см. Глава 19 [Основные режимы], с. 175).

Отметим, что `C-x C-f` и любая другая команда для обращения к файлу также могут использоваться для переключения к существующему буферу, обращающемуся к файлу. См. Раздел 14.2 [Обращение], с. 106.

Emacs использует буферы с именами, начинающимися с пробела, для внутренних целей. Отчасти он обращается с буферами с такими именами особенным образом — например, по умолчанию в них не записывается информация для отмены изменений. Вам лучше избегать использования таких имен для буферов.

## 15.2 Перечисление существующих буферов

`C-x C-b` Перечисляет существующие буферы (`list-buffers`).

Чтобы напечатать список всех существующих буферов, наберите `C-x C-b`. Каждая строка в списке показывает имя одного буфера, его основной режим и файл, к которому он обращается. Буферы перечисляются в том порядке, в котором они были текущими; буферы, которые были текущими недавно, идут первыми.

Знак '`*`' в начале строки указывает, что буфер "модифицирован". Если модифицированы несколько буферов, то может быть пора записать некоторые из них при помощи `C-x s` (см. Раздел 14.3 [Сохранение], с. 108). Знаком '`%`' обозначаются буферы, доступные только для чтения. Знак '`.`' отмечает выбранный буфер. Вот пример перечня буферов:

```
 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
```

Отметим, что буфер '`*Help*`' создается при запросе справки, и это не является обращением к какому-либо файлу. Буфер '`man`' был создан Dired для каталога '`/u2/emacs/man/`'.

## 15.3 Разнообразные операции над буфером

`C-x C-q` Переключить доступ на запись в буфер (`vc-toggle-read-only`).

`M-x rename-buffer` ⟨RET⟩ *имя* ⟨RET⟩
: Изменить имя текущего буфера.

`M-x rename-uniquely`
: Переименовать буфер добавлением '`<`*числа*`>`' в конец имени.

`M-x view-buffer` ⟨RET⟩ *буфер* ⟨RET⟩
: Просмотреть *буфер*.

Буфер может быть *доступен только для чтения*, что означает, что команды для изменения его содержимого не разрешены. Строка режима указывает на то, что буфер доступен только для чтения, знаками '`%%`' или '`%*`' около левого края. Буферы только для чтения обычно создаются подсистемами вроде Dired и Rmail, которые имеют специальные команды для действий над текстом; буфер только для чтения создается также, если вы обращаетесь к файлу, для которого у вас нет доступа на запись.

Если вы хотите сделать изменения в буфере, предназначенном только для чтения, используйте команду `C-x C-q` (`vc-toggle-read-only`). Она делает буфер, доступный только для чтения, доступным для записи, а буфер, доступный для записи — доступным только для чтения. В большинстве случаев эта команда работает, устанавливая переменную `buffer-read-only`, которая имеет локальное значение в каждом буфере и делает буфер закрытым для записи, если ее значение не `nil`. Если этот файл сопровождается с контролем версий, то `C-x C-q` работает через систему управления версиями и изменяет состояния доступа на запись как для файла, так и для буфера. См. Раздел 14.7 [Управление версиями], с. 116.

`M-x rename-buffer` изменяет имя текущего буфера. Новое имя задается как аргумент минибуфера. Значения по умолчанию нет. Если вы напишете имя, которое используется для какого-то другого буфера, то происходит ошибка, и переименование не делается.

`M-x rename-uniquely` переименовывает текущий буфер в похожее имя с добавленным числовым окончанием и делает это имя одновременно уникальным и отличным от других. Этой команде не нужен аргумент. Она полезна для создания нескольких буферов с оболочкой: если вы переименуете буфер '`*Shell*`' и снова сделаете `M-x shell`, то создастся новый буфер оболочки с именем '`*Shell*`'; тем временем старый буфер оболочки продолжает существовать под своим новым именем. Этот метод также полезен для буферов почтовых сообщений, буферов компиляции и большинства программ в Emacs, которые создают специальные буферы с конкретными именами.

`M-x view-buffer` очень похожа на `M-x view-file` (см. Раздел 14.10 [Файлы Разное], с. 132), за исключением того, что она показывает уже существующий буфер Emacs. Режим View предусматривает команды для удобной прокрутки буфера, но не для его изменения. Когда вы выходите из режима View с помощью `q`, вы переключаетесь назад к тому буферу (и позиции), который прежде отображался в этом окне. Или вы можете выйти из режима View с помощью `e`, в результате после прочтения сохранятся буфер и значение точки.

Команды `M-x append-to-buffer` and `M-x insert-buffer` можно использовать для копирования текста из одного буфера в другой. См. Раздел 9.3 [Накопление текста], с. 73.

## 15.4 Уничтожение буферов

Если вы продолжаете работу в Emacs довольно долго, вы можете накопить большое количество буферов. Тогда вы можете посчитать удобным *уничтожить* те из них, в которых вы больше не нуждаетесь. В большинстве операционных систем уничтожение буфера освобождает занимаемое им пространство, так что его смогут использовать другие процессы. Вот несколько команд для уничтожения буферов:

`C-x k` *имя-буфера* ⟨RET⟩
: Уничтожает буфер *имя-буфера* (`kill-buffer`).

`M-x kill-some-buffers`
: Предлагает уничтожить каждый буфер один за другим.

`C-x k` (`kill-buffer`) уничтожает один буфер, чье имя задано в минибуфере. По умолчанию, если вы наберете в минибуфере просто ⟨RET⟩, уничтожится текущий буфер. Если уничтожается текущий буфер, то выбранным становится другой буфер — тот, который выбрали недавно, но сейчас он не виден ни в одном окне. Если вы попросили уничтожить буфер, который обращается к файлу и был модифицирован (содержит несохраненные изменения), тогда вас спросят о подтверждении с `yes` перед тем, как буфер будет уничтожен.

Команда `M-x kill-some-buffers` спрашивает о каждом буфере, один за другим. Ответ `y` означает уничтожить буфер. Уничтожение текущего буфера или буфера, содержащего незаписанные изменения, выбирает новый буфер или требует такого же подтверждения, как и `kill-buffer`.

Меню буферов (см. Раздел 15.5 [Несколько буферов], с. 138) также предоставляет удобный способ уничтожения различных буферов.

Если вы хотите делать что-то особенное каждый раз, когда уничтожается буфер, вы можете добавить свои функции в ловушку `kill-buffer-hook` (см. Раздел 31.2.3 [Ловушки], с. 349).

Если вы продолжаете один сеанс Emacs в течении многих дней, как делают многие, он может наполниться буферами, которые вы использовали несколько дней назад. Команда `M-x clean-buffer-list` — это удобный способ очистить их; она уничтожает все неизмененные буферы, которые вы не использовали долгое время. Обычные буферы уничтожаются, если они не отображались в течении трех дней; однако, вы можете указать определенные буферы, которые никогда не должны уничтожаться автоматически, и другие, которые нужно уничтожать, если их не использовали хотя бы час.

Вы также можете сделать так, чтобы эти буферы очищались для вас сами ежедневно в полночь, включив режим Midnight. Режим Midnight работает каждый день в полночь; в это время он запускает `clean-buffer-list` или другие функции, которые вы поместите в обычную ловушку `midnight-hook` (см. Раздел 31.2.3 [Ловушки], с. 349).

Чтобы включить режим Midnight, используйте буфер Customization для установки переменной `midnight-mode` в значение `t`. См. Раздел 31.2.2 [Простая настройка], с. 344.

## 15.5 Действия над несколькими буферами

*Меню буферов* похоже на “Dired для буферов”. Оно позволяет вам запрашивать действия для различных буферов Emacs при помощи редактирования буфера, содержащего их список. Вы можете сохранять буферы, уничтожать их (здесь это называется *удалением* для согласования с Dired) или показывать их.

`M-x buffer-menu`
: Начать редактирование буфера, содержащего список всех буферов Emacs.

Команда `buffer-menu` записывает список всех буферов Emacs в буфер с именем ‘`*Buffer List*`’ и выбирает этот буфер в режиме Buffer Menu. Этот буфер предназначен только для чтения и может быть изменен только при помощи специальных команд, описанных в этом разделе. В буфере ‘`*Buffer List*`’ могут использоваться обычные команды Emacs для движения курсора. Следующие специальные команды применяются к буферу, описанному в текущей строке:

`d`
: Запросить удаление (уничтожение) буфера, и затем сдвинуться вниз. Запрос показывается как ‘`D`’ на строке перед именем буфера. Затребованные удаления происходят, когда вы печатаете команду `x`.

`C-d`
: Как `d`, но сдвигает вверх, а не вниз.

`s`
: Запросить сохранение буфера. Запрос показывается как ‘`S`’ на строке. Требуемые сохранения происходят, когда используется команда `x`. Вы можете запросить и запись, и удаление для одного и того же буфера.

`x`
: Выполнить ранее запрошенные удаления и сохранения.

`u`
: Уничтожить любой запрос, сделанный для текущей строки, и сдвинуть курсор вниз.

⟨DEL⟩
: Сдвинуть курсор к предыдущей строке и уничтожить любой запрос, сделанный для этой строки.

Команды `d`, `C-d`, `s` и `u` для добавления или сброса флагов также перемещают вниз (или вверх) на одну строку. Они принимают числовой аргумент в качестве счетчика повторов.

Эти команды действуют сразу на буфер, перечисленный в текущей строке:

`~` Пометить буфер как “неизмененный”. Команда `~` делает это немедленно после того, как вы ее ввели.

`%` Переключает флаг доступности на запись для буфера. Команда `%` действует сразу после ввода.

`t` Обратиться к буферу как к таблице тегов. См. Раздел 22.13.3 [Выбор таблицы тегов], с. 228.

Есть также команды для выбора другого буфера или нескольких буферов:

`q` Выйти из меню буферов — сразу отобразить последний буфер, бывший прежде видимым на его месте.

⟨RET⟩
`f` Немедленно выбрать буфер на этой строке на место буфера ‘`*Buffer List*`’.

`o` Немедленно выбрать буфер на этой строке в другом окне, как если бы это было сделано при помощи `C-x 4 b`, оставляя ‘`*Buffer List*`’ видимым.

`C-o` Немедленно отобразить буфер на этой строке в другом окне, но не выбирать это окно.

`1` Немедленно выбрать буфер на этой строке в полноэкранном окне.

`2` Немедленно создать два окна, одно с буфером на этой строке, а второе с предыдущим выбранным буфером (помимо буфера ‘`*Buffer List*`’).

`b` Спрятать буфер, перечисленный в текущей строке.

`m` Пометить этот буфер для показа в другом окне, если вы выйдите с помощью команды `q`. Такой запрос показывается как ‘`>`’ в начале строки. (Один и тот же буфер не может иметь и запрос на показ, и запрос на удаление.)

`v` Немедленно выбрать буфер на этой строке, а также показать в других окнах любые буферы, прежде помеченные с помощью команды `m`. Если таких буферов нет, то эта команда эквивалентна `1`.

Все, что `buffer-menu` делает непосредственно, — это создает и выбирает подходящий буфер и включает режим Buffer Menu. Все остальное из описанного выше осуществляется при помощи специальных команд, предоставляемых в режиме Buffer Menu. Одним из следствий этого является то, что вы можете переключиться из буфера ‘`*Buffer List*`’ в другой буфер Emacs и редактировать там. Вы можете выбрать заново буфер ‘`*Buffer List*`’ позже, чтобы исполнить уже запрошенные действия, или вы можете уничтожить его или больше не обращать на него внимания.

Существует только одно различие между `buffer-menu` и `list-buffers` — это то, что `buffer-menu` выбирает буфер ‘`*Buffer List*`’ в текущем окне, а `list-buffers` отображает его в другом окне. Если вы запускаете `list-buffers` (это происходит при наборе `C-x C-b`) и выбираете список буферов вручную, то вы можете использовать все описанные здесь команды.

Буфер ‘`*Buffer List*`’ не обновляется автоматически, когда создаются или уничтожаются буферы; его содержимое — это просто текст. Если вы создавали, удаляли или переименовывали буферы, вы можете обновить ‘`*Buffer List*`’, чтобы он показывал то, что вы сделали, напечатав `g` (`revert-buffer`) или повторив команду `buffer-menu`.

## 15.6 Косвенные буферы

*Косвенный буфер* разделяет текст с каким-то другим буфером, называемым *базовым буфером* косвенного буфера. Это своего рода аналог символьных ссылок на файлы, но для буферов.

`M-x make-indirect-buffer` *базовый-буфер* ⟨RET⟩ *косвенное-имя* ⟨RET⟩
: Создает косвенный буфер с именем *косвенное-имя*, чей базовый буфер — это *базовый-буфер*.

Текст в косвенном буфере всегда идентичен тексту его базового буфера; изменения, сделанные в одном, сразу же становятся видны в другом. Но во всем остальном косвенный буфер и его базовый буфер абсолютно различны. У них разные имена, разные значения точки, разное сужение, разные пометки, разные основные режимы и разные локальные переменные.

Косвенные буферы не могут обращаться к файлам, но их базовые буферы могут. Если вы пытаетесь записать косвенный буфер, это в действительности работает как запись базового буфера. Уничтожение базового буфера уничтожает косвенный буфер, но уничтожение косвенного буфера не влияет на базовый буфер.

Один из способов применения косвенных буферов — отображение нескольких различных видов схемы текста. См. Раздел 21.8.4 [Несколько видов для Outline], с. 193.

# 16 Множество окон

Emacs может делить фреймы на два или более окна. Несколько окон могут отражать части разных буферов или разные части одного буфера. Использование нескольких фреймов всегда подразумевает множество окон, потому что в каждом фрейме свой набор окон. Каждое окно принадлежит одному и только одному фрейму.

## 16.1 Понятие окна в Emacs

Каждое окно Emacs отображает в одно время один буфер. Один и тот же буфер может появиться более чем в одном окне; если это произошло, то любые изменения в его тексте показываются во всех окнах, где он отображен. Но окна, показывающие один и тот же буфер, могут показывать различные его части, так как каждое окно хранит свое собственное значение точки.

В любой момент одно из окон является *выбранным окном*; буфер, отображаемый этим окном, является текущим буфером. Курсор терминала показывает позицию точки в этом окне. Все другие окна также имеют позицию точки, но так у терминала есть только один курсор, нельзя показать, где находятся эти позиции. Когда есть несколько видимых фреймов в X Windows, в каждом фрейме изображается один курсор, находящийся в выбранном окне этого фрейма. Курсор в выбранном фрейме закрашен; курсоры в других фреймах выглядят как пустые клетки.

Команды движения точки действуют только на значение точки для выбранного окна Emacs. Они не изменяют значение точки в каком-либо другом окне Emacs, даже в показывающем тот же самый буфер. Это верно и для таких команд, как `C-x b` для изменения выбранного буфера в выбранном окне, на другие окна они не действуют вообще. Однако, существуют другие команды, например `C-x 4 b`, которые выбирают другое окно и переключают буферы в нем. Также, все команды, которые показывают информацию в окне, включая (например) `C-h f` (`discribe-function`) и `C-x C-b` (`list-bufers`), работают при помощи переключения буферов в невыбранном окне, не затрагивая выбранное.

Когда несколько окон показывают один и тот же буфер, в них могут быть разные области, потому что они могут иметь разные значения точки. Однако, все они имеют одно и то же значение метки, потому что в каждом буфере может быть только одна позиция метки.

Каждое окно имеет свою собственную строку режима, которая показывает имя буфера, статус модификации и основной и второстепенные режимы буфера, который отражен в данном окне. См. Раздел 1.3 [Строка режима], с. 25, для более подробной информации о строке режима.

## 16.2 Разделение окон

`C-x 2` Разделить выбранное окно на два, находящихся одно под другим (`split-window-vertically`).

`C-x 3` Разделить выбранное окно на два окна, находящихся одно рядом с другим (`split-window-horizontally`).

`C-Mouse-2`
В строке режима или полосе прокрутки, разделяет это окно.

Команда `C-x 2` (`split-window-vertically`) разбивает выбранное окно на два, одно под другим. Оба окна сначала показывают один и тот же буфер с одним и тем же значением точки. По умолчанию каждое из двух окон получает половину высоты окна, которое было разделено; числовой аргумент определяет количество строк, которое необходимо дать верхнему окну.

`C-x 3` (`split-window-horizontally`) разбивает выбранное окно на два рядом стоящих окна. Числовой аргумент определяет, сколько столбцов дать левому окну. Окна разделяются строкой вертикальных штрихов. Окна, которые не занимают всю ширину фрейма, имеют строки режима, но они усечены. На терминалах, где Emacs не поддерживает подсветку, усеченные строки режима иногда появляются не в инверсном изображении.

Вы можете разделить окно горизонтально, щелкнув `C-Mouse-2` на строке режима или полосе прокрутки. Линия раздела проходит от места, где вы щелкнули: если вы щелкнули на строке режима, от этого места отойдет новая полоса прокрутки; если вы щелкнули на полосе прокрутки, в месте вашего щелчка появится строка режима нового окна.

Когда окно меньше полной ширины экрана, часто встречаются слишком длинные строки текста, которые не помещаются в окне. Продолжение всех этих строк может привести к путанице. Переменная `truncate-partial-width-windows` может быть установлена не равной `nil`, это принудительно делает во всех окнах усечение строк до меньшего, чем полная ширина окна, размера независимо от отображаемого буфера и значения переменной `truncate-lines` в нем. См. Раздел 4.8 [Строки продолжения], с. 40.

Горизонтальная прокрутка часто используется в окнах, расположенных рядом вертикально. См. Глава 11 [Изображение], с. 81.

Если `split-window-keep-point` не равна `nil`, как по умолчанию, то оба окна, получающиеся после `C-x 2`, наследуют значение точки от разделяемого окна. Это означает, что необходима прокрутка. Если эта переменная равна `nil`, то `C-x 2` пытается избежать сдвига текста на экране, помещая точку в каждом окне в уже видимом месте. Она также выбирает то окно, которое содержит строку текста, на которой курсор был раньше. Некоторые пользователи предпочитают такой режим на медленных терминалах.

## 16.3 Использование других окон

`C-x o` Выбрать другое окно (`other-window`). Это буква `o`, не ноль.

`C-M-v` Прокрутить следующее окно (`scroll-other-window`).

`M-x compare-windows`
Найти следующее место, где текст выбранного окна не совпадает с текстом в следующем окне.

`Mouse-1` `Mouse-1` на строке режима какого-нибудь окна выбирает это окно, но не перемещает в нем точку (`mouse-select-window`).

Чтобы выбрать другое окно, щелкните `Mouse-1` на его строке режима. С помощью клавиатуры вы можете переключать окна, набирая `C-x o` (`other-window`). Это буква `o`, от

слова ‘other’ (‘другое’), а не ноль. Когда имеется более двух окон, эта команда продвигается через все окна по кругу, обычно сверху вниз и слева направо. От самого правого нижнего окна она идет обратно в то, которое находится в верхнем левом углу. Числовой аргумент означает движение на несколько шагов по круговому порядку окон. Отрицательный аргумент продвигает по кругу в обратном порядке. Когда минибуфер активен, он является последним окном в кольце; вы можете переключиться из окна минибуфера в одно из других окон и позже переключиться обратно и завершить в минибуфере запрашиваемый аргумент. См. Раздел 5.2 [Редактирование в минибуфере], с. 46.

Обычные команды прокрутки (см. Глава 11 [Изображение], с. 81) относятся только к выбранному окну, но существует одна команда для прокрутки следующего окна. `C-M-v` (`scroll-other-window`) прокручивает окно, которое выбрала бы `C-x o`. Она принимает положительные и отрицательные аргументы так же, как и `C-v`. (В минибуфере, `C-M-v` прокручивает окно, содержащее справку для минибуфера, если оно есть, а не следующее окно в стандартном круговом порядке.)

Команда `M-x compare-windows` позволяет вам сравнить два файла или буфера, видимые в двух окнах, продвигаясь по ним до следующего несовпадения. См. Раздел 14.9 [Сравнение файлов], с. 132, для подробностей.

## 16.4 Изображение в другом окне

`C-x 4` — это префиксный ключ для команд, которые выбирают другое окно (разделяя текущее окно, если оно было единственным) и выбирают буфер этом окне. Разные команды на `C-x 4` дают разные способы нахождения буфера для выбора.

`C-x 4 b` *имя-буфера* ⟨RET⟩
: Выбрать буфер *имя-буфера* в другом окне. При этом запускается `switch-to-buffer-other-window`.

`C-x 4 C-o` *имя-буфера* ⟨RET⟩
: Отобразить буфер *имя-буфера* в другом окне, но не выбирать в нем этот буфер. Это запускает `display-buffer`.

`C-x 4 f` *имя-файла* ⟨RET⟩
: Обратиться к файлу *имя-файла* и выбрать его буфер в другом окне. При этом запускается `find-file-other-window`. См. Раздел 14.2 [Обращение], с. 106.

`C-x 4 d` *каталог* ⟨RET⟩
: Выбрать буфер Dired для *каталога* в другом окне. При этом запускается `dired-other-window`. См. Глава 28 [Dired], с. 291.

`C-x 4 m`
: Начать составление почтового сообщения в другом окне. При этом запускается `mail-other-window`; аналог этой команды, работающий в том же окне — `C-x m` (см. Глава 26 [Посылка почты], с. 267).

`C-x 4 .`
: Найти тег в текущей таблице тегов в другом окне. При этом запускается `find-tag-other-window`, многооконный вариант `M-.` (см. Раздел 22.13 [Теги], с. 224).

`C-x 4 r` *имя-файла* ⟨RET⟩
: Обратиться к файлу *имя-файла* в режиме только для чтения и выбрать его буфер в другом окне. Этот ключ запускает команду `find-file-read-only-other-window`. См. Раздел 14.2 [Обращение], с. 106.

## 16.5 Принудительное изображение в том же окне

Некоторые команды Emacs переключают в определенный буфер с особым содержимым. Например, `M-x shell` переключает в буфер, называемый ‘`*Shell*`’. По соглашению, все

эти команды написаны так, чтобы выдавать этот буфер в отдельном окне. Но вы можете указать, что некоторые из этих буферов должны появляться в выбранном окне.

Если вы добавляете имя буфера в список `same-window-buffer-names`, то в результате такие команды отображают этот буфер, переключаясь к нему в том же окне. Например, если вы добавите в этот список элемент `"*grep*"`, то команда `grep` будет показывать буфер с выводом в текущем окне.

Значение `same-window-buffer-names` по умолчанию не равно `nil`: оно задает имена буферов '`*info*`', '`*mail*`' и '`*shell*`' (а также других буферов, используемых более туманными пакетами Emacs). Поэтому `M-x shell` обычно переключает в буфер '`*shell*`' в выбранном окне. Если вы удалите этот элемент из списка `same-window-buffer-names`, поведение `M-x shell` изменится — вместо этого она будет выдавать буфер в другом окне.

Вы можете задать эти буферы в более общей форме с помощью переменной `same-window-regexps`. Установите ее равной списку регулярных выражений; теперь любой буфер, чье имя совпадает с одним из этих регулярных выражений, будет отображаться с переключением к нему в выбранном окне. (Еще раз, это относится только к тем буферам, которые обычно показываются вам в отдельном окне.) Значение этой переменной по умолчанию задает буферы Telnet и rlogin.

Аналогичная возможность позволяет вам задать буферы, которые должны отображаться в принадлежащих только им фреймах. См. Раздел 17.9 [Фреймы специальных буферов], с. 152.

## 16.6 Удаление и переупорядочение окон

`C-x 0` Удалить выбранное окно (`delete-window`). Последний знак в этой последовательности ключей — ноль.

`C-x 1` Удалить в выбранном фрейме все окна, кроме выбранного (`delete-other-windows`).

`C-x 4 0` Удалить выбранное окно и уничтожить буфер, который был в нем показан (`kill-buffer-and-window`). Последний знак в этой последовательности ключей — ноль.

`C-x ^` Сделать выбранное окно выше (`enlarge-window`).

`C-x }` Расширить выбранное окно (`enlarge-window-horizontally`).

`C-x {` Сузить выбранное окно (`shrink-window-horizontally`).

`C-x -` Уменьшить это окно, если буфер в нем не требует столько строк (`shrink-window-if-larger-than-buffer`).

`C-x +` Выровнять высоты всех окон (`balance-windows`).

`Drag-Mouse-1`
Перемещение строки режима окна с помощью `mouse-1` изменяет высоту окон.

`Mouse-2` `Mouse-2` на строке режима окна удаляет все остальные окна в этом фрейме (`mouse-delete-other-windows`).

`Mouse-3` `Mouse-3` на строке режима какого-нибудь окна удаляет это окно (`mouse-delete-window`).

Чтобы удалить окно, наберите `C-x 0` (`delete-window`). (Это ноль). Пространство, занимаемое удаленным окном, достается соседнему окну (но не окну минибуфера, даже если оно активно в этот момент). Как только окно удалено, его атрибуты забываются; их может вернуть только восстановление конфигурации окна. Удаление окна не влияет

на буфер, который оно отображало; этот буфер продолжает существовать, и вы можете выбрать его в любом окне командой `C-x b`.

`C-x 4 0` (`kill-buffer-and-window`) — более сильная команда, чем `C-x 0`; она уничтожает текущий буфер и затем удаляет выбранное окно.

`C-x 1` (`delete-other-windows`) — более мощная в другом смысле; она удаляет все окна за исключением выбранного (и минибуфера). Выбранное окно расширяется, чтобы использовать весь фрейм, за исключением эхо-области.

Вы также можете удалить окно, щелкнув на его строке режима `Mouse-2`, и удалить все окна во фрейме, кроме одного, щелкнув на строке режима этого окна `Mouse-3`.

Простейший способ настроить высоту окон — сделать это с помощью мыши. Если вы нажмете `Mouse-1` на строке режима, вы можете перетащить ее вверх или вниз, изменяя высоту окон сверху и снизу от нее.

Чтобы перенастроить деление пространства между вертикально соседними окнами, используйте `C-x ^` (`enlarge-window`). Она делает текущее выбранное окно больше на одну строку или на столько строк, сколько определено числовым аргументом. С отрицательным аргументом, она делает выбранное окно меньше. `C-x }` (`enlarge-window-horizontally`) делает выбранное окно шире, а `C-x {` (`shrink-window-horizontally`) — уже на заданное число столбцов.

Когда вы увеличиваете окно, оно забирает пространство от одного из своих соседей. Если это делает какие-то окна слишком маленькими, то эти окна удаляются и их пространство делится между соседними окнами. Минимальный размер определяется переменными `window-min-height` и `window-min-width`.

Команда `C-x -` (`shrink-window-if-larger-than-buffer`) уменьшает высоту выбранного окна, если оно выше, чем необходимо для показа всего текста отображаемого в нем буфера. Она передает освободившиеся строки другим окнам в этом фрейме.

Вы также можете использовать `C-x +` (`balance-windows`), чтобы выровнять высоты всех окон в выбранном фрейме.

См. Раздел 5.2 [Редактирование в минибуфере], с. 46, подробное описание режима Resize-Minibuffer, который автоматически изменяет размер окна минибуфера, чтобы в нем вмещался текст минибуфера.

# 17 Фреймы и X Windows

При использовании X Window System вы можете создавать в одном сеансе Emacs несколько окон уровня оконной системы. Каждое принадлежащее Emacs X-окно показывает один *фрейм*, который может содержать одно или несколько окон Emacs. Изначально фрейм содержит одно окно Emacs общего назначения, которое вы можете поделить вертикально или горизонтально на меньшие. Обычно фрейм имеет собственную эхо-область и минибуфер, но вы можете сделать фреймы без них — они будут использовать эхо-область и минибуфер другого фрейма.

Редактирование в одном фрейме затрагивает и другие фреймы. К примеру, если вы поместите текст в список уничтожений в одном фрейме, вы можете восстановить его в другом. Если вы выходите из Emacs через `C-x C-c` в одном фрейме, это удалит все фреймы. Чтобы удалить только один фрейм, используйте `C-x 5 0`.

Чтобы избежать путаницы, мы резервируем слово “окно” для тех окон, которые реализует сам Emacs, и никогда не используем его для обозначения фрейма.

Emacs, скомпилированный для MS-DOS, эмулирует некоторые аспекты оконной системы, так что вы сможете использовать многие возможности из описанных в этой главе. См. Раздел C.1 [MS-DOS Ввод], с. 403, для дальнейшей информации.

## 17.1 Команды мыши для редактирования

Команды мыши для выделения и копирования области в основном совместимы с командами программы `xterm`. Вы можете пользоваться одними и теми же командами мыши для обмена текстом между Emacs другими X-клиентами.

Если вы выделите область с помощью любой из этих команд и сразу же после этого нажмете функциональную клавишу ⟨DELETE⟩, она удалит выделенную область. Функциональная клавиша ⟨BACKSPACE⟩ или ASCII-знак ⟨DEL⟩ не делают этого; и если вы нажмете между командой мыши и ⟨DELETE⟩ какую-то другую клавишу, ⟨DELETE⟩ не подействует таким образом.

`Mouse-1` Перемещает точку туда, где вы щелкнули (`mouse-set-point`). Обычно это левая кнопка.

`Drag-Mouse-1`
: Устанавливает область вокруг текста, по которому вы провели, и копирует этот текст в список уничтожений (`mouse-set-region`). С помощью этой команды вы можете указать оба конца области.

  Если при проведении мышью вы переместите ее за верхний или нижний предел окна, это окно непрерывно прокручивается, пока вы не вернете в него мышь. Таким способом вы можете выделять области, не умещающиеся на экране. Число прокручиваемых за один шаг строк зависит от того, насколько далеко за край окна ушла мышь; минимальный размер шага определяет переменная `mouse-scroll-min-lines`.

`Mouse-2` Восстанавливает последний уничтоженный текст в том месте, где вы щелкнули (`mouse-yank-at-click`). Обычно это средняя кнопка.

`Mouse-3` Эта команда, `mouse-save-then-kill`, имеет несколько назначений в зависимости от того, где вы щелкнули, и от состояния области.

  Самый основной случай — это когда вы щелкаете `Mouse-1` в одном месте, а затем `Mouse-3` в другом. Это выделяет текст между двумя этими позициями в качестве области. Это также копирует новую область в список уничтожений, чтобы вы могли скопировать его в другое место.

Если вы щелкните в тексте `Mouse-1`, прокрутите окно с помощью полоски прокрутки и затем щелкните `Mouse-3`, Emacs запомнит, где была точка перед прокруткой (где вы поместили ее с помощью `Mouse-1`), и использует эту позицию как другой конец области. Это сделано, чтобы вы могли выделять области, которые не умещаются полностью на экране.

В более общем виде, если у вас нет подсвеченной области, `Mouse-3` выделяет в качестве области текст между точкой и местом щелчка. Она делает это, устанавливая метку там, где была точка, и перемещая точку к той позиции, где вы щелкнули.

Если у вас есть подсвеченная область, или если область была установлена непосредственно перед этим с помощью проведения кнопкой 1, `Mouse-3` подстраивает ближайший конец области, перемещая его к месту щелчка. Также, текст подстроенной области замещает в списке уничтожений текст старой области.

Если вы изначально задали область, используя двойной или тройной щелчок `Mouse-1`, чтобы определить область как состоящую из целых слов или строк, то подстройка области с помощью `Mouse-3` также проходит по целым словам или строкам.

Если вы примените `Mouse-3` два раз подряд на одном месте, вы уничтожите уже выделенную область.

`Double-Mouse-1`
: Этот ключ устанавливает область вокруг слова, на котором вы щелкнули. Если вы щелкнули на знаке с синтаксической категорией “symbol” (например, на подчерке в режиме C), он устанавливает область вокруг символа, которому принадлежит этот знак.

  Если вы щелкнули на знаке с синтаксической категорией открывающей или закрывающей круглой скобки, область устанавливается вокруг группы (s-выражения), которая завершается или начинается на этом знаке. Если вы щелкнули на знаке с синтаксической категорией разделителя строк (таком как кавычка или двойные кавычки в Си), область будет установлена вокруг этой строковой константы (с использованием эвристики, чтобы выяснить, является ли этот знак начинающим или завершающим).

`Double-Drag-Mouse-1`
: Этот ключ выделяет область, состоящую из слов, по которым вы провели.

`Triple-Mouse-1`
: Этот ключ устанавливает область вокруг строки, на которой вы щелкнули.

`Triple-Drag-Mouse-1`
: Этот ключ выделяет область, состоящую из строк, по которым вы провели.

Простейший способ уничтожить текст с помощью мыши — нажать `Mouse-1` в одном конце, а затем дважды нажать `Mouse-3` на другом. См. Раздел 9.1 [Уничтожение], с. 69. Чтобы скопировать текст в список уничтожений, не удаляя его из буфера, нажмите `Mouse-3` только один раз или просто проведите по этому тексту с прижатой `Mouse-1`. Потом вы можете скопировать этот текст в другое место восстановлением.

Чтобы восстановить уничтоженный или скопированный текст в другое место, переведите туда мышь и нажмите `Mouse-2`. См. Раздел 9.2 [Восстановление], с. 71. Однако, если `mouse-yank-at-point` не равна `nil`, `Mouse-2` восстанавливает в точке. Тогда не имеет значения, где вы щелкаете, или даже в каком из окон фрейма вы щелкаете. Значение по умолчанию равно `nil`. Эта переменная влияет также на восстановление вторичного выделения.

Чтобы скопировать текст в другое X-окно, уничтожьте его или сохраните в списке уничтожений. Под X это кроме того установит *первичное выделение*. Затем используй-

те в программе, работающей в другом окне, команду “вставить”, чтобы вставить текст выделения.

Чтобы скопировать текст из другого X-окна, используйте в программе, работающей в другом окне, команды “вырезать” или “копировать”, чтобы выделить нужный вам текст. Затем восстановите его в Emacs с помощью `C-y` или `Mouse-2`.

Эти команды вырезания и вставки работают также и в MS-Windows.

Когда Emacs помещает текст в список уничтожений или перекладывает текст на вершину списка уничтожений, он устанавливает *первичное выделение* X-сервера. Именно таким образом другие X-клиенты могут получить доступ к этому тексту. Emacs также сохраняет текст в буфере вырезок, но только если этот текст достаточно короткий (`x-cut-buffer-max` задает максимальное число знаков); помещение в буфер вырезок длинных строк может быть медленным.

Команды восстановления первого вхождения списка уничтожений на самом деле сначала проверяют, нет ли первичного выделения из других программ; после этого они проверяют, нет ли текста в буфере вырезок. Если ни тот, ни другой источник не предоставляют текста для восстановления, используется содержимое списка уничтожений.

## 17.2 Вторичное выделение

*Вторичное выделение* — это другой способ выделения текста с использованием X Windows. Оно не использует точку или метку, поэтому вы можете использовать его для уничтожения текста без установки точки или метки.

`M-Drag-Mouse-1`
: Устанавливает вторичное выделение с одним концом в том месте, где вы нажали кнопку, и другим — в том, где вы ее отпустили (`mouse-set-secondary`). Когда вы проводите мышью, появляется и изменяется подсветка.

  Если при проведении мышью вы сдвинете ее за верхний или нижний предел окна, это окно непрерывно прокручивается, пока вы не вернете в него мышь. Таким способом вы можете выделять области, не умещающиеся на экране.

`M-Mouse-1`
: Устанавливает одну из граничных точек *вторичного выделения* (`mouse-start-secondary`).

`M-Mouse-3`
: Создает вторичное выделение, используя место, указанное с помощью `M-Mouse-1` как его второй конец (`mouse-secondary-save-then-kill`). Второй щелчок на этом же месте уничтожает только что сделанное вторичное выделение.

`M-Mouse-2`
: Вставляет в месте щелчка вторичное выделение (`mouse-yank-secondary`). Это помещает точку в конец восстановленного текста.

Двойные и тройные щелчки `M-Mouse-1` действуют на слова и строки, во многом как для `Mouse-1`.

Если `mouse-yank-at-point` не равна `nil`, `M-Mouse-2` восстанавливает в точке. Тогда не имеет значения, где вы щелкнули; главное, в каком окне. См. Раздел 17.1 [Команды мыши], с. 147.

## 17.3 Следование по ссылкам с помощью мыши

Некоторые буферы Emacs показывают различного рода перечни. Это перечни файлов, буферов, возможных завершений, совпадений с образцом и так далее.

Поскольку восстановление в этих буферах бессмысленно, в большинстве из них `Mouse-2` определена особо, как команда для использования или просмотра пункта, на котором вы щелкнули.

Например, если вы щелкните `Mouse-2` на имени файла в буфере Dired, вы обратитесь к этому файлу. Если вы щелкните `Mouse-2` на сообщении об ошибке в буфере '`*Compilation*`', вы перейдете к исходному коду для этого сообщения. Если вы щелкните `Mouse-2` на завершении в буфере '`*Completions*`', вы выберете это завершение.

Обычно вы можете судить о том, имеет ли `Mouse-2` особое значение, по тому, что чувствительный текст подсвечивается, когда вы проводите над ним мышь.

## 17.4 Щелчки мыши для меню

Щелчки мыши, модифицированные с помощью клавиш ⟨CTRL⟩ и ⟨SHIFT⟩, выводят меню.

`C-Mouse-1`
: Это меню для выбора буфера.

`C-Mouse-2`
: Это меню для задания начертаний и других свойств текста для редактирования форматированного текста. См. Раздел 21.11 [Форматированный текст], с. 198.

`C-Mouse-3`
: Это меню определяется режимом. Для большинства режимов данное меню имеет те же пункты, что содержатся во всех определяемых режимом меню из полоски меню. Некоторые режимы могут определять для этой кнопки другое меню.[1]

`S-mouse-1`
: Это меню для задания основного шрифта фрейма.

## 17.5 Команды мыши для строки режима

Вы можете использовать мышь на строке режима для выбора окон и манипуляций с ними.

`Mouse-1` `Mouse-1` на строке режима выбирает окно сверху. Проводя мышью с нажатой на строке режима `Mouse-1`, вы можете перемещать эту строку режима, изменяя таким образом высоту окон сверху и снизу.

`Mouse-2` `Mouse-2` на строке режима раскрывает окно на весь фрейм.

`Mouse-3` `Mouse-3` на строке режима удаляет окно сверху.

`C-Mouse-2`
: `C-Mouse-2` на строке режима разбивает окно сверху по вертикали в том месте, где вы щелкнули.

`C-Mouse-2` на полоске прокрутки разбивает соответствующее окно по горизонтали. См. Раздел 16.2 [Разделение окон], с. 142.

[1] В некоторых системах для определяемых режимом меню используется `Mouse-3`. Мы провели опрос среди пользователей и выяснили, что они предпочитают оставить `Mouse-3` для выделения и уничтожения областей. Отсюда решение использовать для этого меню `C-Mouse-3`.

## 17.6 Создание фреймов

Префиксный ключ `C-x 5` аналогичен `C-x 4`, и на них определены параллельные подкоманды. Разница между ними в том, что команды с `C-x 5` создают новый фрейм, а не просто новое окно в выбранном фрейме (см. Раздел 16.4 [Всплывающие окна], с. 143). Если запрошенный материал уже показывается в существующем видимом или минимизированном фрейме, эти команды используют существующий фрейм после его поднимания или деминимизации по необходимости.

Команды на `C-x 5` различаются по тому, как они находят или создают буфер для выбора:

`C-x 5 2` Создает новый фрейм (`make-frame-command`).

`C-x 5 b` *имя-буфера* ⟨RET⟩
: Выбирает буфер *имя-буфера* в другом фрейме. Это запускает `switch-to-buffer-other-frame`.

`C-x 5 f` *имя-файла* ⟨RET⟩
: Обращается к файлу *имя-файла* и выбирает его буфер в другом фрейме. Это запускает `find-file-other-frame`. См. Раздел 14.2 [Обращение], с. 106.

`C-x 5 d` *каталог* ⟨RET⟩
: Выбирает буфер Dired для каталога *каталог* в другом фрейме. Это запускает `dired-other-frame`. См. Глава 28 [Dired], с. 291.

`C-x 5 m` Позволяет начать составление почтового сообщения в другом фрейме (`mail-other-frame`). Это вариант `C-x m`, работающий в другом фрейме. См. Глава 26 [Посылка почты], с. 267.

`C-x 5 .` Обращается к тегу из текущей таблицы тегов в другом фрейме. Это запускает `find-tag-other-frame`, вариант `M-.`, работающий с несколькими фреймами. См. Раздел 22.13 [Теги], с. 224.

`C-x 5 r` *имя-файла* ⟨RET⟩
: Обращается к файлу *имя-файла* в режиме только для чтения и выбирает его буфер в другом фрейме. Это запускает `find-file-read-only-other-frame`. См. Раздел 14.2 [Обращение], с. 106.

Вы можете управлять видом вновь создаваемых фреймов, устанавливая параметры фрейма в `default-frame-alist`. Для задания параметров, относящихся только к начальному фрейму, вы можете использовать переменную `initial-frame-alist`. См. раздел “Initial Parameters” в *The Emacs Lisp Reference Manual*, для дальнейшей информации.

Простейший способ указать основной шрифт для всех фреймов Emacs — через X-ресурс (см. Раздел A.7 [Шрифт X], с. 392), но вы также можете сделать это, изменив `default-frame-alist` так, чтобы в нем задавался параметр `font`, как показано здесь:

```
(add-to-list 'default-frame-alist '(font . "10x20"))
```

## 17.7 Создание и использование фрейма Speedbar

Фрейм Emacs может иметь *speedbar*, то есть вертикальное окно, служащее как меню с возможностью прокрутки для файлов, к которым вы можете обратиться, и тегов внутри этих файлов. Чтобы создать speedbar, введите `M-x speedbar`; это создает окно speedbar для выбранного фрейма. После этого вы можете щелкнуть в speedbar на имени файла, чтобы обратиться к нему в соответствующем фрейме Emacs, или щелкнуть на имени тега, чтобы перескочить к нему соответствующем фрейме.

Изначально speedbar перечисляет непосредственное содержание текущего каталога по одному файлу на строке. В каждой строке также есть кнопка, ‘`[+]`’ или ‘`<+>`’, на которой

вы можете щелкнуть `Mouse-2`, чтобы “открыть” содержимое данного элемента. Если на этой строке перечислен каталог, то при открытии к показанному в speedbar добавляется содержание этого каталога под его собственной строкой. Если на строке перечислен обычный файл, при его открытии к содержимому speedbar добавляется список тегов из этого файла. Когда файл открывается, ‘`[+]`’ заменяется на ‘`[-]`’; вы можете щелкнуть на этой кнопке, чтобы “закрыть” этот файл (скрыть его содержание).

Некоторые основные режимы, включая режим Rmail, Info и GUD, имеют специализированные способы поместить в speedbar другие полезные вещи. Например, в режиме Rmail speedbar показывает перечень Rmail-файлов и позволяет вам перенести текущее сообщение в другой Rmail-файл, щелкнув на кнопке ‘`<M>`’.

Speedbar принадлежит одному фрейму Emacs и всегда работает с этим фреймом. Если вы используете несколько фреймов, вы можете сделать speedbar для некоторых или для всех из них; чтобы сделать speedbar для любого данного фрейма, наберите в нем `M-x speedbar`.

## 17.8 Множество дисплеев

Один Emacs может общаться более чем с одним дисплеем X Windows. Изначально Emacs использует только один дисплей — тот, что указан в переменной среды `DISPLAY` или с помощью ключа ‘`-display`’ (см. Раздел A.2 [Ключи запуска], с. 386). Чтобы подсоединиться к другому дисплею, воспользуйтесь командой `make-frame-on-display`:

`M-x make-frame-on-display` ⟨RET⟩ *дисплей* ⟨RET⟩
: Создает новый фрейм на дисплее *дисплей*.

Один X-сервер может обрабатывать более одного экрана. Когда вы открываете фреймы на экранах, принадлежащих одному серверу, Emacs знает, что они разделяют одну клавиатуру и воспринимает все команды, приходящие с этих экранов, как один поток ввода.

Когда вы открываете фреймы на разных X-серверах, Emacs создает для каждого сервера отдельный поток ввода. Поэтому два пользователя могут одновременно печатать на двух дисплеях, и Emacs не смешивает их ввод. Каждый сервер имеет также собственный выбранный фрейм. Команды, которые вы вводите на конкретном X-сервере, относятся к выбранному фрейму этого сервера.

Несмотря на это, люди, использующие одно и то же задание Emacs с разных дисплеев, все же могут вмешиваться в дела друг друга, если будут неосторожны. Например, если любой их них напечатает `C-x C-c`, это прекратит задание Emacs для всех!

## 17.9 Фреймы специальных буферов

Вы можете сделать так, чтобы определенные буферы, для которых Emacs обычно создает второе окно, если у вас есть только одно, появлялись в особых собственных фреймах. Чтобы сделать это, установите переменную `special-display-buffer-names` равной списку имен буферов; любой буфер, чье имя содержится в этом списке, автоматически получает специальный фрейм, когда какая-нибудь команда Emacs хочет показать его “в другом окне”.

Например, если вы установите эту переменную таким образом:

```
(setq special-display-buffer-names
      '("*Completions*" "*grep*" "*tex-shell*"))
```

то списки завершений, вывод `grep` и буфер оболочки режима TEX получат свои собственные фреймы. Эти фреймы и окна в них никогда не разбиваются автоматически и не используются для других буферов. Они продолжают показывать буферы, для которых

были созданы, пока вы не измените их вручную. Уничтожение специального буфера автоматически удаляет его фрейм.

В более общем случае вы можете установить `special-display-regexps` равной списку регулярных выражений; тогда буфер получает собственный фрейм, если его имя совпадает в любым из них. (Еще раз, это относится только к буферам, которые обычно отображаются в отдельном окне.)

Переменная `special-display-frame-alist` задает параметры для этих фреймов. У нее есть значение по умолчанию, поэтому вам необязательно устанавливать ее.

Те, кто знает Лисп, могут также сделать элемент `special-display-buffer-names` или `special-display-regexps` списком. Тогда первый элемент — это имя буфера или регулярное выражение, а остальные указывают, как создавать фрейм. Это может быть ассоциативный список, задающий значения параметров фрейма; эти значения имеют приоритет перед значениями параметров, указанных в `special-display-frame-alist`. Или же остальные элементы могут иметь такую форму:

```
(функция аргументы...)
```

где *функция* — это символ. Тогда фрейм конструируется путем вызова *функции*; ее первым аргументом является буфер, а остальными аргументами — *аргументы*.

Аналогичными средствами вы можете указать буферы, которые должны отображаться в выбранном окне. См. Раздел 16.5 [Использование того же окна], с. 143. Показ в том же окне обладает приоритетом перед показом в специальном фрейме; следовательно, если вы добавили имя буфера к `special-display-buffer-names`, но это не возымело никакого эффекта, проверьте, не используется ли для этого же имени буфера средство показа в том же окне.

## 17.10 Установка параметров фрейма

Этот раздел описывает команды для изменения стиля отображения выбранного фрейма и его поведения в оконной среде.

`M-x set-foreground-color` ⟨RET⟩ *цвет* ⟨RET⟩
: Задает *цвет* для текста в выбранном фрейме.

`M-x set-background-color` ⟨RET⟩ *цвет* ⟨RET⟩
: Задает *цвет* для фона в выбранном фрейме. Также изменяет цвет текста в начертании `modeline`, чтобы оно оставалось инверсией начертания по умолчанию.

`M-x set-cursor-color` ⟨RET⟩ *цвет* ⟨RET⟩
: Задает *цвет* курсора в выбранном фрейме.

`M-x set-mouse-color` ⟨RET⟩ *цвет* ⟨RET⟩
: Задает *цвет* указателя мыши, когда он находится над выбранным фреймом.

`M-x set-border-color` ⟨RET⟩ *цвет* ⟨RET⟩
: Задает *цвет* рамки выбранного фрейма.

`M-x list-colors-display`
: Выводит определенные имена цветов и показывает, как эти цвета выглядят. Это несколько медленная команда.

`M-x auto-raise-mode`
: Переключает автоматическое поднятие выбранного фрейма. Автоматическое поднятие означает, что всякий раз, когда вы передвигаете мышь в этот фрейм, он поднимается.

Обратите внимание, это средство автоматического поднятия реализовано самим Emacs. Некоторые программы управления окнами также реализуют автоподнятие. Если вы включите автоподнятие для фреймов Emacs в вашей программе управления X-окнами, это будет работать, но не под контролем Emacs, и следовательно, `auto-raise-mode` не будет играть роли.

`M-x auto-lower-mode`
: Переключает автоматическое опускание выбранного фрейма. Автоматическое опускание означает, что всякий раз, когда вы передвигаете мышь за пределы этого фрейма, он переносится вниз стека X-окон.

  Команда `auto-lower-mode` не влияет на автоопускание, реализованное программой управления X-окнами. Чтобы контролировать это, вы должны использовать соответствующие средства своей программы управления окнами.

`M-x set-frame-font` ⟨RET⟩ *шрифт* ⟨RET⟩
: Определяет *шрифт* как основной шрифт в выбранном фрейме. Основной шрифт используется для всего отображаемого в этом фрейме текста, кроме случаев, когда для какого-то текста с помощью начертания (см. Раздел 17.13 [Начертания], с. 155) определен другой шрифт. См. Раздел A.7 [Шрифт X], с. 392, чтобы узнать о способах перечисления доступных в вашей системе шрифтов.

  Вы также можете установить основной шрифт фрейма через всплывающее меню. Чтобы вызвать это меню, нажмите `S-Mouse-1`.

В версиях Emacs, которые используют X toolkit, функции для установки цвета и шрифтов не влияют на меню, так как меню отображаются со своими собственными классами виджетов. Чтобы сменить вид меню, вы должны использовать X-ресурсы (см. Раздел A.13 [Ресурсы X], с. 396). См. Раздел A.8 [Цвета X], с. 393, сведения о цветах, а также Раздел A.7 [Шрифт X], с. 392, касающийся вопросов о выборе шрифта.

Для получения информации о параметрах и настройках фреймов смотрите раздел “Frame Parameters” в *The Emacs Lisp Reference Manual*.

## 17.11 Полоски прокрутки

При использовании X, Emacs обычно создает *полоски прокрутки* слева от каждого окна. Полоска прокрутки проходит по всей высоте окна, а внутри ее есть прямоугольник, представляющий показанную в данный момент часть буфера. Полная высота полоски прокрутки представляет всю длину буфера.

Вы можете использовать на полоске прокрутки `Mouse-2` (обычно это средняя кнопка), чтобы передвинуть или перетащить внутренний прямоугольник вверх или вниз. Если вы переместите его к верху полоски прокрутки, вы увидите начало буфера. Если вы переместите его к низу полоски прокрутки, то увидите конец буфера.

Правая и левая кнопки мыши прокручивают на контролируемое число строк. `Mouse-1` (обычно левая кнопка) перемещает строку, на уровне которой вы щелкнули, к верхнему краю окна. `Mouse-3` (обычно правая кнопка) перемещает верхнюю строку окна вниз к тому уровню, где вы щелкнули. Щелкая последовательно на одном месте, вы можете прокручивать на одно и то же расстояние еще и еще.

Помимо прокрутки, вы также можете щелкнуть на полоске прокрутки с помощью `C-Mouse-2`, чтобы разбить окно по горизонтали. Разбиение делается в той строке, где вы щелкнули.

Вы можете включить или выключить режим Scroll Bar командой `M-x scroll-bar-mode`. Без аргумента, она переключает использование полосок прокрутки. С аргументом, она включает использование полосок прокрутки тогда и только тогда, когда аргумент положителен. Эта команда применяется ко всем фреймам, включая те, что еще будут созданы.

Для управления начальной установкой режима Scroll Bar вы можете использовать X-ресурс ‘`verticalScrollBars`’. См. Раздел A.13 [Ресурсы X], с. 396.

Чтобы включить или выключить полоски прокрутки только для выбранного фрейма, воспользуйтесь командой `M-x toggle-scroll-bar`.

## 17.12 Полоски меню

Вы можете включить или выключить показ полосок меню с помощью команды `M-x menu-bar-mode`. Без аргументов, эта команда переключает режим Menu Bar; это второстепенный режим. С аргументом, эта команда включает режим Menu Bar, если аргумент положителен, и выключает, если аргумент неположителен. Для управления начальной установкой режима Menu Bar вы можете использовать X-ресурс ‘`menuBarLines`’. См. Раздел A.13 [Ресурсы X], с. 396. Профессиональные пользователи часто выключают полоску меню, особенно на текстовых терминалах, где это освобождает одну дополнительную строку для текста.

См. Раздел 1.4 [Полоска меню], с. 26, для получения информации о том, как вызывать команды с помощью полоски меню.

## 17.13 Использование разных начертаний

Когда вы используете Emacs с системой X, вы можете настроить несколько стилей отображения знаков. Вы можете контролировать такие аспекты стиля: шрифт, цвет текста, цвет фона и подчеркивание. Emacs частично поддерживает начертания в MS-DOS, позволяя вам контролировать для каждого начертания цвета текста и фона (см. Приложение C [MS-DOS], с. 403).

Вы управляете стилем отображения путем определения именованных *начертаний*. Каждое начертание может задавать шрифт, цвет текста, цвет фона и флаг подчеркивания; но оно не обязано задавать их все. Затем, указывая начертание или начертания для заданной части текста в буфере, вы управляете внешним видом этого текста.

Используемый для каждого данного знака стиль отображения определяется комбинацией нескольких начертаний. Любой аспект стиля, не заданный перекрытиями или свойствами текста, предоставляется самим фреймом.

Режим Enriched, предназначенный для редактирования форматированного текста, включает несколько команд и меню для задания начертаний. См. Раздел 21.11.4 [Начертания в форматированном тексте], с. 200, чтобы узнать, как указать шрифт для текста в буфере. См. Раздел 21.11.5 [Цвета в форматированном тексте], с. 201, о задании цветов текста и фона.

Для изменения внешнего вида начернания используйте буфер настройки. См. Раздел 31.2.2.3 [Настройка начертаний], с. 347. Для определения атрибутов конкретных начертаний вы также можете использовать X-ресурсы (см. Раздел A.13 [Ресурсы X], с. 396).

Чтобы узнать, какие начертания определены в данный момент, и как они выглядят, наберите `M-x list-faces-display`. Одно начертание может выглядеть по-разному в разных фреймах; данная команда показывает его вид в том фрейме, где вы ее набрали. Вот перечень стандартных определенных начертаний:

`default` Это начертание используется для обычного текста, для которого не определено другое начертание.

`modeline` Это начертание используется для строк режима. По умолчанию оно устанавливается как инверсия начертания `default`. См. Раздел 11.7 [Переменные изображения], с. 84.

`highlight`
: Это начертание используется в различных режимах для подсветки частей текста.

`region`
: Это начертание применяется для отображения выделенной области (когда включен режим Transient Mark — смотрите ниже).

`secondary-selection`
: Это начертание используется для отображения вторичного выделения (см. Раздел 17.2 [Вторичное выделение], с. 149).

`bold`
: Это начертание использует жирный вариант шрифта по умолчанию, если он есть.

`italic`
: Это начертание использует курсивный вариант шрифта по умолчанию, если он есть.

`bold-italic`
: Это начертание использует жирный курсивный вариант шрифта по умолчанию, если он есть.

`underline`
: Это начертание подчеркивает текст.

Если включен режим Transient Mark, текст области подсвечивается, когда метка активна. Для этого используется начертание с именем `region`; вы можете управлять стилем подсветки, меняя стиль этого начертания (см. Раздел 31.2.2.3 [Настройка начертаний], с. 347). См. Раздел 8.2 [Transient Mark], с. 64, для получения большей информации о режиме Transient Mark и активизации и деактивизации метки.

Один простой способ использовать разные начертания — включить режим Font Lock. Этот второстепенный режим, всегда локальный для конкретного буфера, подбирает начертания в соответствии с синтаксисом редактируемого текста. Он может распознавать комментарии и строки в большинстве языков; в некоторых языках он умеет даже распознавать и правильно подсвечивать различные другие важные конструкции. См. Раздел 17.14 [Font Lock], с. 156, для получения большей информации о режиме Font Lock и синтаксической подсветке.

Вы можете распечатать буфер с подсветкой, какую вы видите на экране, с помощью команды `ps-print-buffer-with-faces`. См. Раздел 30.5 [Postscript], с. 332.

## 17.14 Режим Font Lock

Режим Font Lock — это второстепенный режим, всегда локальный для каждого буфера, который подсвечивает редактируемый вами текст, используя различные начертания в соответствии с синтаксисом текста. Он может распознавать комментарии и строки в большинстве языков; в некоторых языках он может также распознавать и правильно подсвечивать различные другие важные конструкции — например, имена определяемых функций и зарезервированные ключевые слова.

Команда `M-x font-lock-mode` включает и выключает режим Font Lock в соответствии с аргументом и переключает, если аргумент не задан. Функция `turn-on-font-lock` безусловно включает режим Font Lock. Это полезно в функциях-ловушках режима. Например, чтобы задействовать режим Font Lock всякий раз, когда вы редактируете файл на Си, вы можете сделать так:

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

Чтобы включить режим Font Lock автоматически во всех режимах, которые его поддерживают, используйте функцию `global-font-lock-mode`, как показано здесь:

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

Когда вы редактируете текст в режиме Font Lock, подсветка в измененной строке обновляется автоматически. Чаще всего изменения не оказывают влияния на подсветку последующих строк, но иногда все же влияют. Чтобы обновить подсветку нескольких строк, используйте команду `M-g M-g` (`font-lock-fontify-block`).

В некоторых основных режимах `M-g M-g` обновляет подсветку всей текущей функции. (Как именно можно найти текущую функцию, указывает переменная `font-lock-mark-block-function`.) В других основных режимах `M-g M-g` обновляет подсветку 16-ти строк над и под точкой.

С числовым аргументом *n*, `M-g M-g` обновляет подсветку *n* строк над и под точкой, независимо от режима.

Чтобы получить все преимущества режима Font Lock, вам придется выбрать такой шрифт по умолчанию, у которого есть жирный, курсивный и жирный курсивный варианты; или вам понадобится цветной монитор или монитор, отображающий много градаций серого.

Переменная `font-lock-maximum-decoration` задает предпочтительный уровень оформления для режимов, которые предоставляют несколько уровней. Уровень 1 — это минимальное оформление; некоторые режимы поддерживают до трех уровней. Обычное значение по умолчанию обозначает “как можно больше”. Вы можете указать целое число, которое применяется ко всем режимам, или задать разные числа для конкретных режимов; например, чтобы использовать уровень 1 для режимов C/C++ и уровень по умолчанию в остальных случаях, напишите так:

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

В больших буферах подсветка может быть слишком медленной, поэтому вы можете подавить ее. Переменная `font-lock-maximum-size` задает размер буфера, сверх которого подсветка не делается.

Подсветка комментариев и строк (или “синтаксическая” подсветка) основывается на анализе синтаксической структуры текста буфера. В целях увеличения скорости некоторые режимы, включая режим C и режим Lisp полагаются на особое соглашение: открывающая скобка в самом левом столбце всегда обозначает начало определения функции и, таким образом, всегда находится вне любой строки или комментария. (См. Раздел 22.4 [Определения функций], с. 208.) Если вы не следуете этому соглашению, режим Font Lock может сделать неправильную подсветку текста после открывающей скобки в левом столбце, которая попадает в строку или комментарий.

Переменная `font-lock-beginning-of-syntax-function` (всегда локальная для буфера) указывает, каким образом режим Font Lock может найти позицию, которая гарантированно находится вне любого комментария или строки. В режимах, использующих соглашение об открывающей скобке в левом столбце, значение этой переменной по умолчанию — это `beginning-of-defun`, что велит режиму Font Lock применять это соглашение. Если вы установите эту переменную в `nil`, Font Lock больше не станет полагаться на это соглашение. Это позволяет избежать неверных результатов, но ценой этого будет то, что в некоторых случаях для подсветки придется проходить текст буфера с самого начала.

Образцы подсветки Font Lock уже существуют для многих режимов, но вы можете захотеть раскрасить что-то дополнительно. Чтобы добавить свои собственные образцы подсветки для определенного режима, вы можете использовать функцию `font-lock-add-keywords`. К примеру, чтобы выделить в комментариях Си слова ‘`FIXME:`’, используйте это:

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

## 17.15 Режимы поддержки Font Lock

Режимы поддержки убыстряют режим Font Lock в больших буферах. Есть два режима поддержки: режим Fast Lock и режим Lazy Lock. Они используют два разных метода ускорения режима Font Lock.

### 17.15.1 Режим Fast Lock

Чтобы сделать режим Font Lock более быстрым для буферов, обращающихся к большим файлам, вы можете использовать режим Fast Lock. Режим Fast Lock сохраняет информацию о шрифтах для каждого файла в отдельном файле кеша; всякий раз, когда вы обращаетесь к файлу, он заново считывает информацию о шрифтах из файла кеша вместо того, чтобы вычислять шрифты для текста с нуля.

Команда `M-x fast-lock-mode` включает и выключает режим Fast Lock в соответствии с аргументом (без аргумента, режим переключается). Вы также можете сделать так, чтобы режим Fast Lock включался всякий раз, когда вы используете режим Font Lock, следующим образом:

```
(setq font-lock-support-mode 'fast-lock-mode)
```

Записывать файл кеша для маленьких буферов не имеет смысла. Поэтому есть переменная, `fast-lock-minimum-size`, задающая наименьший размер файла, для которого информация о шрифтах кешируется.

Переменная `fast-lock-cache-directories` указывает, где нужно размещать файлы кеша. Ее значение — это список каталогов, которые будут испробованы; `"."` означает тот же каталог, где файл редактируется. Значение по умолчанию равно `("." "~/.emacs-flc")`, что велит использовать тот же каталог, если это возможно, иначе использовать каталог '`~/.emacs-flc`'.

Переменная `fast-lock-save-others` указывает, должен ли режим Fast Lock сохранять файлы кеша для файлов, чьим владельцем являетесь не вы. Отличное от `nil` значение говорит, что должен (и это значение по умолчанию).

### 17.15.2 Режим Lazy Lock

Чтобы ускорить режим Font Lock для больших буферов, вы можете использовать режим Lazy Lock, который уменьшает количество текста, подлежащего подсветке. В режиме Lazy Lock подсветка буфера делается по необходимости; она производится только для тех частей буфера, которые должны появиться на экране. И подсветка ваших изменений замедлена; она производится, только когда Emacs бездействовал определенный небольшой промежуток времени.

Команда `M-x lazy-lock-mode` включает и выключает режим Lazy Lock в соответствии с аргументом (без аргумента, режим переключается). Вы также можете сделать так, чтобы режим Lazy Lock включался всякий раз, когда вы используете режим Font Lock, следующим образом:

```
(setq font-lock-support-mode 'lazy-lock-mode)
```

Избегать подсветки маленьких буферов не имеет смысла. Минимальный размер буфера, для которого подсветка делается по необходимости, определяется переменной `lazy-lock-minimum-size`. Меньшие буферы расцвечиваются сразу, как в простом режиме Font Lock.

Когда вы изменяете буфер, режим Lazy Lock откладывает подсветку измененного текста. Переменная `lazy-lock-defer-time` задает число секунд, которое Emacs должен оставаться незанятым, прежде чем начать подсветку ваших изменений. Если ее значение равно 0, изменения подсвечиваются незамедлительно, как в простом режиме Font Lock.

Обычно режим Lazy Lock подсвечивает ставшие видимыми фрагменты буфера перед тем, как они впервые показываются на экране. Однако, если значение `lazy-lock-defer-on-scrolling` не равно `nil`, вновь видимый текст подсвечивается только после того, как Emacs бездействовал `lazy-lock-defer-time` секунд.

В некоторых режимах, включая режим C и режим Emacs Lisp, изменение содержимого одной строки изменяет контекст последующих строк и, следовательно, ту подсветку, которая должна для них использоваться. Обычно вы должны набрать `M-g M-g`, чтобы обновить подсветку последующих строк. Однако, если вы установите переменную `lazy-lock-defer-contextually` не равной `nil`, режим Lazy Lock делает это автоматически по истечении `lazy-lock-defer-time` секунд.

Когда Emacs бездействует продолжительное время, Lazy Lock подсвечивает дополнительные фрагменты буфера, которые еще не показаны, на случай, если вы вынесете их на экран позднее. Это называется *скрытой подсветкой*.

Переменная `lazy-lock-stealth-time` определяет, сколько минут Emacs должен оставаться незанятым, прежде чем начать скрытую подсветку. Значение `nil` означает отсутствие скрытой подсветки. Переменные `lazy-lock-stealth-lines` и `lazy-lock-stealth-verbose` задают диапазон и подробность скрытой подсветки.

### 17.15.3 Fast Lock или Lazy Lock?

Вот простые указания, которые помогут вам выбрать один из режимов поддержки Font Lock.

- Режим Fast Lock играет роль только при обращении к файлам и уничтожении буферов (и связанных событий); следовательно, редактирование буфера и прокрутка окна не быстрее и не медленнее, чем в простом режиме Font Lock.
- Режим Fast Lock медленнее при считывании файлов кеша, чем режим Lazy Lock при подсветке буфера; следовательно, режим Fast Lock медленнее при обращении к файлам, чем режим Lazy Lock.
- Режим Lazy Lock работает во время прокрутки окна, чтобы раскрасить текст, выносимый на экран; следовательно, прокрутка медленнее, чем в простом режиме Font Lock.
- Режим Lazy Lock не подсвечивает буфер в процессе редактирования (он откладывает подсветку изменений); следовательно, редактирование в нем быстрее, чем в простом режиме Font Lock.
- Режим Fast Lock может быть сбит с толку файлом, находящимся под контролем системы управления версиями; следовательно, подсветка буфера может производиться, даже если для этого файла есть файл кеша.
- Режим Fast Lock работает только с буферами, обращающимися к файлам; режим Lazy Lock работает с любыми буферами.
- Режим Fast Lock создает файлы кеша; режим Lazy Lock не создает.

Переменная `font-lock-support-mode` указывает, какой из этих режимов поддержки следует использовать; например, чтобы сказать, что в режимах C/C++ используется режим Fast Lock, а в остальных случаях — режим Lazy Lock, установите эту переменную так:

```
(setq font-lock-support-mode
      '((c-mode . fast-lock-mode) (c++-mode . fast-lock-mode)
        (t . lazy-lock-mode)))
```

## 17.16 Режим Highlight Changes

Используйте `M-x highlight-changes-mode`, чтобы включить второстепенный режим, который показывает с помощью разных начертаний (в основном цветов), какие части текста буфера изменялись недавно.

## 17.17 Другие возможности X Windows

Следующие команды позволяют вам создавать и удалять фреймы, а также делать другие операции над ними:

`C-z` Минимизирует выбранный фрейм (`iconify-or-deiconify-frame`). Обычное значение `C-z`, приостановка Emacs, бесполезно под оконной системой, поэтому в данном случае у этого ключа другая привязка.

Если вы введете эту команду в пиктограмме фрейма Emacs, она деминимизирует этот фрейм.

`C-x 5 0` Удаляет выбранный фрейм (`delete-frame`). Это не допускается, если есть только один фрейм.

`C-x 5 o` Выбирает другой фрейм, поднимает его и переносит в него мышь, чтобы он оставался выбранным. Если вы повторяете эту команду, она циклически проходит по всем фреймам на вашем терминале.

## 17.18 Неоконные терминалы

Если ваш терминал не обладает оконной системой, которую поддерживает Emacs, то он может показывать только один фрейм Emacs в один момент времени. Однако, вы все же можете создавать несколько фреймов Emacs и переключаться между ними. На таких терминалах переключение фреймов во многом похоже на переключение между различными конфигурациями окон.

Чтобы создать новый фрейм и переключиться в него, используйте `C-x 5 2`; для кругового прохода по существующим фреймам используйте `C-x 5 o`; чтобы удалить текущий фрейм, используйте `C-x 5 0`.

Каждый фрейм имеет отличительный номер. Если ваш терминал может показывать в одно время только один фрейм, то около начала строки режима появляется номер $n$ текущего фрейма в форме '`F`$n$'.

'`F`$n$' на самом деле — это имя фрейма. Вы также можете указать другое имя, если хотите, и вы можете выбирать фреймы по именам. Чтобы задать новое имя для текущего фрейма, используйте команду `M-x set-frame-name` ⟨RET⟩ *имя* ⟨RET⟩, а для выбора фрейма в соответствии с его именем используйте `M-x select-frame-by-name` ⟨RET⟩ *имя* ⟨RET⟩. Указанное вами имя появляется в строке режима, когда этот фрейм становится выбранным.

# 18 Поддержка разных языков

Emacs поддерживает широкий спектр наборов знаков разных языков, включая европейские варианты латинского алфавита, а также китайскую, девангари (хинди и маратхи), эфиопскую, греческую, IPA, японскую, корейскую, лаосскую, русскую, тайскую, тибетскую и вьетнамскую письменности. Эти возможности были внесены из измененной версии Emacs, известной как MULE (от "MULti-lingual Enhancement to GNU Emacs".[1])

## 18.1 Введение в наборы знаков разных языков

Пользователи этих систем письма выработали много более или менее стандартных систем кодирования для хранения файлов. Внутренне Emacs использует единую многобайтную кодировку, так что в ней можно перемешивать знаки из всех этих систем письма в одном буфере или строке. Эта кодировка представляет каждый знак, не входящий в ASCII, как последовательность байт в промежутке от 0200 до 0377. Emacs переводит из этой многобайтной кодировки в различные другие системы кодирования при считывании и записи файлов, при обмене данными с подпроцессами и (в некоторых случаях) в команде `C-q` (см. Раздел 18.6 [Многобайтные преобразования], с. 164).

Команда `C-h h` (`view-hello-file`) выводит файл '`etc/HELLO`', который показывает, как сказать "здравствуйте" на разных языках. Это иллюстрирует различные виды письменности.

Даже в странах, где используются эти знаки, на клавиатурах обычно нет клавиш для всех из них. Поэтому Emacs поддерживает различные *методы ввода*, как правило, один для каждой письменности или языка, чтобы их было удобно вводить.

Префиксный ключ `C-x` ⟨RET⟩ используется для команд, которые имеют отношение к многобайтным знакам, системам кодирования и методам ввода.

## 18.2 Включение поддержки многобайтных знаков

Вы можете включить или выключить поддержку многобайтных знаков либо для всего Emacs, либо для отдельного буфера. Когда в буфере выключены многобайтные знаки, каждый байт в нем представляет один знак, даже коды от 0200 до 0377. Старые средства для поддержки европейских наборов знаков, ISO Latin-1 и ISO Latin-2, работают так же, как они работали в Emacs 19, и кроме того, работают для других наборов знаков ISO 8859.

Однако, чтобы использовать ISO Latin, необязательно выключать поддержку многобайтных знаков; многобайтный набор знаков Emacs включает все эти знаки, и Emacs может автоматически переводить из него в коды ISO и наоборот.

Чтобы отредактировать определенный файл в однобайтном представлении, обратитесь к нему через `find-file-literally`. См. Раздел 14.2 [Обращение], с. 106. Чтобы превратить буфер в многобайтном представлении в однобайтное представление тех же знаков, проще всего сохранить содержимое этого буфера в файле, уничтожить его и снова обратиться к этому файлу с помощью `find-file-literally`. Вы также можете использовать `C-x` ⟨RET⟩ `c` (`universal-coding-system-argument`) и указать '`raw-text`' в качестве системы кодирования для обращения к файлу или для его сохранения. См. Раздел 18.9 [Задание кодирования], с. 168. Обращение к файлу как к '`raw-text`' не выключает преобразование формата, декомпрессию и автоматический выбор режима, в отличие от `find-file-literally`.

[1] Многоязыковое расширение GNU Emacs. *(Прим. переводчика)*

Чтобы выключить поддержку многобайтных знаков по умолчанию, запустите Emacs с ключом ‘`-unibyte`’ (см. Раздел A.2 [Ключи запуска], с. 386) или установите переменную среды `EMACS_UNIBYTE`. Вы также можете настроить параметр `enable-multibyte-characters` или, что эквивалентно, прямо установить переменную `default-enable-multibyte-characters` в вашем файле инициализации, это дает в основном тот же эффект, что и ‘`-unibyte`’.

Во время инициализации не создаются многобайтные строки из значений переменных среды, вхождений файла ‘`/etc/passwd`’, etc., которые содержат не входящие в ASCII восьмибитные знаки. Однако, файл инициализации обычно считывается как многобайтный — как все файлы на Лиспе — даже если задан ключ ‘`-unibyte`’. Чтобы избежать создания многобайтных строк из находящихся в этом файле строк с не-ASCII-знаками, поместите в его первой строке комментарий с текстом ‘`-*-unibyte: t;-*-`’. Для файлов инициализации других пакетов, вроде Gnus, нужно сделать то же самое.

В строке режима показано, включена ли поддержка многобайтных знаков в текущем буфере. Если она включена, перед двоеточием в начале строки режима стоят два или более знака (чаще всего два дефиса). Когда многобайтные знаки не включены, перед двоеточием есть только один дефис.

## 18.3 Языковые среды

Все поддерживаемые наборы знаков допустимы в буферах Emacs, если включены многобайтные знаки; нет необходимости выбирать конкретный язык, чтобы увидеть его знаки в буфере Emacs. Однако, важно выбрать *языковую среду*, чтобы получить различные установки по умолчанию. На самом деле языковая среда представляет выбор предпочтительной письменности (в большей или меньшей степени), а не выбор языка.

Языковая среда определяет, какие системы кодирования распознаются при считывании текста (см. Раздел 18.8 [Распознавание кодирования], с. 166). Это относится к файлам, приходящей почте, сетевым новостям и любому другому тексту, который вы считываете в Emacs. Она также может задавать систему кодирования, используемую по умолчанию для создания файла. Каждая языковая среда также указывает принимаемый по умолчанию метод ввода.

Языковая среда выбирается командой `M-x set-language-environment`. Не имеет значения, какой буфер является текущим во время запуска этой команды, потому что ее действия применяются глобально ко всему сеансу Emacs. Поддерживаемые языковые среды включают:

> 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 и Vietnamese.

Некоторые операционные системы позволяют вам указать используемый вами язык путем установки переменных среды, определяющих вашу местность. Emacs может обрабатывать один распространенный частный случай: если название вашей региональной установки для типов знаков содержит строку ‘`8859-`*n*’, Emacs автоматически выбирает соответствующую языковую среду.

Чтобы получить сведения об эффектах определенной языковой среды *яз-среда*, используйте команду `C-h L` *яз-среда* ⟨RET⟩ (`describe-language-environment`). Это сообщит вам, для каких языков полезна данная языковая среда, и перечислит приходящие с ней наборы знаков, системы кодирования и методы ввода. Эта команда также показывает образцы текста, которые иллюстрируют используемые в этой языковой среде системы письма. По умолчанию она описывает выбранную языковую среду.

Вы можете настроить для себя любую языковую среду с помощью обычной ловушки `set-language-environment-hook`. Команда `set-language-environment` запускает эту ло-

вушку после подготовки новой языковой среды. Функции этой ловушки могут определить текущую языковую среду по значению переменной `current-language-environment`.

До начала подготовки новой языковой среды, команда `set-language-environment` запускает ловушку `exit-language-environment-hook`. Эта ловушка полезна для отмены настроек, сделанных с помощью `set-language-environment-hook`. К примеру, если вы установили с использованием `set-language-environment-hook` особые привязки ключей для конкретной языковой среды, вам стоит сделать так, чтобы `exit-language-environment-hook` восстанавливала нормальные привязки.

## 18.4 Методы ввода

*Метод ввода* — это разновидность преобразования знаков, разработанная специально для интерактивного ввода. В Emacs, как правило, каждый язык имеет свой метод ввода; иногда несколько языков, в которых используются одни и те же знаки, могут разделять один метод ввода. Есть немного языков, которые поддерживают несколько методов ввода.

В простейшем случае метод ввода работает через отображение ASCII-букв в другой алфавит. Таким способом действуют методы ввода для греческого и русского.

Более мощный способ — составление: преобразование последовательности знаков в одну букву. Составление используется во многих европейских методах ввода для сознания одной не-ASCII-буквы из последовательности, состоящей из буквы, за которой идет знак акцента (или наоборот). Например, некоторые методы ввода преобразуют последовательность `a'` в одну букву с акцентом. В этих методах ввода нет собственных специальных команд; всё, что они делают, — компонуют последовательности печатных знаков.

Методы ввода для силлабических систем письма обычно используют последовательно отображение и затем составление. Таким способом работают методы ввода для тайского и корейского. Сначала буквы отображаются в символы отдельных звуков или меток тона; затем такие последовательности, составляющие целый слог, отображаются в один знак слога.

Для китайского и японского требуются более сложные методы. В китайских методах ввода вы сначала вводите фонетическое написание китайского слова (в методе ввода `chinese-py`, помимо прочих) или последовательность частей знака (методы ввода `chinese-4corner`, `chinese-sw` и другие). Поскольку одно фонетическое написание обычно соответствует многим различным китайским знакам, вы должны выбрать одну из альтернатив с помощью особых команд Emacs. Такие ключи, как `C-f`, `C-b`, `C-n`, `C-p`, и цифры имеют в этой ситуации особые определения, используемые для выбора среди альтернатив. ⟨TAB⟩ выводит буфер, показывающий все возможные варианты.

В японских методах ввода вы сначала вводите целое слово, используя фонетическое написание; потом, когда это слово уже в буфере, Emacs преобразует его в один или несколько знаков, используя большой словарь. Одно фонетическое написание соответствует многим по-разному записанным японским словам, поэтому вы должны выбрать один из них; для циклического прохода по альтернативам используйте `C-n` и `C-p`.

Иногда полезно остановить действие метода ввода, чтобы только что введенные вами знаки не сливались с последующими. Например, в методе ввода `latin-1-postfix` последовательность `e '` комбинируется в ‘`e`’ с акцентом. Что если вы хотели ввести их как раздельные знаки?

Один способ — набрать акцент дважды; это специальное средство для ввода буквы и акцента раздельно. Например, `e ' '` дает два знака ‘`e'`’. Другой способ — набрать после ‘`e`’ еще одну букву, которая не скомбинируется с ней, и сразу удалить ее. Например, вы могли бы набрать `e e` ⟨DEL⟩ `'`, чтобы получить раздельные ‘`e`’ и ‘`'`’.

Еще один способ, более общий, но не такой легкий для набора, — использовать между двумя знаками `C-\ C-\`, чтобы предотвратить их комбинирование. Это команда `C-\` (`toggle-input-method`), примененная дважды.

C-\ C-\ особенно полезна в наращиваемом поиске, поскольку она останавливает ожидание дальнейших знаков для составления и начинает поиск того, что вы уже набрали.

Переменные input-method-highlight-flag и input-method-verbose-flag управляют тем, как методы ввода поясняют происходящее. Если input-method-highlight-flag не равна nil, частичная последовательность подсвечивается в буфере. Если input-method-verbose-flag не равна nil, в эхо-области показывается список возможных следующих знаков (но не в том случае, когда вы находитесь в минибуфере).

## 18.5 Выбор метода ввода

C-\ Включает или выключает использование выбранного метода ввода.

C-x ⟨RET⟩ C-\ *метод* ⟨RET⟩
Выбирает новый метод ввода для текущего буфера.

C-h I *метод* ⟨RET⟩
C-h C-\ *метод* ⟨RET⟩
Описывает метод ввода *метод* (describe-input-method). По умолчанию, она описывает текущий метод ввода (если он есть). Такое описание должно давать вам все подробности о том, как использовать любой конкретный метод ввода.

M-x list-input-methods
Выводит перечень всех поддерживаемых методов ввода.

Чтобы выбрать метод ввода для текущего буфера, используйте C-x ⟨RET⟩ C-\ (set-input-method). Эта команда считывает имя метода ввода из минибуфера; имя обычно начинается с языковой среды, для которой этот метод предназначался. В переменной current-input-method записывается, какой метод ввода был выбран.

Методы ввода используют для обозначения знаков, не входящих в ASCII, различные последовательности ASCII-знаков. Иногда бывает полезно временно выключить метод ввода. Чтобы сделать это, наберите C-\ (toggle-input-method). Чтобы опять задействовать метод ввода, наберите C-\ снова.

Если вы напечатаете C-\, но метод ввода пока не выбран, вас попросят указать его. Это имеет тот же эффект, что и использование C-x ⟨RET⟩ C-\ для задания метода ввода.

Выбор языковой среды определяет метод ввода, используемый по умолчанию. Тогда вы можете выбрать его в текущем буфере, набирая C-\. Переменная default-input-method задает метод ввода, принимаемый по умолчанию (nil означает, что такого нет).

Некоторые методы ввода для алфавитных систем письма работают путем отображения клавиатуры для эмуляции различных раскладок, часто используемых для этих систем письма. Как правильно сделать это отображение, зависит от действительной раскладки вашей клавиатуры. Чтобы указать ее, используйте команду M-x quail-set-keyboard-layout.

Чтобы просмотреть перечень всех поддерживаемых методов ввода, наберите M-x list-input-methods. Перечень сообщает сведения о каждом методе ввода, включая строку, обозначающую этот метод ввода в строке режима.

## 18.6 Однобайтные и многобайтные не-ASCII-знаки

Когда включены многобайтные знаки, знаки с кодами от 0240 (восьмиричное) до 0377 (восьмиричное) на самом деле недопустимы в буфере. Допустимые печатные знаки, не входящие в ASCII, имеют коды, начинающиеся от 0400.

Если вы набираете самовставляющийся знак в недопустимом диапазоне от 0240 до 0377, Emacs предполагает, что вы намеревались использовать один из наборов знаков Latin-*n*, и

преобразует его в код Emacs, представляющий этот знак Latin-*n*. Вы указываете, *какой* набор знаков ISO нужно для этого применять, своим выбором языковой среды (смотрите выше). Если вы не указали свой выбор, по умолчанию используется Latin-1.

То же происходит, когда вы используете `C-q` для ввода восьмиричного кода в этом диапазоне.

## 18.7 Системы кодирования

Носители различных языков выработали много более или менее стандартных систем кодирования для их представления. Emacs не использует эти системы кодирования внутренне; вместо этого, при считывании данных он преобразует их из различных систем кодирования в свою внутреннюю, а при записи он преобразует данные из внутренней системы кодирования в другие системы. Преобразование возможно при считывании и записи файлов, отправке или получении данных с терминала и при обмене данными с подпроцессами.

Emacs присваивает каждой системе кодирования свое имя. Большинство систем кодирования используются для одного языка, и имя такой системы кодирования начинается с имени языка. Некоторые системы кодирования используются для нескольких языков; их имена обычно начинаются с '`iso`'. Есть также специальные системы кодирования `no-conversion`, `raw-text` и `emacs-mule`, которые не делают преобразования печатных знаков вообще.

Помимо преобразований между разными представлениями не-ASCII-знаков, система кодирования может производить преобразование последовательности "конец-строки". Emacs работает с тремя различными соглашениями о том, как разделять строки в файле: переводом строки, возвратом каретки и переводом строки и просто возвратом каретки.

`C-h C` *кодирование* ⟨RET⟩
: Описывает систему кодирования *кодирование*.

`C-h C` ⟨RET⟩
: Описывает систему кодирования, используемую в данный момент.

`M-x list-coding-systems`
: Выводит перечень всех поддерживаемых систем кодирования.

Команда `C-h C` (`describe-coding-system`) выводит сведения о конкретной системе кодирования. Вы можете задать имя системы кодирования в качестве аргумента; иначе, с пустым аргументом, она опишет системы кодирования, выбранные в данный момент для различных целей как в текущем буфере, так и принимаемые по умолчанию, а также перечень приоритетов для распознавания систем кодирования (см. Раздел 18.8 [Распознавание кодирования], с. 166).

Чтобы вывести перечень всех поддерживаемых систем кодирования, наберите `M-x list-coding-systems`. Этот перечень дает информацию о каждой системе кодирования, включая букву, обозначающую ее в строке режима (см. Раздел 1.3 [Строка режима], с. 25).

Каждая система кодирования из перечисленных в этом списке — кроме `no-conversion`, что означает не делать никаких преобразований — указывает, как преобразовывать печатные знаки и нужно ли это делать, но оставляет выбор преобразования конца-строки до решения, основанном на содержимом файла. Например, если оказалось, что в файле для разделения строк используется последовательность возврат каретки-перевод строки, будет использовано преобразование из конца-строки DOS.

Каждая из перечисленных систем кодирования имеет три варианта, которые точно указывают, что делать для преобразования конца-строки:

`...-unix` Не производить преобразования конца-строки; предполагается, что в файле для разделения строк используется перевод строки. (Это соглашение обычно используется в системах Unix и GNU.)

`...-dos` Предполагать, что в файле для разделения строк используется возврат каретки-перевод строки, и делать соответствующее преобразование. (Это соглашение обычно используется в системах Microsoft.[2])

`...-mac` Предполагать, что в файле для разделения строк используется возврат каретки, и делать соответствующее преобразование. (Это соглашение обычно используется в системе Macintosh.)

Эти варианты систем кодирования опускаются для краткости в выводе `list-coding-systems`, поскольку они полностью предсказуемы. Например, система кодирования `iso-latin-1` имеет варианты `iso-latin-1-unix`, `iso-latin-1-dos` и `iso-latin-1-mac`.

Система кодирования `raw-text` хороша для файлов, которые содержат в основном ASCII-текст, но могут включать байты со значениями выше 127, которые не предназначались для кодирования не-ASCII-знаков. С `raw-text`, Emacs копирует эти байты без изменений и, чтобы они интерпретировались правильно, устанавливает в текущем буфере `enable-multibyte-characters` равной `nil`. `raw-text` обрабатывает преобразование конца-строки обычным способом, основываясь на увиденных данных, и имеет три обычных варианта для указания нужного преобразования конца-строки.

В противоположность этому, система кодирования `no-conversion` не задает никакого преобразования кодов знаков вообще — ни для значений байт, выходящих за пределы ASCII, ни для конца-строки. Это полезно для считывания и записи двоичных файлов, tar-файлов и других, которые нужно просматривать буквально. Она тоже устанавливает `enable-multibyte-characters` в значение `nil`.

Простейший способ отредактировать файл без любых преобразований — воспользоваться командой `M-x find-file-literally`. Она использует `no-conversion`, а также подавляет другие средства Emacs, которые могли бы преобразовать содержимое файла до того, как вы его увидите. См. Раздел 14.2 [Обращение], с. 106.

Система кодирования `emacs-mule` полагает, что файл содержит не-ASCII-знаки во внутренней кодировке Emacs. Она обрабатывает преобразование конца-строки, основываясь на увиденных данных, и имеет три обычных варианта для указания нужного преобразования конца-строки.

## 18.8 Распознавание систем кодирования

Чаще всего Emacs может распознать, какую систему кодирования он должен использовать для любого данного файла, — если вы указали свои предпочтения.

Некоторые системы кодирования могут быть распознаны или выделены по тому, какие последовательности знаков появляются среди данных. Однако, есть системы кодирования, которые не могут быть различены, даже потенциально. Например, нет способа отличить Latin-1 от Latin-2; они используют одни и те же значения байт с разными смыслами.

Emacs справляется с такой ситуацией при помощи списка приоритетов систем кодирования. Если вы не указали, какую систему кодирования надо использовать, Emacs во время считывания файла сверяет данные с каждой системой кодирования, начиная с первой по приоритету и продвигаясь вниз по списку, пока не найдет систему кодирования, подходящую для этого файла. Затем он преобразует содержимое файла, предполагая, что оно представлено в этой системе кодирования.

[2] Оно также предписано для тел MIME ‘text/*’ и других контекстов пересылки по сети. Это отличается от формата синтаксиса ссылок SGML начало-записи/конец-записи, который Emacs не поддерживает напрямую.

Список приоритетов систем кодирования зависит от выбранной языковой среды (см. Раздел 18.3 [Языковые среды], с. 162). Например, если вы используете французский, вы, вероятно, захотите, чтобы Emacs предпочитал Latin-1, а не Latin-2; а если вы используете чешский — чтобы предпочтение отдавалось Latin-2. Это одна из причин задавать языковую среду.

Однако, вы можете детально изменять список приоритетов с помощью команды `M-x prefer-coding-system`. Эта команда считывает имя системы кодирования в минибуфере и добавляет ее в начало списка приоритетов, так, чтобы ей отдавалось предпочтение среди остальных. Если вы применяете эту команду несколько раз, при каждом использовании в начало списка приоритетов добавляется один элемент.

Если вы используете систему кодирования, которая определяет тип преобразования последовательности конец-строки, такую как `iso-8859-1-dos`, то это означает, что Emacs должен попытаться распознать предпочтительно `iso-8859-1` и использовать преобразование конца-строки DOS, если `iso-8859-1` была распознана.

Иногда имя файла указывает на то, какая система кодирования должна для него использоваться. Это соответствие задает переменная `file-coding-system-alist`. Для добавления элементов к этому списку есть особая функция, `modify-coding-system-alist`. К примеру, чтобы все ‘`.txt`’-файлы считывались и записывались с использованием системы кодирования `china-iso-8bit`, вы можете выполнить следующее лисповское выражение:

```
(modify-coding-system-alist 'file "\\.txt\\'" 'china-iso-8bit)
```

Первым аргументом должен быть `file`, вторым — регулярное выражение, определяющее, к каким файлам это относится, а третий аргумент говорит, какую систему кодирования применять для этих файлов.

Emacs узнаёт, какой вид преобразования конца-строки следует использовать, основываясь на содержимом файла: если он видит только возвраты каретки или только последовательности возврат каретки-перевод строки, то выбирает соответствующее преобразование. Вы можете подавить автоматическое использование преобразования конца-строки, установив переменную `inhibit-eol-conversion` в значение `nil`.

Вы можете указать систему кодирования для конкретного файла, применяя конструкцию ‘`-*-...-*-`’ в начале этого файла или в списке локальных переменных в его конце (см. Раздел 31.2.5 [Переменные файла], с. 351). Вы делаете это, определяя значение для “переменной” с именем `coding`. На самом деле в Emacs нет переменной `coding`; вместо установки переменной он использует заданную систему кодирования для этого файла. Например, ‘`-*-mode: C; coding: latin-1;-*-`’ велит использовать систему кодирования Latin-1 и режим C. Если вы явно указали систему кодирования в файле, она перекрывает `file-coding-system-alist`.

Переменная `auto-coding-alist` — это самый сильный способ указать систему кодирования для определенных образцов имен файлов; эта переменная даже перекрывает теги ‘`-*-coding:-*-`’ в самом файле. Emacs использует это средство для tar-файлов и архивов, чтобы избежать ошибочной интерпретации тега ‘`-*-coding:-*-`’ в элементе архива как относящегося ко всему архивному файлу.

Когда Emacs выбрал систему кодирования для буфера, он сохраняет ее в `buffer-file-coding-system` и по умолчанию использует эту систему кодирования для операций, которые записывают этот буфер в файл. Это включает команды `save-buffer` и `write-region`. Если вы хотите записывать файлы из этого буфера, используя другую систему кодирования, вы можете указать для этого файла новую систему кодирования с помощью `set-buffer-file-coding-system` (см. Раздел 18.9 [Задание кодирования], с. 168).

Когда вы посылаете сообщение с помощью режима Mail (см. Глава 26 [Посылка почты], с. 267), у Emacs есть четыре разных способа узнать систему кодирования для текста сообщения. Он пробует значение `buffer-file-coding-system`, собственное для этого буфера, если оно не равно `nil`. Иначе, он использует значение `sendmail-coding-system`, если

оно не равно `nil`. Третий способ — использовать систему кодирования, принимаемую по умолчанию для новых файлов, которая управляется вашей языковой средой, если она не `nil`. Если все три эти значения равны `nil`, Emacs кодирует исходящую почту, используя систему кодирования Latin-1.

Когда вы получаете новую почту в Rmail, каждое сообщение автоматически переводится из той системы кодирования, в которой оно было написано — как если бы оно было отдельным файлом. При этом используется заданный вами список приоритетов систем кодирования. Если в сообщении в формате MIME указан набор знаков, Rmail подчиняется этому указанию, если `rmail-decode-mime-charset` не равна `nil`.

Для считывания и сохранения самих Rmail-файлов Emacs использует систему кодирования, задаваемую переменной `rmail-file-coding-system`. Значение по умолчанию равно `nil`, что означает, что Rmail-файлы не переводятся (они считываются и сохраняются во внутренней кодировке Emacs).

## 18.9 Задание системы кодирования

В случаях, когда Emacs не может автоматически подобрать правильную систему кодирования, вы можете указать ее явно с помощью таких команд:

`C-x` ⟨RET⟩ `f` *кодирование* ⟨RET⟩
: Использовать систему кодирования *кодирование* для файла, к которому обращается текущий буфер.

`C-x` ⟨RET⟩ `c` *кодирование* ⟨RET⟩
: Задает систему кодирования *кодирование* для непосредственно следующей команды.

`C-x` ⟨RET⟩ `k` *кодирование* ⟨RET⟩
: Использовать систему кодирования *кодирование* для ввода с клавиатуры.

`C-x` ⟨RET⟩ `t` *кодирование* ⟨RET⟩
: Использовать систему кодирования *кодирование* для вывода на терминал.

`C-x` ⟨RET⟩ `p` *код-ввода* ⟨RET⟩ *код-вывода* ⟨RET⟩
: Использовать системы кодирования *код-ввода* и *код-вывода* для ввода и вывода подпроцесса текущего буфера.

`C-x` ⟨RET⟩ `x` *кодирование* ⟨RET⟩
: Использовать систему кодирования *кодирование* для передачи выделений другим программам и получения их из других программ через оконную систему.

`C-x` ⟨RET⟩ `X` *кодирование* ⟨RET⟩
: Использовать систему кодирования *кодирование* для передачи или получения *одного* выделения — следующего — в оконную систему или из нее.

Команда `C-x` ⟨RET⟩ `f` (`set-buffer-file-coding-system`) задает систему кодирования файла для текущего буфера — другими словами, указывает, какую систему кодирования следует использовать для сохранения или повторного считывания этого файла. Вы задаете систему кодирования в минибуфере. Так как эта команда применяется только к файлу, к которому вы уже обратились, она влияет лишь на способ сохранения этого файла.

Другой способ указать систему кодирования для файла — сделать это во время обращения. Сначала используйте команду `C-x` ⟨RET⟩ `c` (`universal-coding-system-argument`); эта команда считывает в минибуфере имя системы кодирования. После выхода из минибуфера заданная система кодирования применяется для *непосредственно следующей команды*.

Таким образом, если непосредственно следующей командой будет, скажем, `C-x C-f`, то она считает файл, используя указанную систему кодирования (и запоминает эту систему

кодирования для последующей записи файла). Или, если следующей командой будет `C-x C-w`, она запишет файл, используя эту систему кодирования. Другие команды работы с файлами, на которые действует заданная система кодирования, включают `C-x C-i` и `C-x C-v`, а также варианты `C-x C-f` с показом в другом окне.

`C-x` ⟨RET⟩ `c` также влияет на программы, начинающие подпроцессы, включая `M-x shell` (см. Раздел 30.2 [Оболочка], с. 323).

Однако, если непосредственно следующая команда не использует систему кодирования, то `C-x` ⟨RET⟩ `c` в результате не имеет эффекта.

Простой способ обратиться к файлу без преобразования предоставляет команда `M-x find-file-literally`. См. Раздел 14.2 [Обращение], с. 106.

Переменная `default-buffer-file-coding-system` определяет выбор системы кодирования для вновь создаваемых файлов. Она применяется, когда вы обращаетесь к новому файлу или создаете буфер и затем сохраняете его в файл. При выборе языковой среды эта переменная как правило устанавливается в подходящее значение по умолчанию.

Команда `C-x` ⟨RET⟩ `t` (`set-terminal-coding-system`) задает систему кодирования для терминального вывода. Если вы зададите систему кодирования для терминального вывода, все выводимые на терминал знаки переводятся в эту систему.

Это средство полезно для некоторых текстовых терминалов, сделанных с поддержкой какого-то конкретного языка или набора знаков — например, европейских терминалов, поддерживающих один из наборов знаков ISO Latin. При использовании многобайтного текста вам нужно указать систему кодирования, чтобы Emacs знал, какие знаки этот терминал может на самом деле обработать.

По умолчанию вывод на терминал не преобразуется совсем, если только Emacs не может предугадать правильную систему кодирования для вашего типа терминала.

Команда `C-x` ⟨RET⟩ `k` (`set-keyboard-coding-system`) задает систему кодирования для ввода с клавиатуры. Перевод кодов вводимых с клавиатуры знаков полезен для терминалов, клавиши которых посылают графические не-ASCII-знаки, например, для некоторых терминалов, разработанных для кодировки ISO Latin-1 или ее подмножеств.

По умолчанию ввод с клавиатуры не переводится.

Между использованием системы кодирования для перевода ввода с клавиатуры и использованием метода ввода есть некое сходство: в обоих случаях определяются вводимые с клавиатуры последовательности, превращающиеся с один знак. Однако, методы ввода разработаны для удобного интерактивного использования людьми, и переводимые ими последовательности обычно являются последовательностями печатных ASCII-знаков. Системы кодирования как правило переводят последовательности неграфических знаков.

Команда `C-x` ⟨RET⟩ `x` (`set-selection-coding-system`) задает систему кодирования для передачи выделенного текста оконной системе и для получения текста выделений, сделанных в других приложениях. Эта команда относится ко всем будущим выделениям, пока вы не отмените это, снова применив эту команду. Команда `C-x` ⟨RET⟩ `X` (`set-next-selection-coding-system`) задает систему кодирования для следующего выделения, сделанного в Emacs или считанного Emacs.

Команда `C-x` ⟨RET⟩ `p` (`set-buffer-process-coding-system`) задает систему кодирования для ввода и вывода подпроцесса. Эта команда относится к текущему буферу; как правило, каждый подпроцесс имеет собственный буфер, следовательно, вы можете указывать перекодировку ввода и вывода процесса, давая эту команду в соответствующем буфере.

По умолчанию ввод и вывод процессов не переводится совсем.

Переменная `file-name-coding-system` задает систему кодирования, используемую для кодирования имен файлов. Если вы установите ее равной имени системы кодирования (это лисповский символ или строка), Emacs станет кодировать имена файлов при всех файловых операциях, используя эту систему кодирования. Это позволяет использовать в

именах файлов не-ASCII-знаки, или по крайней мере те не-ASCII-знаки, которые могут быть закодированы текущей системой кодирования.

Если `file-name-coding-system` равна `nil`, Emacs использует систему кодирования по умолчанию, определяемую языковой средой. В языковой среде, принимаемой по умолчанию, любые знаки в именах файлов, не входящие в ASCII, никак особенно не кодируются; они появляются в файловой системе во внутреннем представлении Emacs.

**Внимание:** если вы измените `file-name-coding-system` (или языковую среду) в середине сеанса Emacs, вы можете столкнуться с проблемами, если вы уже обратились к файлам, чьи имена были закодированы с использованием старой системы кодирования и не могут быть представлены (или кодируются иначе) в новой системе кодирования. Если вы попытаетесь сохранить один из таких буферов под именем файла, к которому он обращается, может быть использовано неправильное имя или может возникнуть ошибка. Если случается такая проблема, используйте `C-x C-w`, чтобы задать для этого буфера новое имя файла.

## 18.10 Наборы шрифтов

Шрифт X Windows обычно определяет начертание для одного алфавита или письменности. Поэтому для отображения полного спектра всех систем письма, которые поддерживает Emacs, необходимо множество шрифтов. В Emacs такое множество называется *набором шрифтов*. Набор шрифтов определяется как список шрифтов, каждый из которых предназначается для работы с одним диапазоном кодов знаков.

Каждый набор шрифтов имеет имя, как и отдельный шрифт. Доступные шрифты определяются X-сервером; наборы шрифтов определяются внутри самого Emacs. Как только вы определили набор шрифтов, вы можете использовать его в Emacs, указывая его имя в любом контексте, где вы могли бы написать один шрифт. Разумеется, наборы шрифтов Emacs могут содержать только те шрифты, которые поддерживаются X-сервером; если некоторые знаки появляются на экране как пустые прямоугольники, это означает, что в используемом наборе шрифтов нет шрифта для этих знаков.

Emacs создает два набора шрифтов автоматически: *стандартный набор шрифтов* и *стартовый набор шрифтов*. Стандартный набор шрифтов скорее всего содержит шрифты для широкого спектра знаков, не входящих в ASCII; однако, по умолчанию Emacs использует не его. (По умолчанию Emacs старается найти шрифт, которые имеет жирный и курсивный варианты.) Вы можете указать, что нужно использовать стандартный набор шрифтов, с помощью ключа '`-fn`' или с помощью X-ресурса '`Font`' (см. Раздел A.7 [Шрифт X], с. 392). Например,

```
emacs -fn fontset-standard
```

Набор шрифтов не обязан задавать шрифт для каждого кода. Если набор шрифтов не определяет шрифт для некоторого знака, или его он определяет шрифт, которого нет в вашей системе, то он не может правильно отобразить этот знак. Вместо этого знака будет показан пустой прямоугольник.

Высота и ширина набора шрифтов определяются ASCII-знаками (то есть шрифтами, используемыми в этом наборе для ASCII-знаков). Если другой шрифт в этом наборе имеет иную высоту или ширину, то знаки, приписанные к этому шрифту, обрезаются до размера набора шрифтов. Если `highlight-wrong-size-font` отлична от `nil`, то вокруг знаков с неправильным размером еще выводится прямоугольник.

## 18.11 Определение наборов шрифтов

Emacs создает стандартный набор шрифтов автоматически в соответствии с `standard-fontset-spec`. Именем этого набора является

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

или просто ‘`fontset-standard`’ для краткости.

Жирный, курсивный и жирный курсивный варианты стандартного набора шрифтов создаются автоматически. Их имена имеют ‘`bold`’ вместо ‘`medium`’, или ‘`i`’ вместо ‘`r`’ или и то, и другое.

Если вы задали ASCII-шрифт по умолчанию с помощью ресурса ‘`Font`’ или аргумента ‘`-fn`’, Emacs автоматически генерирует из него набор шрифтов. Это *стартовый набор шрифтов*, и его имя — `fontset-startup`. Emacs делает это, заменяя в имени шрифта поля *foundry*, *family*, *add_style* и *average_width* на ‘`*`’, заменяя *charset_registry* на ‘`fontset`’, а поле *charset_encoding* — на ‘`startup`’ и используя затем полученную строку для задания набора шрифтов.

К примеру, если вы запустили Emacs таким образом:

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

Emacs генерирует следующий набор шрифтов и использует его для первого фрейма:

```
-*-*-medium-r-normal-*-14-140-*-*-*-*-fontset-startup
```

В X-ресурсе ‘`Emacs.Font`’ вы можете указывать набор шрифтов, точно так же, как и обычное имя шрифта. Но будьте внимательны и не задавайте набор шрифтов в ресурсе с символами подстановки, как ‘`Emacs*Font`’, — такая спецификация применяется для различных целей, например для меню, а меню не может обращаться с наборами шрифтов.

Вы можете определить дополнительные наборы шрифтов, используя X-ресурсы с именами ‘`Fontset-`*n*’, где *n* — число, отсчитываемое от нуля. Значение этого ресурса должно иметь такую форму:

*шаблон-шрифта*`,` [*имя-кодировки*`:`*имя-шрифта*]. . .

*шаблон-шрифта*, кроме двух последних полей, должен иметь форму стандартного имени X-шрифта. Два последних поля должны иметь вид ‘`fontset-`*псевдоним*’.

У набора шрифтов есть два имени, одно длинное, а другое короткое. Длинное имя — это *шаблон-шрифта*. Короткое имя — это ‘`fontset-`*псевдоним*’. Вы можете ссылаться на набор шрифтов по любому из этих имен.

Конструкция ‘*кодировка*`:`*шрифт*’ определяет, какой шрифт должен использоваться (в этом наборе) для одного конкретного набора знаков. Здесь *кодировка* — это имя набора знаков, а *шрифт* — это используемый для него шрифт. При определении одного набора шрифтов вы можете применять эту конструкцию любое число раз.

Для остальных наборов знаков Emacs выбирает шрифт, основываясь на *шаблоне-шрифта*. Он заменяет ‘`fontset-`*псевдоним*’ на значения, описывающие набор знаков. Для шрифта знаков ASCII, ‘`fontset-`*псевдоним*’ заменяется на ‘`ISO8859-1`’.

Кроме того, когда несколько последовательных полей являются символами подстановки, Emacs сжимает их в один символ. Это делается для предотвращения использования автоматически масштабированных шрифтов. Шрифты, получаемые масштабированием более крупного шрифта, непригодны для редактирования, а масштабирование мелкого шрифта бессмысленно, потому что мелкий шрифт лучше использовать с его собственным размером, что Emacs и делает.

Таким образом, если *шаблон-шрифта* задан так:

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

то спецификация шрифта для ASCII-знаков была бы такой:

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

а спецификация шрифта для китайских знаков GB2312 такой:

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

У вас может не оказаться китайских шрифтов, соответствующих приведенной выше спецификации. Большинство дистрибутивов X Windows включают только китайские

шрифты с ‘song ti’ или ‘fangsong ti’ в поле *family*. В таком случае ‘Fontset-*n*’ можно задать таким образом:

```
Emacs.Fontset-0: -*-fixed-medium-r-normal-*-24-*-*-*-*-*-fontset-24,\
        chinese-gb2312:-*-*-medium-r-normal-*-24-*-gb2312*-*
```

Тогда спецификации всех шрифтов, кроме китайских GB2312, будут иметь ‘fixed’ в поле *family*, а спецификации для китайских знаков GB2312 несут в поле *family* символ подстановки ‘*’.

Функция, которая обрабатывает значение ресурса, определяющего набор шрифтов, и создает этот набор, называется `create-fontset-from-fontset-spec`. Вы также можете вызывать эту функцию явно, чтобы сгенерировать набор шрифтов.

См. Раздел A.7 [Шрифт X], с. 392, для большей информации об именовании шрифтов в X.

## 18.12 Поддержка однобайтных европейских знаков

Наборы знаков ISO 8859 Latin-*n* определяют коды знаков в диапазоне от 160 до 255 для обращения с акцентированными буквами и знаками препинания, необходимыми в различных европейских языках. Если вы выключите поддержку многобайтных знаков, Emacs все же сможет работать с *одной* из этих кодировок. Чтобы указать, *какие* из этих кодов следует использовать, вызовите `M-x set-language-environment` и задайте подходящую языковую среду, такую как ‘Latin-*n*’.

Для получения большей информации об однобайтном режиме смотрите Раздел 18.2 [Включение многобайтных знаков], с. 161. В частности, обратите внимание на то, что ваши файлы инициализации считываются как однобайтные, если они содержат не-ASCII-знаки.

Emacs может также отображать такие знаки, при условии, что они поддерживаются терминалом или шрифтом. Это работает автоматически. Или, если вы используете оконную систему, Emacs может отображать однобайтные знаки через наборы шрифтов, показывая в действительности эквивалентные многобайтные знаки в соответствии с языковой средой. Чтобы затребовать это, установите переменную `unibyte-display-via-language-environment` в отличное от `nil` значение.

Если ваш терминал не поддерживает набор знаков Latin-1, Emacs может отображать их как ASCII-последовательности, которые по крайней мере дают вам ясное представление о том, что это за знаки. Чтобы сделать так, загрузите библиотеку `iso-ascii`. Могут быть реализованы похожие библиотеки и для других наборов знаков Latin-*n*, но пока их у нас нет.

Обычно не входящие в ISO-8859 знаки (между 128 и 159 включительно) отображаются как восьмиричные управляющие последовательности. Вы можете изменить это для нестандартных ‘расширенных’ версий наборов знаков ISO-8859, используя функцию `standard-display-8bit` из библиотеки `disp-table`.

Есть три разных способа вводить однобайтные не-ASCII-знаки:

- Если ваша клавиатура может генерировать коды знаков от 128 и выше, представляющие знаки, не входящие в ASCII, выполните следующее выражение, чтобы Emacs смог их понимать:

```
(set-input-mode (car (current-input-mode))
                (nth 1 (current-input-mode))
                0)
```

- Вы можете использовать метод ввода для выбранной языковой среды. См. Раздел 18.4 [Методы ввода], с. 163. Когда вы используете метод ввода в однобайтном буфере, задаваемые с его помощью знаки переводятся в однобайтное представление.

- Для ввода печатных знаков Latin-1 вы можете использовать `C-x 8` как префикс “составления”. `C-x 8` удобен для вставки (в минибуфере, а также в остальных буферах), для поиска и во всех других контекстах, где допускаются последовательности знаков.

  `C-x 8` работает путем загрузки библиотеки `iso-transl`. Когда эта библиотека загружена, клавиша-модификатор ⟨ALT⟩, если она у вас есть, служит для той же цели, что и `C-x 8`; используйте ⟨ALT⟩ вместе со знаком акцента, чтобы модифицировать следующую букву. Кроме того, если у вас есть залипающие клавиши для генерации акцентов Latin-1, то они тоже определены для компоновки со следующим знаком, если `iso-transl` загружена.

# 19 Основные режимы

Emacs предоставляет много различных *основных режимов*, каждый из которых настраивает Emacs на редактирование текста определенного вида. Основные режимы являются взаимоисключающими, и каждый буфер находится в каждый момент времени в одном основном режиме. Строка режима обычно содержит имя текущего основного режима в круглых скобках (см. Раздел 1.3 [Строка режима], с. 25).

Наименее специализированный основной режим называется *Fundamental*. Этот режим не имеет специальных режимных переопределений или устанавливаемых переменных, так что каждая команда Emacs ведет себя самым обычным образом и каждый параметр находится в своем состоянии по умолчанию. Для редактирования некоторого текста определенного типа, такого как код на Лиспе или английский текст, вы должны переключить Emacs в соответствующий основной режим, такой как режим Lisp или режим Text.

Выбор основного режима изменяет значение нескольких ключей таким образом, чтобы они стали более приспособленным к редактируемому языку. Одни из наиболее часто изменяемых ключей — это ⟨TAB⟩, ⟨DEL⟩ и `C-j`. Префиксный ключ `C-c` обычно содержит команды, специфичные для режима. Помимо этого, команды для управления комментариями используют режим для определения того, каким образом комментарии должны ограничиваться. Многие основные режимы переопределяют синтаксические свойства знаков, появляющихся в буфере. См. Раздел 31.6 [Синтаксис], с. 366.

Основные режимы делятся на три основных группы. Режим Lisp (который имеет несколько вариантов), режим C и режим Fortran — для специфических языков программирования. Режим Text, режим Nroff, режим TEX и режим Outline — для редактирования текста на естественном языке. Остальные основные режимы не предназначены для использования с файлами пользователей; они используются в буферах, создаваемых Emacs для специальных целей, это такие режимы, как режим Dired для буферов, созданных Dired (см. Глава 28 [Dired], с. 291), режим Mail для буферов, созданных при помощи `C-x m` (см. Глава 26 [Посылка почты], с. 267), и режим Shell для буферов, используемых для связи с подчиненным процессом оболочки (см. Раздел 30.2.2 [Интерактивная оболочка], с. 324).

Большинство основных режимов для языков программирования указывают, что только пустые строки разделяют абзацы. Таким образом, команды работы с абзацами остаются удобными. (См. Раздел 21.3 [Абзацы], с. 183.) Они так же заставляют режим Auto Fill использовать определение ⟨TAB⟩ для создания отступа во вновь создаваемых им строках. Это далется, поскольку большинство строк в программе обычно начинаются с отступа. (См. Глава 20 [Отступы], с. 177.)

## 19.1 Как выбираются основные режимы

Вы можете выбрать основной режим для текущего буфера явно, но чаще Emacs сам определяет, какой режим использовать, основываясь на имени файла или на специальном тексте в файле.

Явный выбор нового основного режима делается при помощи команды `M-x`. Чтобы получить имя команды для выбора режима, добавьте к имени основного режима окончание `-mode`. Таким образом, вы можете войти в режим Lisp, выполнив команду `M-x lisp-mode`.

Когда вы обращаетесь к файлу, Emacs обычно выбирает правильный основной режим, основываясь на имени этого файла. Например, файлы, чьи имена оканчиваются на ‘`.c`’, редактируются в режиме C. Соответствие между именем файла и основным режимом контролируется переменной `auto-mode-alist`. Ее значение — это список, каждый элемент которого имеет такой вид:

(*регулярное-выражение* . *функция-режима*)

или такой:

(*регулярное-выражение функция-режима флаг*)

Например, один элемент, обычно находящийся в этом списке, имеет вид `("\\.c\\'" . c-mode)`, и это является сигналом для выбора режима C для файлов, чьи имена кончаются на '`.c`'. (Отметим, что '`\\`' необходимо по синтаксису Лиспа для того, чтобы включить в эту строку знак '`\`', а он нужен для подавления специального значения '`.`' в регулярном выражении.) Если этот элемент имеет форму (*регулярное-выражение функция-режима флаг*), и *флаг* не `nil`, то после вызова *функции-режима* суффикс, совпавший с *регулярным-выражением*, отбрасывается, и в списке производится повторный поиск другого совпадения.

Вы можете указать, какой основной режим должен использоваться для редактирования определенного файла, с помощью текста специального вида в первой непустой строке файла. В этой строке должно появиться имя режима, до и после него должны стоять строки '`-*-`'. В этой строке также может появиться другой текст. Например,

```
;-*-Lisp-*-
```

приказывает Emacs использовать режим Lisp. Такое явное определение отменяет значение по умолчанию, основанное на имени файла. Отметим, что точка с запятой используется для того, чтобы Лисп трактовал эту строку как комментарий.

Другой формат определения режима:

`-*- mode:` *имя-режима*`;-*-`

что позволяет вам также задать локальные переменные, как здесь:

`-*- mode:` *имя-режима*`;` *пер*`:` *значение*`; ... -*-`

См. Раздел 31.2.5 [Переменные файла], с. 351, для получения большей информации об этом.

Если содержимое файла начинается с '`#!`', он может служить в качестве выполняемой команды оболочки, которая работает путем запуска интерпретатора, названного в первой строке этого файла. Остальная часть файла подается на вход интерпретатора.

Когда вы обращаетесь к подобному файлу в Emacs, если имя файла не задает основной режим, Emacs использует для выбора режима имя интерпретатора в первой строке. Если первая строка — это имя поддерживаемой программы-интерпретатора, такой как '`perl`' или '`tcl`', Emacs использует режим, подходящий для программ для этого интерпретатора. Переменная `interpreter-mode-alist` задает соответствие между именами программ-интерпретаторов и основными режимами.

Когда первая строка начинается с '`#!`', вы не можете (на многих системах) использовать в ней '`-*-`', поскольку при запуске интерпретатора это ввело бы в заблуждение систему. Поэтому в таких файлах Emacs ищет '`-*-`' на второй строке, а не только на первой.

Когда вы обращаетесь к файлу, который не указывает, какой основной режим использовать, или когда вы создаете новый буфер при помощи `C-x b`, то используемым основным режимом является тот, что определен переменной `default-major-mode`. Обычно ее значение — это символ `fundamental-mode`, который задает режим Fudamental. Если `default-major-mode` равна `nil`, то основной режим берется из ранее выбранного буфера.

Если вы изменили основной режим буфера, вы можете вернуться к тому основному режиму, который Emacs выбрал бы автоматически: используйте для этого команду `M-x normal-mode`. Это та же функция, которую вызывает `find-file` для выбора основного режима. Она также обрабатывает список локальных переменных файла, если он есть.

Команды `C-x C-w` и `set-visited-file-name` переключают в новый основной режим, если новое имя файла подразумевает выбор режима (см. Раздел 14.3 [Сохранение], с. 108). Однако, это не происходит, если содержимое буфера задает основной режим; и некоторые "специальные" основные режимы не допускают изменения режима. Вы можете выключить эту возможность переключения режимов, установив `change-major-mode-with-file-name` в значение `nil`.

# 20 Отступы

Эта глава описывает команды Emacs, которые создают, убирают или настраивают отступы.

⟨TAB⟩ Сделать отступ текущей строки, “соответствующий” режиму.

`C-j` Выполнить ⟨RET⟩, за которым следует ⟨TAB⟩ (`newline-and-indent`).

`M-^` Слить две строки (`delete-indentation`). Это отменяет действие `C-j`.

`C-M-o` Разбить строку в точке; текст на строке после точки становится новой строкой с отступом до того столбца, с которого он начинается сейчас (`split-line`).

`M-m` Передвинуться (вперед или назад) к первому непустому знаку на текущей строке (`back-to-indetation`).

`C-M-\` Сделать отступ нескольких строк до одного и того же столбца (`indent-region`).

`C-x` ⟨TAB⟩ Жестко сдвинуть блок строк влево или вправо (`indent-rigidly`).

`M-i` Сделать отступ от точки к следующему предопределенному столбцу позиции табуляции (`tab-to-tab-stop`).

`M-x indent-relative`
Сделать отступ от точки к месту под точкой отступа в предыдущей строке.

Большинство языков программирования имеют некоторое соглашение по отступам. Для Лисп-кода отступ строк выполняется согласно их вложенности в круглые скобки. Та же самая общая идея используется для кода на Си, хотя многие детали отличаются.

В любом языке для создания отступа в строке используется команда ⟨TAB⟩. Каждый основной режим определяет эту команду так, чтобы она выполняла соответствующий этому языку отступ. В режиме Lisp ⟨TAB⟩ расставляет строки в соответствии с их глубиной вложенности в круглые скобки. Вне зависимости от того, в каком месте строки вы находитесь, когда набираете ⟨TAB⟩, она выравнивает строку целиком. В режиме C, ⟨TAB⟩ осуществляет утонченный и сложный стиль отступа, который знает о многих аспектах синтаксиса Си.

В режиме Text, ⟨TAB⟩ запускает команду `tab-to-tab-stop`, которая делает отступ к следующему столбцу позиции табуляции. Вы можете установить позиции табуляции с помощью `M-x edit-tab-stops`.

## 20.1 Способы и команды отступа

Чтобы передвинуться через отступ на строке, сделайте `M-m` (`back-to-indentation`). Эта команда, данная где угодно на строке, помещает точку на первый непустой знак в этой строке.

Чтобы вставить строку с отступом перед текущей строкой, сделайте `C-a C-o` ⟨TAB⟩. Чтобы сделать строку с отступом после текущей строки, используйте `C-e C-j`.

Если вы просто хотите вставить в буфер символ табуляции, то вы можете набрать `C-q` ⟨TAB⟩.

`C-M-o` (`split-line`) сдвигает текст от точки до конца строки вертикально вниз, так что текущая строка становится двумя строками. `C-M-o` сначала передвигает точку вперед через любое количество пробелов и табуляций. Затем она вставляет после точки ограничитель строки и достаточное количество отступов, чтобы достичь того же столбца, на котором находится точка. Точка остается перед вставляемым переводом строки; с этой точки зрения `C-M-o` напоминает `C-o`.

Чтобы начисто соединить две строки, используйте команду `M-^` (`delete-indentation`). Она удаляет отступ в начале текущей строки, а так же ограничитель строки, заменяя их одиночным пробелом. В особом случае (полезном для кода на Лиспе) одиночный пробел опускается, если соединяемыми знаками являются последовательные открывающие или закрывающие круглые скобки, или если после слияния идет еще одна новая строка. Чтобы удалить просто отступ строки, перейдите в начало строки и используйте `M-\` (`delete-horizontal-space`), которая удаляет все пробелы и табуляции около курсора.

Если есть префикс заполнения, `M-^` убирает его, если он находится после удаляемого перевода строки. См. Раздел 21.5.3 [Префикс заполнения], с. 187.

Имеются также команды для изменения отступов нескольких строк сразу. `C-M-\` (`indent-region`) применяется для всех строк, которые начинаются в данной области; она делает для каждой из этих строк “обычный” отступ, как если бы вы напечатали ⟨TAB⟩ в начале строки. Числовой аргумент определяет столбец для отступа, и каждая строка сдвигается влево или вправо так, что ее первый непустой знак появляется в этом столбце. `C-x` ⟨TAB⟩ (`indent-rigidly`) сдвигает все строки в области вправо в соответствии со своим аргументом (влево при отрицательном аргументе). Вся группа строк жестко сдвигается в одну сторону, именно поэтому эта команда получила такое имя.

`M-x indent-relative` выполняет отступ точки, основываясь на предыдущей строке (фактически, по последней непустой строке). Она вставляет пробел в точке, двигая точку до тех пор, пока она не встанет под точкой отступа в предыдущей строке. Точка отступа является концом последовательности пробелов или концом строки. Если точка находится дальше вправо, чем любая точка отступа в предыдущей строке, то все пробельные знаки перед точкой удаляются, и используется первая применимая теперь точка отступа. Если даже после этого нет пригодной точки отступа, `indent-relative` запускает `tab-to-tab-stop` (смотрите следующий раздел).

`indent-relative` — это определение ⟨TAB⟩ в режиме Indented Text. См. Глава 21 [Текст], с. 181.

См. Раздел 21.11.6 [Отступы в форматированном тексте], с. 201, другой способ задать отступы для части вашего текста.

## 20.2 Позиции табуляции

Для набора таблиц вы можете использовать определение ⟨TAB⟩ в режиме Text, `tab-to-tab-stop`. Эта команда вставляет перед точкой отступ, достаточный для того, чтобы достичь следующего столбца позиции табуляции. Если вы находитесь не в режиме Text, эту функцию можно найти по ключу `M-i`.

Вы можете произвольно установить используемые в `M-i` позиции табуляции. Они запоминаются в переменной с именем `tab-stop-list` как список номеров столбцов в возрастающем порядке.

Удобный способ установить позиции табуляции — воспользоваться командой `M-x edit-tab-stops`, которая создает и выбирает буфер, содержащий описание установленных позиций табуляции. Вы можете отредактировать этот буфер для определения других позиций табуляции и затем набрать `C-c C-c`, чтобы сделать эти новые позиции табуляции действующими. `edit-tab-stops` запоминает, какой буфер был текущим, когда вы запускали ее, и записывает позиции табуляции обратно в этот буфер; обычно все буферы разделяют одни и те же позиции табуляции, и изменение их в одном буфере влияет на все, но если вам случится сделать `tab-stop-list` локальной в одном буфере, то `edit-tab-stops` будет редактировать локальные установки.

Покажем, как выглядит представляющий табуляцию текст для обычных позиций табуляции через каждые восемь столбцов.

```
        :       :       :       :       :       :
```

```
0         1         2         3         4
01234567890123456789012345678901234567890123456789012345678
To install changes, type C-c C-c
```

Первая строка содержит двоеточие в каждой позиции табуляции. Остальные строки представлены просто для того, чтобы помочь вам понять, где находится двоеточие, и сообщить, что вы можете делать.

Заметим, что позиции табуляции, которые управляют `tab-to-tab-stop`, не имеют ничего общего с показанными символами табуляции в буфере. См. Раздел 11.7 [Переменные изображения], с. 84, для более подробной информации на этот счет.

## 20.3 Табуляция по сравнению с пробелами

Обычно Emacs использует для отступа строк как табуляцию, так и пробелы. Если вы захотите, то все отступы будут делаться только при помощи пробелов. Чтобы потребовать это, установите переменную `indent-tab-mode` равной `nil`. Это переменная буфера; изменение ее влияет только на текущий буфер, но имеется и значение по умолчанию, которое вы тоже можете изменить. См. Раздел 31.2.4 [Локальные переменные], с. 350.

Существуют также команды для превращения табуляции в пробелы и наоборот, всегда сохраняющие столбцы всего непустого текста. `M-x tabify` находит в области последовательности пробелов и преобразует в табуляцию любую последовательность, состоящую по меньшей мере из трех пробелов, если это можно сделать без изменения отступа. `M-x untabify` заменяет все табуляции в области на соответствующее число пробелов.

# 21 Команды для естественных языков

Термин *текст* имеет два широко распространенных значения в нашей области компьютерной науки. Одно — это данные, которые являются последовательностью знаков. Любой файл, который вы редактируете при помощи Emacs, — это текст в этом смысле слова. Другое значение более узкое: последовательность знаков на естественном языке, предназначенная для чтения людьми (возможно после обработки форматированием), в противоположность программам или командам для программы.

В естественных языках приняты стилистические и синтаксические условности, которые могут поддерживаться или выгодно использоваться командами редактирования: это условности, включающие использование слов, предложений, абзацев и прописных букв. Данная глава описывает команды Emacs для всех этих вещей. Существуют также команды для *заполнения*, что означает перестройку строк абзацев таким образом, чтобы они были приблизительно равной длины. Команды для перемещения или уничтожения слов, предложений и абзацев, предназначенные в первую очередь для редактирования текста, часто бывают полезными и для редактирования программ.

Emacs имеет несколько основных режимов для редактирования текста на естественном языке. Если файл содержит несложный чистый текст, используйте режим Text, который быстро настраивает Emacs на синтаксические условности текста. Режим Outline предоставляет особые команды для действий с со структурированным текстом. См. Раздел 21.8 [Режим Outline], с. 190.

Для текста, который содержит встроенные команды для программ форматирования, Emacs имеет другие основные режимы, свой для каждого формата. Таким образом, для ввода в TEX вы должны использовать режим TEX (см. Раздел 21.9 [Режим TeX], с. 194). Для ввода в nroff — режим Nroff.

Вместо использования программы форматирования, вы можете редактировать форматированный текст в стиле WYSIWYG (“what you see is what you get”)[1] с помощью режима Enriched. Тогда форматирование появляется на экране в Emacs во время редактирования. См. Раздел 21.11 [Форматированный текст], с. 198.

## 21.1 Слова

В Emacs существуют команды для передвижения по словам или воздействия на них. По соглашению, все ключи для этого являются Meta-знаками.

`M-f` Перейти вперед через слово (`forward-word`).

`M-b` Перейти назад через слово (`backward-word`).

`M-d` Уничтожить вперед все вплоть до конца слова (`kill-word`).

`M-`⟨DEL⟩ Уничтожить назад все вплоть до начала слова (`backward-kill-word`).

`M-@` Пометить конец следующего слова (`mark-word`).

`M-t` Переставить два слова или перенести одно слово через другие слова (`transpose-words`).

Заметьте, как эти ключи образуют ряд, который соответствует ключам, работающим со знаками: `C-f`, `C-b`, `C-d`, ⟨DEL⟩ и `C-t`. `M-@` соответствует `C-@`, которая иначе называется `C-`⟨SPC⟩.

Команды `M-f` (`forward-word`) and `M-b` (`backward-word`) передвигают вперед или назад через слова. Таким образом, эти Meta-знаки аналогичны `C-f` и `C-b`, которые передвигают

[1] Что вы видите, то и получаете. *(Прим. переводчика)*

через одиночные знаки в тексте. Аналогия распространяется на числовые аргументя, которые служат счетчиками повторов. `M-f` с отрицательным аргументом передвигает назад, а `M-b` с отрицательным аргументом передвигает вперед. Движение вперед останавливается сразу после последней буквы слова, тогда как движение назад останавливается сразу перед первой буквой.

`M-d` (`kill-word`) уничтожает слово после точки. Точнее, она уничтожает все от точки до того места, куда переместила бы команда `M-f`. Таким образом, если точка находится в середине слова, `M-d` уничтожает только часть слова после точки. Если между точкой и следующим словом находятся какие-то знаки препинания, то они уничтожаются вместе со словом. (Если вы хотите уничтожить только следующее слово, но не уничтожать знаки препинания перед ним, то просто сделайте `M-f`, чтобы перейти на конец, и уничтожьте слово в обратном направлении при помощи `M-`⟨DEL⟩.) `M-d` трактует аргументы точно так же, как `M-f`.

`M-`⟨DEL⟩ (`backward-kill-word`) уничтожает слово перед точкой. Она уничтожает все от точки назад к тому месту, куда передвинула бы `M-b`. Если точка находится после пробела в ‘`FOO, BAR`’, то уничтожается ‘`FOO, `’. (Если вы хотите уничтожить просто ‘`FOO`’, сделайте `M-b M-d` вместо `M-`⟨DEL⟩.)

`M-t` (`transpose-words`) меняет местами слово, стоящее перед точкой или содержащее ее, со следующим словом. Разграничительные знаки между словами не сдвигаются. Например, ‘`FOO, BAR`’ превращается в ‘`BAR, FOO`’, а не в ‘`BAR FOO,`’. Для более подробной информации о перестановках и аргументах команд перестановки смотрите Раздел 13.2 [Перестановка], с. 101.

Чтобы подействовать на следующие $n$ слов с помощью операции, которая применяется между точкой и меткой, вы можете либо установить метку в точке и затем передвинуть точку через слова, либо использовать команду `M-@` (`mark-word`), которая не перемещает точку, но устанавливает метку туда, куда ее передвинула бы команда `M-f`. `M-@` принимает числовой аргумент, который говорит, через сколько слов нужно поместить метку. В режиме Transient Mark эта команда активизирует метку.

Понятие о синтаксисе у команд, работающих со словами, полностью управляется синтаксической таблицей. Любой знак может быть объявлен, например, как разделитель слов. См. Раздел 31.6 [Синтаксис], с. 366.

## 21.2 Предложения

Команды Emacs для действий над предложениями и абзацами в большинстве своем приданы Meta-ключам, чтобы они были подобны командам работы со словами.

`M-a` Перейти назад к началу предложения (`backward-sentence`).

`M-e` Перейти вперед к концу предложения (`forward-sentence`).

`M-k` Уничтожить вперед до конца предложения (`kill-sentence`).

`C-x` ⟨DEL⟩ Уничтожить все в обратном направлении до начала предложения (`backward-kill-sentence`).

Команды `M-a` и `M-e` (`backward-sentence` и `forward-sentence`) передвигают точку к началу и к концу текущего предложения, соответственно. Они выбраны так, чтобы напоминать `C-a` и `C-e`, которые сдвигают к концу и началу строки. В отличие от них, `M-a` и `M-e` при повторении или с заданными числовыми аргументами передвигают через последовательные предложения.

Перемещение назад через предложение помещает точку непосредственно перед первым знаком этого предложения; перемещение вперед помещает точку сразу после знака препинания, завершающего предложение. Ни одна из этих команд не перемещает через пропуски на границах предложений.

Точно так же, как `C-a` и `C-e` имеют соответствующую им команду уничтожения `C-k`, так и `M-a` и `M-e` имеют соответствующую команду уничтожения `M-k` (`kill-sentence`), которая уничтожает все от точки до конца предложения. С аргументом, равным минус единице, она уничтожает в обратном направлении до начала предложения. Большие аргументы служат для подсчета повторов. Есть также особая команда `C-x` ⟨DEL⟩ (`backward-kill-sentence`) для уничтожения в обратном направлении к началу предложения. Она удобна, когда вы меняете свое решение в процессе сочинения текста.

Команды работы с предложениями предполагают, что вы следуете соглашению американских машинисток — ставить в конце предложения два пробела; они считают предложение оконченным, если там есть знаки '`.`', '`?`' или '`!`', за которыми следует конец строки или два пробела; в середине допустимо любое число знаков '`)`', '`]`' или '`"`'. Предложение также начинается или кончается, если начинается или кончается абзац.

Переменная `sentence-end` управляет распознаванием конца предложения. Это регулярное выражение, которое соответствует последним нескольким знакам предложения вместе с пробелами, следующими за предложением. Его нормальное значение таково:

```
"[.?!][]\"')]*\\($\\|\t\\|  \\)[ \t\n]*"
```

Этот пример объясняется в разделе о регулярных выражениях. См. Раздел 12.5 [Регулярные выражения], с. 91.

Если вы хотите использовать между предложениями только один пробел, вам нужно установить `sentence-end` в такое значение:

```
"[.?!][]\"')]*\\($\\|\t\\| \\)[ \t\n]*"
```

Вам нужно также установить переменную `sentence-end-double-space` равной `nil`, чтобы команды заполнения ожидали и оставляли в конце предложений только один пробел. Заметьте, что при этом невозможно отличить точки, завершающие предложения, и точек в сокращениях.

## 21.3 Абзацы

Команды Emacs для работы с абзацами — это также Meta-ключи.

`M-{` Перейти назад к началу предыдущего абзаца (`backward-paragraph`).

`M-}` Переместиться вперед к концу следующего абзаца (`forward-paragraph`).

`M-h` Поставить точку и метку вокруг этого или следующего абзаца (`mark-paragraph`).

`M-{` двигает точку в начало текущего или предыдущего абзаца, в то время как `M-}` двигает ее к концу текущего или следующего абзаца. Абзацы разделяются пустыми строками и строками команд форматирования текста, которые в свою очередь не являются частью какого-либо абзаца. В режиме Fundamental, но не в режиме Text, строка с отступом также начинает новый абзац. (Если перед абзацем стоит пустая строка, данные команды считают эту пустую строку началом абзаца.)

В основных режимах для программ, абзацы начинаются и кончаются только пустыми строками. Это делает команды для абзацев по-прежнему удобными, даже хотя абзацев как таковых нет.

Когда имеется префикс заполнения, абзацы ограничиваются всеми строками, которые не начинаются с этого префикса. См. Раздел 21.5 [Заполнение], с. 185.

Когда вы захотите оперировать с абзацем, вы можете использовать команду `M-h` (`mark-paragraph`), чтобы установить вокруг него область. Таким образом, например, `M-h C-w` уничтожает абзац вокруг или после точки. Команда `M-h` ставит точку в начале абзаца, содержащего точку, и метку в его конце. В режиме Transient Mark она активизирует метку. Если точка находится между абзацами (в области пустых строк или на границе),

то точкой и меткой окружается абзац, следующий за точкой. Если первой строке абзаца предшествуют пустые строки, то одна из этих пустых строк включается в область.

Точным определением границ абзаца управляют две переменные: `paragraph-separate` и `paragraph-start`. Значение `paragraph-start` — это регулярное выражение, которое должно соответствовать любой строке, которая либо начинает, либо разделяет абзацы. Значение `paragraph-separate` — это еще одно регулярное выражение, которое должно соответствовать только строкам, которые разделяют абзац, но не являются частью какого-либо абзаца (например, пустые строки). Строки, которые начинают новый абзац и содержатся в нем, должны соответствовать только `paragraph-start`, но не `paragraph-separate`. Например, в режиме Fundamental, `paragraph-start` равна `"[ \t\n\f]"`, а `paragraph-separate` — это `"[ \t\f]*$"`.

Обычно желательно, чтобы границы страниц разделяли абзацы. Значения по умолчанию этих переменных распознают обычный разделитель страниц.

## 21.4 Страницы

Очень часто файлы представляются разделенными на *страницы* с помощью знаков *прогона* (или перевода) *страницы* (ASCII Control-L, восьмиричный код 014). Когда вы печатаете файл, этот знак принудительно разбивает страницу; таким образом, каждая страница файла будет начинаться на новом листе бумаги. Большинство команд Emacs рассматривают знак-разделитель страниц точно так же, как любые другие знаки: вы можете вставить их при помощи `C-q C-l` или удалить с помощью ⟨DEL⟩. Таким образом, вы свободны в выборе, делить на страницы ваш файл или нет. Однако, из-за того, что деление на страницы часто является смысловым делением файла, то предусмотрены команды для перемещения по страницам и для действий над ними.

`C-x [` Сместить точку к предыдущей странице (`backward-page`).

`C-x ]` Сместить точку к следующей странице (`forward-page`).

`C-x C-p` Поставить точку и метку по краям этой (или другой) страницы (`mark-page`).

`C-x l` Сосчитать строки в этой странице (`count-lines-page`).

Команда `C-x [` (`backward-page`) двигает точку к позиции непосредственно после предыдущего разделителя страницы. Если точка уже находится сразу после разделителя, то команда пропускает эту страницу и останавливается на предшествующей ей. Числовой аргумент служит в качестве счетчика повторов. Команда `C-x ]` (`forward-page`) передвигает точку вперед, пропуская следующий разделитель страниц.

Команда `C-x C-p` (`mark-page`) ставит точку в начале текущей страницы, а метку в ее конце. Разделитель страниц в конце включается в область (метка следует за ним). Разделитель страниц в начале не включается (точка следует за ним). `C-x C-p C-w` дает удобный способ уничтожить страницу или переместить ее в другое место. Если вы сдвинитесь к разделителю еще одной страницы с помощью `C-x [` и `C-x ]`, а затем восстановите уничтоженную страницу, все страницы будут снова правильно разграничины. `C-x C-p` включает в область только разделитель следующей страницы именно для этого.

Числовой аргумент для `C-x C-p` используется для указания страницы, к которой необходимо отправиться, относительно текущей. Ноль означает текущую страницу. Единица означает следующую страницу, а $-1$ — предыдущую.

Команда `C-x l` (`count-lines-page`) хороша для принятия решения, где разорвать страницу на две. Она печатает в эхо-области общее число строк в текущей странице и затем делит ее на те, которые предшествуют текущей строке, и на те, что следуют за ней, как в примере:

```
Page has 96 (72+25) lines
```
[2]

Заметьте, что значение суммы на единицу меньше; это верно, если точка не стоит в начале строки.

Переменная `page-delimiter` говорит, где начинается страница. Ее значение — это регулярное выражение, соответствующее началу строки, которая разделяет страницы. Обычное значение этой переменной равно `"^\f"`, что соответствует знаку перевода страницы в начале строки.

## 21.5 Заполнение текста

*Заполнение* текста означает разбиение его на строки определенной длины. Emacs может делать заполнение двумя способами. В режиме Auto Fill, вставка текста с помощью самовставляющихся знаков также автоматически заполняет его. Есть также явные команды для заполнения, которые вы можете использовать, когда редактирование текста оставляет его незаполненным. Когда вы редактируете форматированный текст, вы можете задать стиль заполнения каждого фрагмента (см. Раздел 21.11 [Форматированный текст], с. 198).

### 21.5.1 Режим Auto Fill

Режим *Auto Fill* — это второстепенный режим, в котором строки обрываются автоматически, когда становятся слишком длинными. Разрыв происходит только тогда, когда вы набираете ⟨SPC⟩ или ⟨RET⟩.

`M-x auto-fill-mode`
: Включение и выключение режима Auto Fill.

⟨SPC⟩
⟨RET⟩
: В режиме Auto Fill прерывает строку, если это нужно.

`M-x auto-fill-mode` включает режим Auto Fill, если он был отключен, или выключает, если он был включен. С положительным аргументом она всегда включает режим Auto Fill, а отрицательным — всегда отключает. Вы можете видеть, когда режим Auto Fill действует, по присутствию слова ‘`Fill`’ в строке режима внутри круглых скобок. Режим Auto Fill — второстепенный режим, включаемый или выключаемый для каждого буфера отдельно. См. Раздел 31.1 [Второстепенные режимы], с. 341.

В режиме Auto Fill строки автоматически разрываются на пробелах, когда они становятся длиннее желаемой величины. Прерывание и перерасположение строки происходит, только когда вы набираете ⟨SPC⟩ или ⟨RET⟩. Если вы хотите вставить пробел или знак новой строки с запретом прерывания строки, наберите `C-q` ⟨SPC⟩ или `C-q C-j` (напомним, что знак новой строки — это на самом деле control-J). `C-o` также вставляет новую строку без прерывания строки.

Режим Auto Fill хорошо работает с режимами для языков программирования, так как он делает в новых строках отступ с помощью ⟨TAB⟩. Если строка, заканчивающаяся комментарием, получилась слишком длинной, то текст комментария разбивается на две строки. Возможно, в конце первой строки и в начале второй вставятся новые ограничители комментариев, таким образом, чтобы каждая строка стала отдельным комментарием; этим выбором управляет переменная `comment-multi-line` (см. Раздел 22.7 [Комментарии], с. 219).

Адаптивное заполнение (смотрите следующий раздел) работает с режимом Auto Fill так же, как с явными командами заполнения. Оно автоматически берет префикс заполнения из второй или первой строки абзаца.

---

[2] Страница содержит 96 (72+25) строк. *(Прим. переводчика)*

Режим Auto Fill не перезаполняет целые абзацы; он может прерывать строки, но не может их объединять. Таким образом, редактирование в середине абзаца может привести к созданию абзаца, который неправильно заполнен. Простейшим способом сделать абзац снова правильно заполненным обычно служит применение явных команды заполнения.

Многие пользователи любят режим Auto Fill и хотят использовать его во всех текстовых файлах. Раздел о файлах инициализации рассказывает, как устроить, чтобы это было для вас постоянным. См. Раздел 31.7 [Файл инициализации], с. 366.

### 21.5.2 Явные команды заполнения

`M-q` Заполнить текущий абзац (`fill-paragraph`).

`C-x f` Установить столбец заполнения (`set-fill-column`).

`M-x fill-region`
Заполнить каждый абзац в области (`fill-region`).

`M-x fill-region-as-paragraph`
Заполнить область, рассматривая ее как один абзац.

`M-s` Отцентрировать строку.

Чтобы перезаполнить один абзац, используйте команду `M-q` (`fill-paragraph`). Она действует на абзац, в котором находится точка, или на абзац после точки, если она стоит между абзацами. Перезаполнение работает путем удаления всех разрывов строк и вставки новых в тех местах, где это требуется.

Чтобы перезаполнить много абзацев, используйте `M-x fill-region`, которая делит область на абзацы и заполняет каждый из них.

Команды `M-q` и `fill-region` используют для нахождения границ абзаца тот же самый критерий, что и `M-h` (см. Раздел 21.3 [Абзацы], с. 183). Для большего контроля, вы можете использовать `M-x fill-region-as-paragraph`, которая перезаполняет все между точкой и меткой. Эта команда удаляет в области все пустые строки, поэтому отдельные блоки текста в результате объединяются в один блок.

Числовой аргумент для `M-q` приводит к тому, что помимо заполнения, текст еще и *выравнивается*. Это значит, что вставляются дополнительные пробелы, чтобы правый край строки попадал точно в столбец заполнения. Чтобы уничтожить дополнительные пробелы, используйте `M-q` без аргумента. (Аналогично и для `fill-region`.) Другой способ управлять выравниванием или выбрать другие стили заполнения состоит в применении свойства текста `justification`; смотрите Раздел 21.11.7 [Формат Выравнивание], с. 202.

Команда `M-s` (`center-line`) центрирует текущую строку в пределах текущего столбца заполнения. С аргументом $n$, она центрирует несколько строк отдельно и переходит через них.

Максимальная ширина строки для заполнения содержится в переменной `fill-column`. Изменение значения `fill-column` делает ее локальной для текущего буфера; до этого момента действует значение по умолчанию. Изначально оно равно 70. См. Раздел 31.2.4 [Локальные переменные], с. 350. Наилегчайший способ установить `fill-column` — использовать команду `C-x f` (`set-fill-column`). Запущенная с числовым аргументом, она использует его в качестве нового столбца заполнения. Просто с `C-u` в качестве аргумента, она устанавливает `fill-column` соответственно текущей горизонтальной позиции точки.

Команды Emacs обычно рассматривают точку, за которой следуют два пробела или перевод строки, как конец предложения; точка, после которой идет только один пробел, указывает на сокращение и не является концом предложения. Чтобы сохранить разграничение между двумя этими вариантами использования точки, команды заполнения не обрывают строку после точки, за которой идет только один пробел.

Если переменная `sentence-end-double-space` равна `nil`, то команды заполнения ожидают и оставляют в конце предложений только один пробел. Обычно эта переменная равна `t`, поэтому команды заполнения настаивают на постановке двух пробелах в конце предложения, как объяснено выше. См. Раздел 21.2 [Предложения], с. 182.

Если переменная `colon-double-space` не равна `nil`, команды заполнения ставят после двоеточия два пробела.

### 21.5.3 Префикс заполнения

Чтобы заполнить абзац, в котором каждая строка начинается с особого маркера (который может несколькими пробелами, что дает абзац с отступом), используйте так называемый *префикс заполнения*. Префикс заполнения — это цепочка знаков, с которой, по предположению Emacs, начинается каждая строка, и которая не включается в заполнение. Вы можете задать префикс заполнения явно; кроме того, Emacs может вычислять его автоматически (см. Раздел 21.5.4 [Адаптивное заполнение], с. 188).

`C-x .` Установить префикс заполнения (`set-fill-prefix`).

`M-q` Заполнить абзац с текущим префиксом заполнения (`fill-paragraph`).

`M-x fill-individual-paragraphs`
: Заполнить область, рассматривая каждое изменение отступа как начало нового абзаца.

`M-x fill-nonuniform-paragraphs`
: Заполнить область, считая началом нового абзаца только строки-разделители абзацев.

Чтобы задать префикс заполнения, передвиньтесь к строке, которая начинается с желаемого префикса, поставьте точку в конец префикса и дайте команду `C-x .` (`set-fill-prefix`). После `C-x` стоит точка. Чтобы выключить префикс заполнения, определите пустой префикс: наберите `C-x .`, когда точка находится в начале строки.

Когда префикс заполнения в действии, команды заполнения уничтожают его в каждой строке перед заполнением и вставляют его в каждую строку после заполнения. Режим Auto Fill также автоматически вставляет в каждую вновь созданную строку префикс заполнения. Команда `C-o` вставляет в созданные ей строки префикс заполнения, когда вы используете ее в начале строки (см. Раздел 4.7 [Пустые строки], с. 39). С другой стороны, команда `M-^` уничтожает префикс (если он есть) после удаляемого перевода строки (см. Глава 20 [Отступы], с. 177).

Например, если `fill-column` равна 40 и вы установили префикс заполнения равным ‘`;;` ’, то `M-q` в таком тексте:

```
;; Это пример
;; абзаца внутри
;; комментария в стиле Лиспа.
```

дает следующее:

```
;; Это пример абзаца внутри комментария
;; в стиле Лиспа.
```

Строки, не начинающиеся с префикса заполнения, рассматриваются как начинающие абзац и в `M-q`, и в командах работы с абзацами; это дает хорошие результаты для абзацев с висящим отступом (все строки, кроме первой, имеют отступ). Строки, ставшие пустыми или имеющими отступ после удаления префикса, также разделяют или начинают абзац; это именно то, что вы хотите, если вы пишете комментарии, состоящие из нескольких абзацев, с ограничителем комментария на каждой строке.

Вы можете использовать `M-x fill-individual-paragraphs`, чтобы установить префикс заполнения для каждого абзаца автоматически. Эта команда делит область на абзацы, считая любое изменение величины отступа началом нового абзаца, и заполняет каждый из этих абзацев. Таким образом, все строки одного “абзаца” имеют одинаковый отступ. Именно этот отступ служит префиксом заполнения для каждого абзаца.

`M-x fill-nonuniform-paragraphs` — это похожая команда, которя делит область на абзацы другим способом. Она рассматривает только строки-разделители абзацев (как определено `paragraph-separate`) в качестве начинающих новый абзац. Поскольку это означает, что строки одного абзаца могут иметь разный отступ, в качестве префикса заполнения используется отступ наименьшего среди всех строк этого абзаца размера. Это дает хорошие результаты для стилей, в которых первая строка абзаца имеет больший или меньший отступ, чем остальная часть абзаца.

Префикс заполнения хранится в переменной `fill-prefix`. Ее значение — это либо строка, либо `nil`, когда префикса заполнения нет. В каждом буфере для этой переменной есть свое значение; ее изменение воздействует только на текущий буфер, но имеется и значение по умолчанию, которое вы также можете изменить. См. Раздел 31.2.4 [Локальные переменные], с. 350.

Свойство текста `indentation` предоставляет другой способ управления величиной отступа абзаца. См. Раздел 21.11.6 [Формат Отступ], с. 201.

### 21.5.4 Адаптивное заполнение

Команды заполнения могут в некоторых случаях автоматически вычислять подходящий для абзаца префикс заполнения: пропуски или определенная пунктуация в начале строки распространяются на все строки абзаца.

Если в абзаце есть две или более строки, префикс заполнения берется из второй, но только если он также появляется и в первой.

Если в абзаце есть только одна строка, команды заполнения *могут* взять префикс из этой строки. Здесь сложно принять решение, потому что в таком случае разумными могут оказаться три варианта:

- Использовать префикс первой строки для всех строк этого абзаца.
- Сделать в последующих строках отступ из пропусков таким образом, чтобы они выровнялись по тексту, следующему после префикса на первой, но не копировать в действительности префикс первой строки.
- Не предпринимать никаких особенных действий для второй и последующих строк.

Все три этих стиля форматирования применяются часто. Поэтому команды заполнения пытаются выяснить, какой бы вам понравился, основываясь на появляющемся префиксе и на основном режиме. Как это делается, описано ниже.

Если префикс, обнаруженный на первой строке, соответствует регулярному выражению `adaptive-fill-first-line-regexp`, или он оказался последовательностью, начинающей комментарий (это зависит от основнего режима), то для заполнения абзаца используется этот найденный префикс, при условии, что он не будет действовать как начало абзаца в следующих строках.

Иначе, найденный префикс преобразуется в эквивалентное число пробелов, и в качестве префикса заполнения для оставшихся строк используются эти пробелы, при условии, что они не будут действовать как начало абзаца в следующих строках.

В режиме Text и в других режимах, где абзацы разделяются только пустыми строками и переводами страницы, префикс, выбираемый адаптивным заполнением, никогда не ведет себя как начало абзаца, поэтому он всегда может использоваться для заполнения.

Переменная `adaptive-fill-regexp` определяет, какие виды начала строки могут служить префиксом заполнения: используются любые знаки в начале строки, соответствующие этому регулярному выражению. Если вы установите переменную `adaptive-fill-mode` равной `nil`, префикс заполнения никогда не выбирается автоматически.

Вы можете задать более сложные методы автоматического выбора префикса заполнения, установив переменную `adaptive-fill-function` в значение функции. Эта функция вызывается, когда точка находится с левого края строки, и она должна вернуть подходящий префикс заполнения. Если она возвращает `nil`, это означает, что она не увидела в этой строке префикс заполнения.

## 21.6 Команды преобразования регистра

В Emacs есть команды для перевода одиночных слов или любого произвольного текста в верхний или в нижний регистр.

`M-l` Перевести следующее слово в нижний регистр (`downcase-word`).

`M-u` Перевести следующее слово в верхний регистр (`upcase-word`).

`M-c` Сделать первую букву следующего слова заглавной, а остальные — строчными (`capitalize-word`).

`C-x C-l` Перевести область в нижний регистр (`downcase-region`).

`C-x C-u` Перевести область в верхний регистр (`upcase-region`).

Команды преобразования слов наиболее полезны. `M-l` (`downcase-word`) переводит слово после точки в нижний регистр, передвигая точку за него. Таким образом, повторение `M-l` переводит последующие слова. `M-u` (`upcase-word`) переводит все слово в прописные буквы, в то время как `M-c` (`capitalize-word`) ставит первую букву слова в верхнем регистре, а остальные — в нижнем регистре. Все эти команды переводят несколько слов за один раз, если им придать аргумент. Они особенно удобны для перевода большого объема текста, набранного полностью в верхнем регистре, в смешанный регистр, потому что вы можете двигаться по тексту, используя `M-l`, `M-u` или `M-c`, когда это необходимо, и используя иногда `M-f`, чтобы пропустить слово.

Когда задан отрицательный аргумент, команды перевода регистра в словах применяются к соответствующему числу слов перед точкой, не сдвигая ее саму. Это удобно, когда вы только что набрали слово в неправильном регистре: вы можете дать команду перевода регистра и продолжать набор.

Если команда перевода регистра в словах дается в середине слова, то она применяется только к части слова, которая следует за точкой. Это очень похоже на то, что делает `M-d` (`kill-word`). С отрицательным аргументом, перевод регистра применяется только к части слова перед точкой.

Другие команды перевода регистра — это `C-x C-u` (`upcase-region`) и `C-x C-l` (`downcase-region`), которые переводят все между точкой и меткой в заданный регистр. Точка и метка не сдвигаются.

Команды перевода регистра в области, `upcase-region` и `downcase-region`, обычно заблокированы. Это означает, что они запрашивают подтверждение, если вы пытаетесь их использовать. При подтверждении вы можете включить эти команды, тогда они больше не будут запрашивать подтверждения. См. Раздел 31.4.11 [Блокирование команды], с. 364.

## 21.7 Режим Text

Когда вы редактируете текстовые файлов на естественном языке, вам будет удобнее воспользоваться режимом Text, а не Fundamental. Чтобы войти в режим Text, наберите `M-x text-mode`.

В режиме Text абзацы разделяются только пустыми строками и разделителями страниц. В результате абзацы могут иметь отступ, и адаптивное заполнение может определить, какой отступ должен использоваться для заполнения абзаца. См. Раздел 21.5.4 [Адаптивное заполнение], с. 188.

В режиме Text ⟨TAB⟩ запускает функцию `indent-relative` (см. Глава 20 [Отступы], с. 177), чтобы вам было удобно делать отступ как в предыдущей строке. Когда в предыдущей строке нет отступа, `indent-relative` запускает `tab-to-tab-stop`, которая использует устанавливаемые вами позиции табуляции (см. Раздел 20.2 [Позиции табуляции], с. 178).

Режим Text выключает средства, связанные с комментариями, кроме тех случаев, когда вы явно вызовете их. Он изменяет синтаксическую таблицу таким образом, что точки не рассматриваются как часть слова, тогда как знак забоя, подчеркивание и апострофы считаются таковыми.

Если вы делаете отступ в первой строке абзаца, вам нужно использовать режим Paragraph-Indent Text вместо режима Text. В этом режиме вам не обязательно ставить между абзацами пустые строки, потому что отступа в первой строке достаточно для начала нового абзаца; однако, абзацы, в которых каждая строка имеет отступ, не поддерживаются. Чтобы войти в этот режим, используйте `M-x paragraph-indent-text-mode`.

Режим Text и все режимы, основанные на нем, определяют `M-`⟨TAB⟩ как команду `ispell-complete-word`, которая производит завершение части слова перед точкой в данном буфере, используя орфографический словарь как пространство возможных слов. См. Раздел 13.4 [Правописание], с. 102.

Вход в режим Text запускает ловушку `text-mode-hook`. Другие основные режимы, родственные с режимом Text, также запускают эту ловушку и потом свои ловушки; к ним относятся режим Paragraph-Indent Text, режим Nroff, режим TEX, режим Outline и режим Mail. Функции ловушки `text-mode-hook` могут проверить значение `major-mode`, чтобы узнать, в какой из этих режимов вы на самом деле входите. См. Раздел 31.2.3 [Ловушки], с. 349.

## 21.8 Режим Outline

Режим Outline — это основной режим, очень похожий на режим Text, но предназначенный для редактирования структурированного текста. Он позволяет вам делать части текста временно невидимыми, так что вы можете видеть просто просмотреть структуру текста. Наберите `M-x outline-mode`, чтобы включить режим Outline в текущем буфере.

Когда режим Outline делает строку невидимой, эта строка не появляется на экране. Экран имеет точно такой же вид, как если бы невидимая строка была удалена, за исключением того, что в конце предыдущей видимой строки появляется многоточие (только одно, независимо от того, сколько невидимых строк следует дальше).

Команды редактирования, работающие со строками, такие как `C-n` и `C-p`, трактуют текст невидимой строки как часть предыдущей видимой. Уничтожение полной видимой строки, включая ограничивающий ее знак новой строки, на самом деле уничтожает вместе с ней все следующие невидимые строки.

Второстепенный режим Outline предоставляет те же команды, что и основной режим Outline, но вы можете использовать его совместно с другими основными режимами. Чтобы включить второстепенный режим Outline в текущем буфере, наберите `M-x outline-minor-mode`. Вы также можете указать это в тексте файла с помощью локальной переменной в форме ‘`mode: outline-minor`’ (см. Раздел 31.2.5 [Переменные файла], с. 351).

Основной режим, режим Outline, предоставляет особые привязки ключей на префиксе `C-c`. Второстепенный режим Outline предоставляет похожие привязки с `C-c @` в качестве префикса; это нужно, чтобы уменьшить риск конфликта со специальными командами основного режима. (Используемый префикс управляется переменной `outline-minor-mode-prefix`.)

При входе в режим Outline запускается ловушка `text-mode-hook` сразу после ловушки `outline-mode-hook` (см. Раздел 31.2.3 [Ловушки], с. 349).

### 21.8.1 Формат схем текста

Режим Outline предполагает, что строки в буфере делятся на два типа: *строки заголовка* и *строки тела*. Строки заголовка представляет тему в схеме текста. Они начинаются с одной или более звездочек; число звездочек определяет глубину заголовка в структуре текста. Таким образом, строка заголовка с одной звездочкой — это основная тема; все строки заголовка с двумя звездочками между этой строкой и следующей строкой заголовка с одной звездочкой являются ее подтемами и так далее. Любая строка, которая не является строкой заголовка, — это строка тела. Строки тела относятся к предшествующей строке заголовка. Вот пример:

```
* Еда
Это тело, которое
говорит что-то о еде.

** Вкусная еда
Это тело заголовка второго уровня.

** Противная еда
Здесь тоже могло бы
быть тело на
нескольких строках.

*** Общепит

* Приют
Еще одна тема первого уровня со своей строкой заголовка.
```

Строка заголовка вместе со всеми последующими строками тела в совокупности называются *вхождением*. Строка заголовка вместе со всеми следующими более глубокими заголовками и их строками тела называется *поддеревом*.

Вы можете настроить критерий для различения строк заголовка, установив переменную `outline-regexp`. Любая строка, чье начало содержит совпадение с этим регулярным выражением, рассматривается как строка заголовка. Соответствия, которые начинаются с середины строки (не в начале), не рассматриваются. Длина текста соответствия определяет уровень заголовка: более длинное соответствие создает глубже вложенный уровень. Например, если программа форматирования имеет команды ‘`@chapter`’, ‘`@section`’ и ‘`@subsection`’ для деления документа на главы и разделы, вы можете сделать эти строки воспринимаемыми в качестве строк заголовка, установив `outline-regexp` равной ‘`"@chap\\|@\\(sub\\)*section"`’. Обратите внимание на хитрость: слова ‘`chapter`’ и ‘`section`’ имеют равную длину, но определив регулярное выражение как совпадающее только с ‘`chap`’, мы гарантируем, что длина текста, соответствующего заголовку главы, будет короче; таким образом, режим Outline будет знать, что разделы содержатся в главах. Это работает, если никакая другая команда не начинается с ‘`@chap`’.

Есть возможность изменить правило подсчета уровня строк заголовка, путем установки переменной `outline-level`. Значение `outline-level` должно быть функцией, не принима-

ющей аргументов и возвращающей номер уровня текущего заголовка. Некоторые основные режимы, например режимы C, Nroff и Emacs Lisp, устанавливают эту переменную, чтобы ими можно было пользоваться со второстепенным режимом Outline.

### 21.8.2 Команды перемещения по структуре

Режим Outline предоставляет особые команды перемещения, которые передвигают назад и вперед по строкам заголовков.

`C-c C-n` Передвинуть точку к следующей видимой строке заголовка (`outline-next-visible-heading`).

`C-c C-p` Передвинуть точку к предыдущей видимой строке заголовка (`outline-previous-visible-heading`).

`C-c C-f` Передвинуть точку к следующей видимой строке заголовка того же уровня, что и строка, на которой находится точка (`outline-forward-same-level`).

`C-c C-b` Передвинуть точку к предыдущей видимой строке заголовка этого же уровня (`outline-backward-same-level`).

`C-c C-u` Передвинуть точку назад к видимой строке заголовка более низкого уровня (`outline-up-heading`).

`C-c C-n` (`outline-next-visible-heading`) переходит вниз на следующую строку заголовка. `C-c C-p` (`outline-previous-visible-heading`) передвигает аналогично, но назад. Обе принимают числовой аргумент как счетчик повторов. Имена этих команд подчеркивают, что невидимые заголовки пропускаются, но это на самом деле не специальная особенность. Все команды редактирования, которые просматривают строки, игнорируют невидимые строки автоматически.

Более мощные команды движения понимают уровневую структуру заголовков. `C-c C-f` (`outline-forward-same-level`) и `C-c C-b` (`outline-backward-same-level`) передвигают от одной строки заголовка к другой видимой строке заголовка той же самой глубины в структуре. `C-c C-u` (`outline-up-heading`) передвигает назад к другому заголовку, который имеет меньшую глубину вложенности.

### 21.8.3 Команды управления видимостью структуры

Чтобы сделать строки видимыми или невидимыми, используются другие специальные команды режима Outline. Все их имена начинаются либо с `hide`, либо с `show`. Большинство из них составляют пары противоположностей. Они не могут быть отменены; вместо этого вы можете произвести отмену безотносительно к видимости текста. Изменение видимости строк просто не записывается механизмом отмены.

`C-c C-t` Сделать все строки тела в буфере невидимыми (`hide-body`).

`C-c C-a` Сделать все строки в буфере видимыми (`show-all`).

`C-c C-d` Сделать все под этим заголовком невидимым, но не сам этот заголовок (`hide-subtree`).

`C-c C-s` Сделать все под этим заголовком видимым, включая тело, подзаголовки и их тела (`show-subtree`).

`C-c C-l` Сделать тело этой строки заголовка и все его подзаголовки невидимыми (`hide-leaves`).

`C-c C-k` Сделать все подзаголовки этого заголовка видимыми на всех уровнях (`show-branches`).

`C-c C-i` Сделать непосредственные подзаголовки (на один уровень вниз) этого заголовка видимыми (`show-children`).

`C-c C-c` Сделать тело этого заголовка невидимым (`hide-entry`).

`C-c C-e` Сделать тело этого заголовка видимым (`show-entry`).

`C-c C-q` Скрыть все, кроме *n* верхних уровней строк заголовков (`hide-sublevels`).

`C-c C-o` Скрыть все, кроме заголовка или тела, в котором находится точка, и заголовков, ведущих отсюда к верхнему уровню структуры (`hide-other`).

Две команды, которые строго противоположны, — это `C-c C-c` (`hide-entry`) и `C-c C-e` (`show-entry`). Они применяются, когда точка расположена на заголовке, и относятся только к строкам тела этого заголовка. Подтемы и их тела не затрагиваются.

Две более мощные противоположности — это `C-c C-d` (`hide-subtree`) и `C-c C-s` (`show-subtree`). Обе предполагают использование, когда точка находится на заголовке, и обе применяются ко всем строкам *поддерева* этого заголовка: его телу, всем его подзаголовкам, как прямым, так и косвенным, и всем их телам. Другими словами, поддерево содержит все, что следует за этим заголовком, вплоть до (но не включая) следующего заголовка того же самого или более высокого ранга.

Промежуточное состояние между видимым и невидимым поддеревом — это когда видимы все подзаголовки, но не видимо ни одно тело. Для осуществления этого есть две команды, в зависимости от того, хотите ли вы скрыть тела или сделать видимыми подзаголовки. Это `C-c C-l` (`hide-leaves`) и `C-c C-k` (`show-branches`).

Команда `C-c C-i` (`show-children`) немного слабее `show-branches`. Она делает видимыми только непосредственные подзаголовки — те, что на один уровень ниже. Более глубокие подзаголовки остаются невидимыми, если они были таковыми.

Две команды производят действие, охватывающее весь файл. `C-c C-t` (`hide-body`) делает все строки тела невидимыми, так что вы видите просто схему текста. `C-c C-a` (`show-all`) делает все строки видимыми. Эти команды могут рассматриваться как пара противоположных, хотя `C-c C-a` применяется не только к строкам тела.

Команда `C-c C-q` (`hide-sublevels`) скрывает все заголовки, кроме заголовков верхнего уровня. С числовым аргументом *n*, она скрывает все, кроме строк заголовков *n* верхних уровней.

Команда `C-c C-o` (`hide-other`) скрывает все, кроме заголовка или текста тела, в котором находится точка, и их родителей (заголовков, ведущих отсюда к верхнему уровню структуры).

Использование многоточий в конце видимых строк может быть отключено путем установки `selective-display-ellipses` равной `nil`. Тогда не будет явного указания на существование невидимых строк.

Когда наращиваемый поиск находит текст, который скрыт режимом Outline, он делает эту часть буфера видимой. Если вы выйдите из поиска в этой позиции, текст останется видимым.

### 21.8.4 Просмотр одной схемы в нескольких видах

Вы можете просмотреть два вида одной схемы одновременно в разных окнах. Чтобы сделать так, вы должны создать косвенный буфер, используя `M-x make-indirect-buffer`. Первый аргумент этой команды — это имя существующего буфера Outline, а второй аргумент — это имя, которое будет использоваться для нового косвенного буфера. См. Раздел 15.6 [Косвенные буферы], с. 139.

Когда косвенный буфер создан, вы можете показать его в окне, как обычно, с помощью `C-x 4 b` или других команд Emacs. Команды режима Outline для показа или скрывания

частей текста действуют в каждом буфере независимо; в результате каждый буфер может иметь свой вид. Если вы хотите получить более двух видов одной и той же схемы, создайте дополнительные косвенные буферы.

## 21.9 Режим TEX

TEX — это мощная программа компьютерного набора, написанная Дональдом Кнутом. Он также является свободным программным продуктом, как и GNU Emacs. LaTEX — это упрощенный формат ввода для TEX, реализованный на макросах TEX. Он распространяется вместе с TEX. SliTEX — это особая форма LaTEX.

В Emacs есть специальный режим TEX для редактирования входных TEX-файлов. Он предусматривает средства для проверки сбалансированности ограничителей и для вызова TEX для всего файла или его части.

Режим TEX имеет три варианта: режим Plain TEX, режим LaTEX и режим SliTEX (три этих основных режима отличающихся друг от друга лишь слегка). Они предназначены для редактирования трех различных входных форматов. Команда `M-x tex-mode` проверяет содержимое буфера, чтобы определить, не является ли это входом для LaTEX или SliTEX; если это так, она выбирает подходящий режим. Если содержимое файла не оказалось ни LaTEX, ни SliTEX, она выбирает режим TEX. Если содержимого файла оказалось недостаточно для определения формата, то используется режим, задаваемый переменной `tex-default-mode`.

Когда `M-x tex-mode` делает неправильное предположение, вы можете использовать команды `M-x plain-tex-mode`, `M-x latex-mode` и `M-x slitex-mode` для явного выбора конкретного варианта режима TEX.

### 21.9.1 Команды редактирования режима TEX

Здесь перечислены специальные команды, предусмотренные в режиме TEX для редактирования текста файла.

`"` Вставить согласно контексту либо ‘■’, либо ‘"’, либо ‘■’ (`tex-insert-quote`).

`C-j` Вставить разрыв абзаца (два перевода строки) и проверить предыдущий абзац на несбалансированные фигурные скобки или знаки доллара (`tex-terminate-paragraph`).

`M-x tex-validate-region`
Проверить каждый абзац в буфере на несбалансированные фигурные скобки или знаки доллара.

`C-c {` Вставить ‘{}’ и расположить точку между ними (`tex-insert-braces`).

`C-c }` Перейти вперед за следующую непарную закрывающую фигурную скобку (`up-list`).

Знак ‘"’ обычно не используется в TEX; мы используем ‘■’, чтобы открыть кавычки, и ‘■’, чтобы закрыть. Чтобы облегчить редактирование с учетом этого соглашения о форматировании, режим TEX заменяет обычное значение клавиши `"` на команду, вставляющую пару одиночных простых или обратных кавычек (`tex-insert-quote`). Если говорить точно, эта команда вставляет ‘■’ после пропуска или открывающей фигурной скобки, ‘"’ после обратной косой черты и ‘■’ после всех остальных знаков.

Если вам нужен знак ‘"’ сам по себе в необычном контексте, используйте для его вставки `C-q`. Также, `"` с числовым аргументом всегда вставляет указанное число знаков ‘"’. Вы можете выключить средство раскрытия `"`, убрав эту привязку из локальной раскладки (см. Раздел 31.4 [Привязки ключей], с. 356).

Знак ‘$’ имеет в режиме TEX особый синтаксический код, который перетендует на понимание способа, которым ограничители математической моды TEX соответствуют друг другу. Когда вы вводите ‘$’, который используется для выхода из математической моды, на секунду отображается позиция парного ‘$’, который вводил в математическую моду. Это то же самое средство, которое показывает открывающую фигурную скобку, соответствующую вставленной закрывающей. Однако, нет способа узнать, является ли ‘$’ входом или выходом из математической моды; поэтому когда вы вводите ‘$’, который входит в математическую моду, показывается позиция предыдущего ‘$’, как если бы она была они составляли пару, даже если фактически они не относятся друг к другу.

TEX использует фигурные скобки как ограничители, которые обязаны составлять пары. Некоторые пользователи предпочитают поддерживать фигурные скобки все время сбалансированными, а не вставлять их по отдельности. Используйте `C-c {` (`tex-insert-braces`), чтобы вставить пару фигурных скобок. Эта команда оставляет точку между двумя этими скобками, чтобы вы могли вставить текст внутрь. Потом используйте команду `C-c }` (`up-list`), чтобы перейти вперед через закрывающую фигурную скобку.

Существуют две команды для контроля соответствия фигурных скобок. `C-j` (`tex-terminate-paragraph`) проверяет абзац перед точкой и вставляет два ограничителя новой строки для начала нового абзаца. Если будет найдено какое-то несоответствие, она напечатает сообщение в эхо-области. `M-x tex-validate-region` проверяет область, абзац за абзацем. Ошибки перечисляются в буфере ‘`*Occur*`’, и вы можете использовать в нем `C-c C-c` или `Mouse-2`, чтобы перейти к конкретному несоответствию.

Заметьте, что команды Emacs подсчитывают в режиме TEX не только фигурные скобки, но и квадратные и круглые. Для проверки синтаксиса TEX это не совсем корректно. Тем не менее, круглые и квадратные скобки, скорее всего, используются в тексте в качестве парных разделителей, и будет полезно, если различные команды движения и автоматического показа пар будут с ними работать.

### 21.9.2 Команды редактирования режима LaTEX

Режим LaTEX и его вариация, режим SliTEX, предоставляют несколько дополнительных возможностей, не относящихся к plain TEX.

`C-c C-o` Вставляет ‘`\begin`’ и ‘`\end`’ для блока LaTEX и помещает точку на строке между ними (`tex-latex-block`).

`C-c C-e` Закрывает самый внутренний еще не закрытый блок LaTEX (`tex-close-latex-block`).

В LaTEX для группировки блоков текста используются команды ‘`\begin`’ и ‘`\end`’. Чтобы вставить ‘`\begin`’ и парную ‘`\end`’ (на новой строке после ‘`\begin`’), используйте `C-c C-o` (`tex-latex-block`). Между двумя этими строками вставляется пустая строка, и на ней оставляется точка. При вводе типа блока вы можете использовать завершение; чтобы задать имена дополнительных типов блоков, установите переменную `latex-block-names`. Например, добавить ‘`theorem`’, ‘`corollary`’ и ‘`proof`’ можно таким образом:

```
(setq latex-block-names '("theorem" "corollary" "proof"))
```

Во входном тексте LaTEX команды ‘`\begin`’ и ‘`\end`’ должны соответствовать друг другу. Вы можете использовать `C-c C-e` (`tex-close-latex-block`), чтобы автоматически вставить ‘`\end`’, соответствующую последней ‘`\begin`’, оставшей без пары. Эта команда делает для ‘`\end`’ отступ в соответствии с ее ‘`\begin`’. Если точка находится в начале строки, она вставляет после ‘`\end`’ новую строку,

### 21.9.3 Команды печати для TEX

Вы можете вызвать TEX как подчиненный процесс Emacs либо для всего содержимого буфера, либо только на область, за один раз. Запуск TEX таким способом только в одной

главе дает удобный метод увидеть, как выглядят ваши изменения, не тратя время на форматирование всего файла.

`C-c C-r` Вызвать TEX для текущей области вместе с заголовоком буфера (`tex-region`).

`C-c C-b` Вызывать TEX для всего текущего буфера (`tex-buffer`).

`C-c ⟨TAB⟩` Вызывать BibTEX для текущего файла (`tex-bibtex-file`).

`C-c C-f` Вызывать TEX для текущего файла (`tex-file`).

`C-c C-l` Переместить центр окна, показывающего вывод подчиненного TEX, чтобы можно было увидеть последнюю строку (`tex-recenter-output-buffer`).

`C-c C-k` Уничтожить подпроцесс TEX (`tex-kill-job`).

`C-c C-p` Печатать вывод из последней команды `C-c C-r`, `C-c C-b` или `C-c C-f` (`tex-print`).

`C-c C-v` Запустить предварительный просмотр вывода последней команды `C-c C-r`, `C-c C-b` или `C-c C-f` (`tex-view`).

`C-c C-q` Показать очередь принтера (`tex-show-print-queue`).

Вы можете пропустить текущий буфер через подчиненный TEX с помощью `C-c C-b` (`tex-buffer`). Отформатированный вывод появляется во временном файле; чтобы напечатать его, наберите `C-c C-p` (`tex-print`). Потом вы можете использовать `C-c C-q` (`tex-show-printer-queue`), чтобы увидеть, как скоро ваш вывод будет напечатан. Если ваш терминал может показывать выходные файлы TEX, вы можете просмотреть вывод на терминале с помощью команды `C-c C-v` (`tex-view`).

Вы можете указать каталог для запуска TEX, установив переменную `tex-directory`. Значением по умолчанию является `"."`. Если переменная среды `TEXINPUTS` содержит относительные имена каталогов, или ваши файлы содержат команды '`\input`' с относительными именами, то `tex-directory` *должна* быть равна `"."`, или вы получите неправильные результаты. В противном случае, можно без опасения задать какой-то другой каталог, например, `"/tmp"`.

Если вы хотите указать, какие команды оболочки нужно использовать в подчиненном процессе TEX, вы можете сделать это установкой значений переменных `tex-run-command`, `latex-run-command`, `slitex-run-command`, `tex-dvi-print-command`, `tex-dvi-view-command` и `tex-show-queue-command`. Вы *обязаны* установить значение `tex-dvi-view-command` для вашего конкретного терминала; эта переменная не имеет значения по умолчанию. Другие переменные имеют значения по умолчанию, которые могут подойти (а могут и не подойти) для вашей системы.

Обычно имя файла, передаваемое этим командам, пишется в конце командной строки: например, '`latex` *имя-файла*'. Однако в некоторых случаях имя файла должно быть вставлено в команду; это может понадобиться, к примеру, когда вам нужно предоставить имя файла в качестве аргумента команде, чей вывод направляется другой программе. Вы можете указать, в какое место следует подставить имя файла, с помощью знака '`*`' в командной строке. Например,

```
(setq tex-dvi-print-command "dvips -f * | lpr")
```

Терминальный вывод TEX, включающий все сообщения об ошибках, появляется в буфере с именем '`*tex-shell*`'. Если TEX получил ошибку, вы можете переключиться в этот буфер и подать ему какой-то ввод (это работает как в режиме Shell, см. Раздел 30.2.2 [Интерактивная оболочка], с. 324). Без переключения в этот буфер, вы можете прокрутить его с помощью `C-c C-l` так, что последняя строчка в нем станет видимой.

Наберите `C-c C-k` (`tex-kill-job`), чтобы уничтожить процесс TEX, если вы понимаете, что его вывод уже бесполезен. Использование `C-c C-b` или `C-c C-r` также уничтожает любой работающий процесс TEX.

Вы также можете пропустить произвольную область через подчиненный TEX, набрав `C-c C-r` (`tex-region`). Однако, это ненадежно, потому что большинство входных файлов TEX содержат в начале команды, устанавливающие какие-то параметры и определяющие макросы, без которых дальнейшая часть файла не отформатируется правильно. Для того, чтобы решить эту проблему, `C-c C-r` позволяет вам обозначить часть файла как содержащую важные команды; она вставляется перед заданной областью как часть ввода TEX. Обозначенная часть файла называется *заголовком*.

Чтобы обозначить границы заголовка в режиме Plain TEX, вы вставляете в файл две специальные строки. Вставьте '`%**start of header`' перед заголовком и '`%*end of header`' после него. Обе должны появиться полностью на одной строке, но перед ними или после них допускается другой текст. Строки, содержащие эти фразы, включаются в заголовок. Если '`%**start of header`' не появится в пределах первых 100 строк буфера, `C-c C-r` предполагает, что заголовка нет.

В режиме LaTEX заголовок начинается с команды '`\documentstyle`' и заканчивается командой '`\begin{document}`'. LaTEX требует, чтобы вы использовали эти команды в любом случае, так что для определения заголовка не требуется делать ничего особенного.

Команды (`tex-buffer`) и (`tex-region`) делают свою работу во временном каталоге, и им недоступны вспомогательные файлы, нужные TEX для перекрестных ссылок; эти команды в общем случае не подходят для обработки окончательной копии, в которой все перекрестные ссылки должны быть правильными.

Когда вам нужны вспомогательные файлы для перекрестных ссылок, используйте `C-c C-f` (`tex-file`), которая запускает TEX для файла текущего буфера в каталоге этого файла. Перед запуском TEX она предлагает сохранить все измененные буферы. В общем случае, вы должны использовать (`tex-file`) дважды, чтобы получить правильные перекрестные ссылки.

Значение переменной `tex-start-options-string` задает ключи для запуска TEX. Значение по умолчанию велит TEX работать в безостановочном режиме. Чтобы запустить TEX интерактивно, установите эту переменную равной `""`.

Большие документы TEX часто разбивают на несколько файлов — один главный плюс подфайлы. Запуск TEX для подфайла как правило не сработает; вы должны запускать его для главного файла. Чтобы сделать `tex-file` полезной при редактировании подфайла, вы можете установить переменную `tex-main-file` равной имени главного файла. Тогда `tex-file` запустит TEX для этого файла.

Наиболее удобный способ использования `tex-main-file` — указать ее в перечне локальных переменных в каждом из подфайлов. См. Раздел 31.2.5 [Переменные файла], с. 351.

С LaTEX-файлами вы можете использовать BibTEX, чтобы обработать вспомогательные файлы для файла текущего буфера. BibTEX находит библиографические цитаты в базе данных и подготавливает процитированные ссылки для раздела библиографии. Команда `C-c TAB` (`tex-bibtex-file`) запускает команду оболочки (`tex-bibtex-command`), чтобы получить '`.bbl`'-файл для файла текущего буфера. Вообще говоря, вам нужно сначала сделать `C-c C-f` (`tex-file`), чтобы получить '`.aux`'-файл, затем сделать `C-c TAB` (`tex-bibtex-file`) и после этого повторить `C-c C-f` (`tex-file`) еще раз, чтобы сгенерировать правильные перекрестные ссылки.

При входе в любую разновидность режима TEX запускаюся ловушки `text-mode-hook` и `tex-mode-hook`. Затем запускаются `plain-tex-mode-hook` или `latex-mode-hook`, что подходит. Для SliTEX-файлов запускается ловушка `slitex-mode-hook`. При старте оболочки TEX запускается `tex-shell-hook`. См. Раздел 31.2.3 [Ловушки], с. 349.

## 21.10 Режим Nroff

Режим Nroff — это режим, похожий на режим Text, но модифицированный для управления командами nroff, присутствующими в тексте. Вызовите `M-x nroff-mode`, чтобы войти

в этот режим. Он отличается от режима Text только несколькими возможностями. Все строки команд nroff считаются разделителем абзацев, так что заполнение никогда не исказит команды nroff. Страницы разделяются командами '`.bp`'. Комментарии начинаются с обратной косой черты и двойных кавычек. Также предусмотрены три специальные команды, которых нет в режиме Text:

`M-n` Перейти на начало следующей строки, которая не является командой nroff (`forward-text-line`). Аргумент служит счетчиком повторов.

`M-p` Похожа на `M-n`, но сдвигает вверх (`backward-text-line`).

`M-?` Напечатать в эхо-области число текстовых строк (строк, которые не являются командами nroff) в текущей области (`count-text-lines`).

Другое свойство режима Nroff — это то, что вы можете включать режим Electric Nroff. Это второстепенный режим, который вы можете включать или выключать при помощи `M-x electric-nroff-mode` (см. Раздел 31.1 [Второстепенные режимы], с. 341). Если этот режим включен, то каждый раз, когда вы набираете ⟨RET⟩ для окончания строки, которая содержит команду nroff, открывающую некоторый вид группы, в следующую строку автоматически вставляется соответствующая закрывающая группу команда nroff. Например, если вы находитесь в начале строки и наберете `. ( b` ⟨RET⟩, то в новую строку, следующую за точкой, будет вставлена соответствующая команда '`.)b`'.

Если с режимом Nroff вы используете второстепенный режим Outline (см. Раздел 21.8 [Режим Outline], с. 190), строками заголовков будут строки вида '`.H`' с последующим числом (уровнем заголовка).

Вход в режим Nroff запускает ловушку `text-mode-hook`, а затем ловушку `nroff-mode-hook` (см. Раздел 31.2.3 [Ловушки], с. 349).

## 21.11 Редактирование форматированного текста

*Режим Enriched* — это второстепенный режим для редактирования файлов, которые содержат форматированный текст в стиле WYSIWYG, как в текстовом процессоре. На данный момент форматированный текст в режиме Enriched может задавать шрифты, цвета, подчеркивание, поля и типы заполнения и выравнивания. В будущем мы планируем реализовать также и другие возможности для форматирования.

Режим Enriched — это второстепенный режим (см. Раздел 31.1 [Второстепенные режимы], с. 341). Как правило он используется вместе с режимом Text (см. Раздел 21.7 [Режим Text], с. 190). Однако, вы можете также использовать его и с другими основными режимами, такими как режим Outline и режим Paragraph-Indent Text.

Потенциально Emacs может сохранять файлы с форматированным текстом во многих форматах. На текущий момент реализован только один формат: *text/enriched*, который определяется протоколом MIME. См. раздел "Format Conversion" в *the Emacs Lisp Reference Manual*, для получения подробностей о том, как Emacs распознает и пребразует форматы файлов.

Дистрибутив Emacs содержит файл с форматированным текстом, который может служить примером. Он называется '`etc/enriched.doc`'. Этот файл содержит образцы, иллюстрирующие все возможности, описанные в этом разделе. В нем также есть перечень идей для будущих улучшений.

### 21.11.1 Запрос на редактирование форматированного текста

Когда вы обращаетесь к файлу, который был сохранен в формате text/enriched, Emacs автоматически преобразует информацию о форматировании из этого файла во внутренний формат Emacs (свойства текста) и включает режим Enriched.

Чтобы создать новый файл с форматированным текстом, обратитесь сначала к несуществующему файлу, а перед тем как начать редактирование наберите `M-x enriched-mode`. Эта команда включает режим Enriched. Делайте это до того, как вы начнете вставлять текст, чтобы вставляемый текст наверняка обрабатывался правильно.

В более общем виде, команда `enriched-mode` включает режим Enriched, если он был выключен, и выключает его, если он был включен. Запущенная с числовым аргументом, эта команда включает режим Enriched, если аргумент положителен, и выключает в противном случае.

Когда вы сохраняете буфер при задействованном режиме Enriched, Emacs автоматически преобразует текст к формату text/enriched во время записи в файл. Когда вы снова обратитесь к этому файлу, Emacs автоматически распознает формат, преобразует текст обратно и снова включит режим Enriched.

Обычно после обращения к файлу в формате text/enriched, Emacs перезаполняет каждый абзац так, чтобы он умещался по заданному правому полю. Вы можете выключить это перезаполнение, чтобы сэкономить время, установив переменную `enriched-fill-after-visiting` в значение `nil` или `ask`.

Однако, при обращении к файлу, записанному в формате Enriched, нет нужды в перезаполнении, поскольку Emacs сохраняет установки правого поля вместе с текстом.

Делая добавления к `enriched-translations`, вы можете вносить пометки для сохранения дополнительных свойств текста, которые Emacs обычно не сохраняет. Заметьте, что стандарт text/enriched требует, чтобы имена всех нестандартных пометок начинались с ‘`x-`’, например ‘`x-read-only`’. Это позволяет быть уверенным в том, что они не будут конфликтовать со стандартными пометками, добавленными позже.

### 21.11.2 Жесткие и гибкие переводы строк

Emacs различает в форматированном тексте два разных вида переводов строк: *жесткие* и *гибкие*.

Жесткие переводы строк используются для разделения абзацев, или пунктов перечня, или везде, где строка должна всегда разрываться вне зависимости от полей. Команды ⟨RET⟩ (`newline`) и `C-o` (`open-line`) вставляют жесткие переводы строк.

Гибкие переводы строк применяются для того, чтобы уместить текст в пределы полей. Все команды заполнения, включая Auto Fill, вставляют гибкие переводы строк, и они удаляют всегда только гибкие переводы строк.

Хотя жесткие и гибкие переводы строк выглядят одинаково, важно помнить об их различии. Не используйте ⟨RET⟩, чтобы разорвать строку в середине заполненного абзаца, или иначе вы получите жесткие переводы строк, которые послужат барьером последующему заполнению. Вместо этого позвольте разбивать строки режиму Auto Fill, чтобы при изменении текста или полей Emacs мог правильно перезаполнить строки. См. Раздел 21.5.1 [Auto Fill], с. 185.

С другой стороны, в таблицах и перечнях, где строки должны всегда оставаться такими, как вы их набрали, вы можете использовать для завершения строк ⟨RET⟩. Для таких строк вы также можете установить стиль выравнивания в `unfilled`. См. Раздел 21.11.7 [Формат Выравнивание], с. 202.

### 21.11.3 Редактирование информации о формате

Есть два способа изменить информацию о формате для файла с форматированным текстом: командами клавиатуры или с помощью мыши.

Простейший способ добавить свойства к вашему документу — воспользоваться меню Text Properties. Вы можете попасть в это меню двумя путями: из меню Edit в полоске

меню или с помощью `C-mouse-2` (прижмите клавишу ⟨CTRL⟩ и нажмите среднюю кнопку мыши).

Большинство пунктов из меню Text Properties ведут к другим подменю. Подменю описаны в последующих разделах. Некоторые пункты запускают команды непосредственно:

`Remove Properties`
: Удаляет из области все свойства текста, с которыми работает меню Text Properties (`facemenu-remove-props`).

`Remove All`
: Удаляет *все* свойства текста из области (`facemenu-remove-all`).

`List Properties`
: Перечисляет все свойства текста для знака после точки (`list-text-properties-at`).

`Display Faces`
: Показывает перечень всех определенных начертаний.

`Display Colors`
: Показывает перечень всех определенных цветов.

### 21.11.4 Начертания в форматированном тексте

В подменю Faces перечислены разные начертания Emacs, включая `bold`, `italic` и `underline`. Выбор одного из них добавляет это начертание к области. См. Раздел 17.13 [Начертания], с. 155. Вы также можете задать начертания с помощью таких команд клавиатуры:

`M-g d`
: Говорит, что область или следующий вставленный знак должны появиться в начертании `default` (`facemenu-set-default`).

`M-g b`
: Говорит, что область или следующий вставленный знак должны появиться в начертании `bold` (`facemenu-set-bold`).

`M-g i`
: Говорит, что область или следующий вставленный знак должны появиться в начертании `italic` (`facemenu-set-italic`).

`M-g l`
: Говорит, что область или следующий вставленный знак должны появиться в начертании `bold-italic` (`facemenu-set-bold-italic`).

`M-g u`
: Говорит, что область или следующий вставленный знак должны появиться в начертании `underline` (`facemenu-set-underline`).

`M-g o` *начертание* ⟨RET⟩
: Говорит, что область или следующий вставленный знак должны появиться в заданном *начертании* (`facemenu-set-face`).

Если вы используете эти команды с префиксным аргументом — или, в режиме Transient Mark, если область не активна — то они задают начертание для следующего самовставляющегося ввода. См. Раздел 8.2 [Transient Mark], с. 64. Это относится как к командам клавиатуры, так и к командам меню.

Режим Enriched определяет два дополнительных начертания: `fixed` и `excerpt`. Они соответствуют кодам, используемым в формате файлов text/enriched.

Начертание `excerpt` предназначено для цитат. Оно совпадает с начертанием `italic`, если вы его не перенастроили (см. Раздел 31.2.2.3 [Настройка начертаний], с. 347).

Начертание `fixed` означает “Использовать для этой части текста равноширинный шрифт”. В настоящее время Emacs поддерживает только равноширинные шрифты; следовательно, пометка `fixed` пока не так необходима. Однако, в будущих версиях Emacs

мы планируем реализовать поддержку шрифтов переменной ширины, и другие системы, способные отображать формат text/enriched, могут не использовать по умолчанию равноширинный шрифт. Поэтому если вы хотите, чтобы какая-то часть текста появлялась именно с равноширинным шрифтом, вам следут задать для этой части начертание `fixed`.

Начертание `fixed` обычно определено так, что для него используется другой шрифт, отличающийся от шрифта по умолчанию. Однако, на разных системах установлены разные шрифты, поэтому вам может понадобиться настроить это.

Если ваш терминал не умеет отображать разные начертания, у вас не получится их увидеть, но вы все же сможете редактировать документы, содержащие их. Вы даже сможете добавить в текст начертания и цвета. Они станут видимы, когда файл будут просматривать на терминале, который способен их отобразить.

### 21.11.5 Цвета в форматированном тексте

Вы можете указать цвета букв и фона для фрагментов текста. Есть меню для задания цвета текста и меню для задания цвета фона. Оба меню цветов перечисляют все цвета, которые вы использовали в режиме Enriched в текущем сеансе Emacs.

Если вы задаете цвет с префиксным аргументом — или, в режиме Transient Mark, если область не активна — то этот цвет применяется для самовставляемого ввода. См. Раздел 8.2 [Transient Mark], с. 64. В противном случае эта команда относится к области.

Оба меню цветов содержат дополнительный пункт: ‘`Other`’. Вы можете использовать этот пункт для задания цвета, который не перечислен в меню; имя цвета считывается в минибуфере. Чтобы просмотреть перечень доступных цветов и их имена, используйте пункт ‘`Display Colors`’ в меню Text Properties (см. Раздел 21.11.3 [Редактирование формата], с. 199).

Любой цвет, заданный таким способом или упомянутый в считанном файле с форматированным текстом, добавляется в оба меню цветов и сохраняется там на протяжении всего сеанса Emacs.

Для задания цветов нет привязок ключей, но вы можете указывать их при помощи расширенных команд `M-x facemenu-set-foreground` и `M-x facemenu-set-background`. Обе эти команды считывают имя цвета в минибуфере.

### 21.11.6 Отступы в форматированном тексте

При редактировании форматированного текста вы можете задать различные величины отступа для правого или левого края целого абзаца или его части. Указанные вам поля автоматически учитываются команды Emacs для заполнения (см. Раздел 21.5 [Заполнение], с. 185) и разрыва строк.

Подменю Indentation предоставляет удобный интерфейс для указания этих свойств. Оно содержит четыре пункта:

`Indent More`
: Увеличивает отступ области на 4 столбца (`increase-left-margin`). В режиме Enriched эта команда также доступна на `C-x` ⟨TAB⟩; если вы предоставите числовой аргумент, то он говорит, сколько столбцов нужно добавить к полю (отрицательный аргумент уменьшает число столбцов).

`Indent Less`
: Удаляет 4 столбца отступа из области.

`Indent Right More`
: Сужает область, делая с правого края отступ в 4 столбца.

`Indent Right Less`
: Удаляет 4 столбца отступа с правого края.

Вы можете использовать эти команды несколько раз для увеличения или уменьшения величины отступа.

Наиболее частый способ применения этих команд — изменять отступ целого абзаца. Однако это не единственное их применение. Вы можете поменять размеры полей в любой точке; новые значения проявляются в конце этой строки (для правого поля) или в начале следующей (для левого поля).

Это позволяет форматировать абзацы с *висящими отступами*, что означает, что отступ первой строки меньше отступа последующих строк. Чтобы установить висящий отступ, увеличьте отступ области, начинающейся после первого слова абзаца и продолжающейся до его конца.

Отступ в первой строке абзаца делается проще. Установите поле для всего абзаца там, где вы хотели бы видеть его для тела абзаца, а затем увеличьте отступ первой строки, добавив пробелы или знаки табуляции.

Иногда в результате редактирования заполнение абзаца сбивается — части абзаца могут выйти за левые или правые поля. Когда такое происходит, воспользуйтесь `M-q` (`fill-paragraph`), чтобы перезаполнить этот абзац.

Число столбцов, которые добавляют или удаляют из отступа эти команды, задается переменной `standard-indent`. Ее значение равно по умолчанию четырем. Общее правое поле, принимаемое по умолчанию для режима Enriched, контролируется переменной `fill-column`, как обычно.

Префикс заполнения, если он задан, действует совместно с указанным отступом абзаца: `C-x .` не включает пропуск из указанного отступа в новое значение префикса заполнения, а команды заполнения ищут префикс заполнения в каждой строке после отступа. См. Раздел 21.5.3 [Префикс заполнения], с. 187.

### 21.11.7 Выравнивание в форматированном тексте

При редактировании форматированного текста вы можете задавать различные стили выравнивания абзацев. Указанный вами стиль автоматически учитывается командами Emacs для заполнения.

Подменю Justification предоставляет удобный интерфейс для указания стиля выравнивания. Оно содержит пять пунктов:

`Flush Left`
: Это наиболее распространенный стиль выравнивания (по крайней мере для английского языка). Строки выравниваются по левому полю, но оставляются неровными с правого края.

`Flush Right`
: Это выравнивает каждую строку по правому полю. Если необходимо, слева добавляются пробелы и знаки табуляции, чтобы правые концы строк выстраивались в линию.

`Full`
: Это выравнивает текст по обоим концам строк. Выровненный таким образом текст смотрится красиво в печатной книге, где все пробелы можно настроить одинаково, но смотрится не так хорошо с равноширинным шрифтом на экране. Возможно, будущие версии Emacs позволят настраивать ширину пробелов в строке, чтобы достичь элегантного выравнивания.

`Center`
: Это центрирует каждую строку между текущими полями.

`None` Это выключает заполнение полностью. Каждая строка будет оставаться такой, как вы ее написали; функции заполнения и автоматического заполнения не будут иметь эффекта в тексте с такой установкой. Вы все же можете делать отступ слева. В незаполненных областях все переводы строк считаются жесткими (см. Раздел 21.11.2 [Жесткие и гибкие переводы строк], с. 199) .

В режиме Enriched вы также можете задавать стиль выравнивания с клавиатуры, используя префиксный знак `M-j`:

`M-j c`
`M-S` Центрирует область (`set-justification-center`).

`M-j u` Делает область невыровненной (`set-justification-none`).

`M-j l` Выравнивает область слева (`set-justification-left`).

`M-j r` Выравнивает область справа (`set-justification-right`).

`M-j f` Выравнивает область полностью (`set-justification-full`).

Стили выравнивания применяются к целым абзацам. Все команды для изменения выравнивания действуют на абзац, содержащий точку, или, если область активна, на все абзацы, пересекающиеся с областью.

Стиль выравнивания по умолчанию задается переменной `default-justification`. Ее значением должен быть один из символов `left`, `right`, `full`, `center` или `none`.

### 21.11.8 Установка других свойств текста

Меню Other Properties позволяет вам добавлять или удалять три других полезных свойства текста: `read-only`, `invisible` и `intangible`. Свойство `intangible` запрещает движение точки внутри этого текста, свойство текста `invisible` делает текст невидимым, а свойство `read-only` запрещает изменение текста.

Для добавления каждого из этих особых свойств к области есть пункт меню. Последний пункт меню, '`Remove Special`', удаляет все эти особые свойства из текста области.

На данный момент свойства `invisible` и `intangible` *не* сохраняются в формате text/enriched. Свойство `read-only` сохраняется, но оно не входит в стандарт формата text/enriched, поэтому другие редакторы могут его игнорировать.

### 21.11.9 Принудительное включение режима Enriched

Обычно Emacs знает, когда вы редактируете форматированный текст, поскольку он распознает специальные пометки, использованные в файле, к которому вы обратились. Однако, бывают ситуации, в которых вы должны предпринять особые меры, чтобы преобразовать содержимое файла или включить режим Enriched:

- Когда вы обращаетесь к файлу, который был создан каким-то другим редактором, Emacs может не распознать этот файл как отфоратированный в text/enriched. В таком случае, когда вы обращаетесь к файлу, вы увидите команды форматирования, а не форматированный текст. Наберите `M-x format-decode-buffer`, чтобы перевести их.
- Когда вы *вставляете* файл в буфер, а не обращаетесь к нему. Emacs делает необходимые преобразования вставляемого текста, но не включает режим Enriched. Если вы хотите сделать это, введите `M-x enriched-mode`.

Команда `format-decode-buffer` переводит текст из различных форматов во внутренний формат Emacs. Она просит вас указать формат, из которого делать преобразование; однако, как правило вы можете просто нажать ⟨RET⟩, что велит Emacs предположить формат самому.

Если вы хотите просмотреть на текст в text/enriched-файле буквально, как последовательность знаков, а не как форматированный текст, воспользуйтесь командой `M-x find-file-literally`. Она обращается к файлу, как и `find-file`, но не производит преобразование формата. Она также подавляет преобразование кодов знаков (см. Раздел 18.7 [Системы кодирования], с. 165) и автоматическую распаковку (см. Раздел 14.11 [Сжатые файлы], с. 133). Чтобы выключить преобразование формата, но позволить перевод кодов знаков и/или автоматическую распаковку, если она нужна, используйте `format-find-file` с подходящими аргументами.

# 22 Редактирование программ

В Emacs есть много команд, предназначенных для понимания синтаксиса языков программирования, таких как Лисп и Си. Эти команды могут:

- Передвигать или уничтожать сбалансированные выражения или *s-выражения* (см. Раздел 22.2 [Списки], с. 206).
- Передвигать через или помечать выражения верхнего уровня — *определения функций* в Лиспе, функции в Си (см. Раздел 22.4 [Определения функций], с. 208).
- Показывать, как сбалансированы круглые скобки (см. Раздел 22.6 [Парность], с. 218).
- Вставлять, уничтожать или выравнивать комментарии (см. Раздел 22.7 [Комментарии], с. 219).
- Следовать обычным соглашениям об отступах, принятых в языке (см. Раздел 22.5 [Отступы в программах], с. 208).

Команды для слов, предложений и абзацев очень удобны при редактировании программ, даже хотя их традиционным применением является редактирование текстов на естественном языке. Большинство символов содержат слова (см. Раздел 21.1 [Слова], с. 181); предложения могут быть найдены в строках или комментариях (см. Раздел 21.2 [Предложения], с. 182). Абзацы так таковые не присутствуют в коде, но команды работы с абзацами тем не менее полезны, так как основные режимы для языков программирования определяют абзацы как куски текста, начинающиеся и заканчивающиеся пустыми строками (см. Раздел 21.3 [Абзацы], с. 183). Разумное использование пустых строк для улучшения читаемости программы будет также предоставлять командам, оперрирующим с абзацами, интересные куски текста для работы.

Средство выборочного показа полезно для просмотра общей структуры функции (см. Раздел 11.4 [Выборочный показ], с. 83). Это средство делает так, что на экране появляются только те строки, отступ в которых меньше заданной величины.

## 22.1 Основные режимы для языков программирования

Emacs также имеет основные режимы для языков программирования Лисп, Scheme (вариант Лиспа), Awk, Си, Си++, Фортран, Icon, Java, Objective-C, Паскаль, Perl, Pike, CORBA IDL, и Tcl. Есть также основной режим для Make-файлов, называемый режимом Makefile. Второй альтернативный режим для Perl называется режимом CPerl.

В идеале, основной режим должен быть реализован для каждого языка программирования, который вы можете пожелать редактировать при помощи Emacs; но часто режим для одного языка может обслуживать другие языки со схожим синтаксисом. Существующие режимы для языков — это те, которые кто-то взял на себя труд написать.

Есть несколько разновидностей режима Lisp, которые отличаются способом взаимодействия с исполнением Лиспа. См. Раздел 23.8 [Вычисление Лиспа], с. 254.

Каждый из основных режимов для языка программирования определяет ключ ⟨TAB⟩ для запуска функции, делающей отступ, которой известны соглашения об отступах для этого языка и которая соответственно изменяет отступ текущей строки. Например, в режиме C, ⟨TAB⟩ привязан к `c-indent-line`. `C-j` обычно определяется так, чтобы делать ⟨RET⟩, за которым следует ⟨TAB⟩; таким образом, эта команда тоже делает отступ в режимозависимом виде.

В большинстве языков программирования отступ часто изменяется от строки к строке. Следовательно, основные режимы для таких языков переприязывают ⟨DEL⟩ так, чтобы он трактовал знак табуляции как эквивалентное количество пробелов (используя команду `backward-delete-char-untabify`). Это позволяет стирать отступ по одному столбцу, не заботясь о том, сделан ли он с помощью пробелов или знаков табуляции. Чтобы удалить в этих режимах знак табуляции перед точкой, используйте `C-b C-d`.

Режимы языков программирования определяют, что абзацы разделяются только пустыми строками, так что команды работы с абзацами остаются полезными. Режим Auto Fill, включенный в основном режиме языка программирования, делает отступ в создаваемых им новых строках.

Включение основного режима запускает обычную ловушку, называемую *ловушкой режима*, которая является значением лисповской переменной. Для каждого основного режима есть своя ловушка, и имя этой ловушки всегда составляется из имени команды, запускающей этот режим, и слова '`-hook`'. Например, включение режима C запускает ловушку `c-mode-hook`, тогда как включение режима Lisp запускает ловушку `lisp-mode-hook`. См. Раздел 31.2.3 [Ловушки], с. 349.

## 22.2 Списки и s-выражения

По соглашению, ключи Emacs для работы со сбалансированными выражениями обычно являются Control-Meta-знаками. По действию они стремятся походить на свои Control- и Meta-аналоги. Обычно считается, что эти команды имеют отношение к выражениям в языках программирования, но они могут оказаться полезными в любом языке, в котором существует какая-либо разновидность круглых скобок (включая естественные языки).

Эти команды делятся на два класса. Некоторые имеют дело только со *списками* (заключенными в скобки группами). Они не видят ничего, кроме круглых, квадратных или фигурных скобок (тех, которые должны быть сбалансированы в языке, с которым вы работаете) и управляющих символов, которые могут быть использованы, чтобы экранировать эти скобки.

Другие команды имеют дело с выражениями или *s-выражениями*. Слово 's-выражение' происходит от *s-expression*, старого термина для выражения в Лиспе. Но в Emacs понятие 's-выражение' не ограничивается Лиспом. Оно обозначает выражение в любом языке, на котором написана ваша программа. Каждый язык программирования имеет свой собственный основной режим, который настраивает синтаксические таблицы так, что выражения на этом языке рассматриваются как s-выражения.

Обычно s-выражение включает в себя символы, числа и строковые константы, а также все, что содержится в круглых, квадратных или фигурных скобках.

В языках, которые используют префиксные и инфиксные операторы, таких как Си, не все выражения могут быть s-выражениями. Например, режим C не распознает '`foo + bar`' как s-выражение, несмотря на то, что это *является* выражением Си; он распознает '`foo`' как одно s-выражение и '`bar`' как другое, со знаком '`+`' в качестве пунктуации между ними. Это фундаментальная неоднозначность: как '`foo + bar`', так и '`foo`' являются законными кандидатами на s-выражение, через которое нужно передвинуться, если точка находится на '`f`'. Заметьте, что '`(foo + bar)`' — это единое s-выражение в режиме C.

Некоторые языки имеют туманную форму синтаксиса выражений, и никто не позаботился о том, чтобы Emacs его правильно понимал.

## 22.3 Команды работы со списками и s-выражениями

`C-M-f` Передвинуться вперед через s-выражение (`forward-sexp`).

`C-M-b` Передвинуться назад через s-выражение (`backward-sexp`).

`C-M-k` Уничтожить s-выражение вперед (`kill-sexp`).

`C-M-`⟨DEL⟩ Уничтожить s-выражение назад (`backward-kill-sexp`).

`C-M-u` Перейти вверх и назад по структуре списка (`backward-up-list`).

`C-M-d` Перейти вниз и вперед по структуре списка (`down-list`).

`C-M-n` Передвинуться вперед через список (`forward-list`).

`C-M-p` Передвинуться назад через список (`backward-list`).

`C-M-t` Переставить выражения (`transpose-sexps`).

`C-M-@` Поставить метку после следующего выражения (`mark-sexp`).

Чтобы передвинуться вперед через s-выражение, используйте `C-M-f` (`forward-sexp`). Если первая значащая литера после точки — это открывающий ограничитель (‘`(`’ в Лиспе; ‘`(`’, ‘`[`’ или ‘`{`’ в Си), то `C-M-f` передвигает за парный закрывающий ограничитель. Если этот знак начинает символ, строку или число, то `C-M-f` передвигает через них.

Команда `C-M-b` (`backward-sexp`) двигает назад через s-выражение. Подробные правила похожи на описанные выше для `C-M-f`, но с противоположным направлением. Если перед s-выражением стоят какие-либо префиксные символы (в Лиспе это одиночная кавычка, обратная кавычка и запятая), то `C-M-b` переходит и через них. Команды для s-выражений передвигаются через комментарии, как это делается для пропусков в большинстве режимов.

`C-M-f` или `C-M-b` с аргументом повторяют операцию заданное число раз; с отрицательным аргументом, они перемещают в противоположном направлении.

Уничтожение целого s-выражения может быть сделано при помощи `C-M-k` (`kill-sexp`) или `C-M-`⟨DEL⟩ (`backward-kill-sexp`). `C-M-k` уничтожает знаки, через которые передвинула бы `C-M-f`, а `C-M-`⟨DEL⟩ уничтожает знаки, через которые передвинула бы `C-M-b`.

*Команды для списков* передвигают через списки, как и команды s-выражений, но легко перескакивают через любое количество других видов s-выражений (символы, строки и так далее). Это `C-M-n` (`forward-list`) и `C-M-p` (`backward-list`). Они полезны в основном тем, что обычно игнорируют комментарии (так как комментарии как правило не содержат никаких списков).

`C-M-n` и `C-M-p` остаются на одном уровне скобок, когда это возможно. Чтобы передвинуться *вверх* на один (или $n$) уровень, используйте `C-M-u` (`backward-up-list`). `C-M-u` двигает назад и вверх мимо одного непарного открывающего ограничителя. Положительный аргумент служит счетчиком повторов; отрицательный аргумент меняет направление движения и также запрашивает повторение, таким образом, в этом случае движение происходит вперед и вверх на один или больше уровней.

Чтобы передвинуться *вниз* по структуре списков, используйте `C-M-d` (`down-list`). В режиме Lisp, где ‘`(`’ — это единственный открывающий ограничитель, это почти то же самое, что и поиск ‘`(`’. Количество уровней скобок, на какое следует спуститься, определяет аргумент.

Команда `C-M-t` (`transpose-sexp`), которая переносит предыдущее s-выражение через следующее, отчасти кажется случайно сюда попавшей, но тем не менее она очень удобна. Аргумент служит для подсчета числа повторов, а отрицательный аргумент перетаскивает выражение в обратном направлении (таким образом отменяя действие `C-M-t` с положительным аргументом). Аргумент, равный нулю, вместо того чтобы ничего не делать, переставляет местами s-выражения, кончающиеся после точки и метки.

Чтобы установить область вокруг следующего s-выражения в буфере, используйте `C-M-@` (`mark-sexp`), которая ставит пометку в то же самое место, куда должна бы была передвинуться `C-M-f`. `C-M-@` воспринимает аргумент так же, как `C-M-f`. В частности, отрицательный аргумент удобен для установки метки в начале предыдущего s-выражения.

Понимание синтаксиса командами для списков и s-выражений полностью управляется синтаксической таблицей. Любой знак может быть объявлен, например, открывающим ограничителем и действовать как открывающая круглая скобка. См. Раздел 31.6 [Синтаксис], с. 366.

## 22.4 Определения функций

В Emacs, заключенные в скобки группы на верхнем уровне в буфере называются *определениями функций*. Это название происходит от того факта, что большинство списков верхнего уровня в Лисп-файле — это экземпляры специальной формы `defun`, но любая группа верхнего уровня, заключенная в скобки, на языке Emacs понимается как определение функции, независимо от ее содержания и от используемого языка программирования. Например, тело функции в Си — это определение функции.

`C-M-a` Передвинуться к началу текущего или предшествующего определения функции (`beginning-of-defun`).

`C-M-e` Передвинуться в конец текущего или следующего определения функции (`end-of-defun`).

`C-M-h` Пометить область вокруг всего текущего или следующего определения функции (`mark-defun`).

Команды движения к началу или концу текущего определения функции — это `C-M-a` (`beginning-of-defun`) и `C-M-e` (`end-of-defun`).

Если вы пожелаете произвести какие-то действия над текущим определением функции, используйте `C-M-h` (`mark-defun`), которая ставит точку в начале и метку в конце текущего или следующего определения функции. Например, это простейший способ получить готовое для перемещения в другое место определение функции. В режиме C, `C-M-h` запускает функцию `c-mark-function`, которая почти эквивалентна `mark-defun`; различие состоит в том, что она переходит через объявления аргументов, имя функции и тип возвращаемых данных, так что функция Си оказывается внутри области полностью. См. Раздел 8.4 [Пометка объектов], с. 65.

Emacs предполагает, что любые открывающие скобки, найденные в самом левом столбце, — это начало определения функции. Поэтому **никогда не ставьте открывающие скобки с левого края в Лисп-файле, если они не являются началом списка верхнего уровня. Никогда не ставьте открывающую фигурную скобку или другой открывающий ограничитель в начале строки в программе на Си, если только они не начинают тело функции.** Большинство возможных проблем возникает, когда вы хотите поставить открывающий ограничитель в начале строки внутри строковой константы. Чтобы избежать неприятностей, поставьте экранирующий знак (‘\’ в Си и Emacs Lisp, ‘/’ в некоторых других диалектах Лиспа) перед открывающим ограничителем. Это не повлияет на содержимое строки.

В очень далеком прошлом оригинальный Emacs находил определения функций, двигаясь вверх по уровням скобок до тех пор, пока не доходил до уровня, от которого некуда было идти дальше. Это всегда требовало просмотра полного пути обратно до начала буфера, даже для маленькой функции. Чтобы ускорить эту операцию, Emacs был изменен, и теперь он предполагает, что любой знак ‘(’ (или любой другой, приписанный к синтаксическому классу открывающего ограничителя) на левой границе строки — это начало определения функций. Эта эвристика почти всегда правильна и позволяет избежать ресурсоемкого просмотра; однако, она требует выполнения описанных выше соглашений.

## 22.5 Отступы в программах

Наилучший способ сохранить правильность отступов в программе — это использовать Emacs для создания новых отступов по мере внесения изменений. В Emacs есть команды для создания правильного отступа одиночной строки, заданного числа строк или всех строк внутри одной группы, заключенной в скобки.

Emacs также предоставляет программу структурной печати для Лиспа, реализованную в библиотеке `pp`. Эта программа переформатирует лисповский объект, выбирая отступы таким образом, чтобы результат хорошо выглядел и удобно читался.

### 22.5.1 Основные команды для отступов в программах

⟨TAB⟩ Установить отступ текущей строки.

`C-j` Эквивалент ⟨RET⟩, за которым следует ⟨TAB⟩ (`newline-and-indent`).

Основная команда отступа — это ⟨TAB⟩, которая дает текущей строке правильный отступ, основываясь на отступе предыдущих строк. Функция, которую запускает ⟨TAB⟩, зависит от основного режима; в режиме Lisp это `lisp-indent-line`, в режиме C это `c-indent-line` и так далее. Эти функции понимают различные синтаксисы разных языков, но все они делают примерно одно и то же. ⟨TAB⟩ в основном режиме любого языка программирования вставляет или удаляет пробельные знаки в начале текущей строки, независимо от того, где в строке располагается точка. Если точка находится среди пробельных знаков в начале строки, ⟨TAB⟩ оставляет ее после них; в противном случае ⟨TAB⟩ оставляет точку фиксированной по отношению к окружающим ее знакам.

Чтобы вставить в точке знак табуляции, используйте `C-q` ⟨TAB⟩.

При вводе нового кода используйте `C-j` (`newline-and-indent`), которая эквивалентна ⟨RET⟩, за которой следует ⟨TAB⟩. `C-j` создает пустую строку, а затем дает ей соответствующий отступ.

⟨TAB⟩ создает отступ во второй и следующих строках тела группы, заключенной в скобки, так, что каждая оказывается под предыдущей; поэтому, если вы изменяете отступ одной строки на нестандартный, то строки ниже будут стремиться следовать ему. Такое поведение удобно в тех случаях, когда вы заменяете стандартный результат ⟨TAB⟩, поскольку вы нашли его неэстетичным для какой-то строки.

Помните, что открывающие круглые и фигурные скобки или другие открывающие ограничители на левом крае рассматриваются Emacs (и правилами отступа) как начало функции. Поэтому вы никогда не должны ставить открывающий ограничитель, не являющийся началом функции, в нулевом столбце, даже внутри строковой константы. Это ограничение жизненно важно для скорости работы команд отступа; вы должны просто принять его. Для более подробной информации об этом смотрите Раздел 22.4 [Определения функций], с. 208.

### 22.5.2 Отступ в нескольких строках

Если вы хотите поменять отступ нескольких строк кода, которые были изменены или передвинуты на другой уровень в структуре списков, вы имеете в своем распоряжении несколько команд.

`C-M-q` Сделать новый отступ во всех строках в пределах одного списка (`indent-sexp`).

`C-u` ⟨TAB⟩ Сдвинуть весь список жестко в сторону так, чтобы его первая строка получила надлежащий отступ.

`C-M-\` Сделать новый отступ во всех строках в области (`indent-region`).

Вы можете вновь сделать отступ содержимого одиночного списка, переместив точку в его начало и набрав `C-M-q` (это команда `indent-sexp` в режиме Lisp, `c-indent-exp` в режиме C; она также привязана к другим подходящим функциям в других режимах). Отступ строки, на которой начинается это s-выражение, не изменяется; поэтому изменяется только относительный отступ в пределах списка, а не его позиция. Чтобы исправить также и его позицию, наберите ⟨TAB⟩ перед `C-M-q`.

Если относительный отступ внутри списка правильный, но отступ его первой строки — нет, перейдите к этой строке и наберите `C-u` ⟨TAB⟩. ⟨TAB⟩ с числовым аргументом делает в текущей строке обычный отступ, а затем изменяет отступ во всех строках в группе, начиная с текущей, на ту же самую величину. Другими словами, она обновляет отступ целой

группы как неделимой единицы. Это разумно, хотя и не изменяет строки, которые начинаются внутри строковых констант, или строки препроцессора Си, когда это происходит в режиме C.

Можно указать диапазон строк, в которых следует вновь сделать отступ, другим способом — с помощью области. Команда `C-M-\` (`indent-region`) применяет ⟨TAB⟩ к каждой строке, чей первый знак находится между точкой и меткой.

### 22.5.3 Настройка отступов для Лиспа

Образец отступа для лисповского выражения может зависеть от функции, вызываемой этим выражением. Для каждой лисповской функции вы можете выбирать среди нескольких предопределенных образцов отступа или определить произвольный отступ с помощью программы на Лиспе.

Стандартный шаблон отступа таков: вторая строка выражения сдвигается под первый аргумент, если он находится на той же самой строке, что и начало выражения; в противном случае вторая строка сдвигается под имя функции. Каждая следующая строка имеет тот же отступ, что и предыдущая строка с той же глубиной вложенности.

Если переменная `lisp-indent-offset` не равна `nil`, то она перекрывает обычный шаблон отступа для второй строки выражения, так что такие строки всегда сдвигаются вправо на `lisp-indent-offset` столбцов дальше, чем содержащий их список.

Стандартный шаблон перекрывается в некоторых определенных функциях. Для функций, чьи имена начинаются с `def`, отступ второй строки всегда делается на `lisp-body-indention` дополнительных столбцов дальше открывающей скобки, начинающей выражение.

Стандартный шаблон может перекрываться различными способами для отдельных функций согласно свойству имени этой функции `lisp-indent-function`. Есть четыре варианта для этого свойства:

`nil` Это то же самое, что и отсутствие свойства; используется стандартный шаблон отступа.

`defun` Шаблон, используемый для имен функций, которые начинаются с `def`, также используется и для этой функции.

число, *n* Первые *n* аргументов этой функции считаются *отличительными* аргументами, остальные рассматриваются как *тело* выражения. Строка в этом выражении отступается в соответствии с тем, является ли в ней первый аргумент отличительным или нет. Если аргумент является частью тела, то строка отступается на `lisp-body-indent` столбцов больше, чем открывающая скобка, начинающая содержащее ее выражение. Если аргумент является отличительным, и это первый или второй аргумент, то отступ делается на *вдвое большее* число дополнительных столбцов. Если аргумент отличителен и не является первым или вторым, то для этой строки применяется стандартный шаблон.

символ, *символ*
*символ* должен быть именем функции; эта функция вызывается для вычисления отступа строки в пределах этого выражения. Функция получает два аргумента:

*состояние* Значение, возвращаемое из `parse-partial-sexp` (это примитив Лиспа для подсчета величины отступов и вложенностей), когда она делает разбор вплоть до начала этой строки.

*позиция* Позиция, с которой начинается строка, в которой делается отступ.

Она должна возвращать либо число, которое равно количеству столбцов отступа для этой строки, либо список, чей головной элемент является таким числом.

Отличие между возвращением числа и возвращением списка заключается в том, что число говорит, что все следующие строки того же уровня вложенности должны получать такой же отступ, как эта строка; список говорит, что следующие строки могут требовать отличные отступы. Это важно, если отступы подсчитываются с помощью `C-M-q`; если значение — это число, то `C-M-q` не нуждается в пересчете отступа для следующих строк до конца списка.

### 22.5.4 Команды для отступов в Си

Вот команды для создания отступов в режиме C и родственных с ним:

`C-c C-q` Обновляет отступ в текущем определении функции верхнего уровня или собирает в одно целое объявление типа (`c-indent-defun`).

`C-M-q` Обновляет отступ в каждой строке сбалансированного выражения, которое следует после точки (`c-indent-exp`). Префиксный аргумент подавляет проверку ошибок и вывод предупреждений о недопустимом синтаксисе.

⟨TAB⟩ Обновляет отступ в текущей строке и/или в некоторых случаях вставляет знак табуляции (`c-indent-command`).

Если `c-tab-always-indent` равна `t`, эта команда всегда обновляет отступ текущей строки и не делает ничего больше. Это принимается по умолчанию.

Если эта переменная равна `nil`, данная команда обновляет отступ текущей строки, только если точка находится с левого края или на отступе; в противном случае она вставляет табуляцию (или эквивалентное число пробелов, если `indent-tabs-mode` равна `nil`).

Любое другое значение (не `nil` или `t`) означает, что нужно всегда обновлять отступ строки, а также вставлять знак табуляции, если точка находится внутри комментария, строки или директивы препроцессора.

`C-u` ⟨TAB⟩ Обновляет отступ текущей строки в соответствии с ее синтаксисом; кроме того, жестко смещает все остальные строки выражения, начинающегося на текущей строке. См. Раздел 22.5.2 [Многострочный отступ], с. 209.

Чтобы обновить отступ всего текущего буфера, наберите `C-x h C-M-\`. Это сначала выделяет весь буфер как область, а затем обновляет отступ в этой области.

Чтобы обновить отступ в текущем блоке, используйте `C-M-u C-M-q`. Эта команда перемещает к началу блока и делает в нем отступ.

### 22.5.5 Настройка отступа в Си

Режим C и родственные режимы используют простой, но гибкий механизм для настройки отступа. Этот механизм работает в два этапа: сначала строки классифицируются синтаксически в соответствии с их содержимым и контекстом; затем каждому виду синтаксических конструкций привязывается значение сдвига, который вы можете настроить.

#### 22.5.5.1 Шаг 1 — синтаксический анализ

На первом шаге механизм отступов в Си смотрит на строку перед той, в которой вы в данный момент делаете отступ, и определяет синтаксические компоненты конструкции на этой строке. Он строит список этих синтаксических компонентов, где каждый компонент содержит *синтаксический символ* и, иногда, позицию в буфере. Некоторые синтаксические символы описывают грамматические элементы, например `statement` и `substatement`; другие описывают положения в составе грамматических элементов, например `class-open` и `knr-argdecl`.

По идее, строка кода на Си всегда имеет отступ относительно отступа какой-то строки выше по этому буферу. Это представляется позицией в буфере в списке синтаксических компонентов.

Вот пример. Предположим, что у нас есть следующий код в буфере с режимом C++ (номера строк в действительности не появляются в буфере):

```
1: void swap (int& a, int& b)
2: {
3:   int tmp = a;
4:   a = b;
5:   b = tmp;
6: }
```

Если вы наберете `C-c C-s` (что запускает команду `c-show-syntactic-information`) на строке 4, будет показан результат работы механизма отступов для этой строки:

```
((statement . 32))
```

Это указывает на то, что данная строка является оператором, и она имеет отступ относительно позиции 32 в буфере, то есть относительно ‘`i`’ в `int` на строке 3. Если вы переместите курсор к строке 3 и наберете `C-c C-s`, это покажет следующее:

```
((defun-block-intro . 28))
```

Это указывает на то, что строка `int` — это первый оператор в блоке, и она имеет отступ относительно позиции 28, то есть фигурной скобки сразу после заголовка функции.

Вот еще один пример:

```
1: int add (int val, int incr, int doit)
2: {
3:   if (doit)
4:     {
5:       return (val + incr);
6:     }
7:   return (val);
8: }
```

Если в строке 4 набрать `C-c C-s`, вы увидите вот что:

```
((substatement-open . 43))
```

Это говорит, что данная фигурная скобка *открывает* блок подоператора. Кстати, *подоператор* — это строка после операторов `if`, `else`, `while`, `do`, `switch`, `for`, `try`, `catch`, `finally` или `synchronized`.

Внутри команд для отступа в Си, после того как строка синтаксически проанализирована, описание результатов анализа хранится в списке в переменной `c-syntactic-context`. Каждый элемент этого списка — это *синтаксический компонент*: пара, содержащая синтаксический символ и (возможно) соответствующую ему позицию в буфере. В списке компонент может несколько элементов; как правило только один из них имеет позицию в буфере.

### 22.5.5.2 Шаг 2 — подсчет отступа

Механизма отступов в Си вычисляет величину отступа для текущей строки, используя список синтаксических компонентов, `c-syntactic-context`, полученный из синтаксического анализа. Каждый компонент — это пара, которая содержит синтаксический символ и может содержать позицию в буфере.

Каждый компонент дает вклад в окончательный отступ строки двумя путями. Во-первых, синтаксический символ определяет элемент `c-offsets-alist`, это ассоциативный

список, ставящий в соответствие синтаксическим символам величины сдвига. Сдвиг каждого синтаксического символа добавляется к общему отступу. Во-вторых, если компонент включает позицию в буфере, к отступу добавляется номер столбца этой позиции. Все эти сдвиги и номера столбцов с сумме дают общий отступ.

Следующие примеры демонстрируют работу механизма отступов в языке Си:

```
1: void swap (int& a, int& b)
2: {
3:   int tmp = a;
4:   a = b;
5:   b = tmp;
6: }
```

Предположим, что точка находится на строке 3, и вы нажимаете ⟨TAB⟩, чтобы обновить в этой строке отступ. Как объяснялось выше (см. Раздел 22.5.5.1 [Синтаксический анализ], с. 211), синтаксическим компонетом этой строки будет:

```
((defun-block-intro . 28))
```

В данном случае при подсчете отступа сначала просматривается `defun-block-intro` в ассоциативном списке `c-offsets-alist`. Предположим, что там найдено число 2; оно добавляется к общему (инициализированному нулем), выдавая общей обновленный отступ в 2 пробела.

Следующий шаг — найти номер столбца для позиции 28 в буфере. Поскольку фигурная скобка в позиции 28 расположена в нулевом столбце, к общему числу добавляется 0. Так как в этой строке есть только один синтаксический компонет, общий отступ для этой строки равен двум пробелам.

```
1: int add (int val, int incr, int doit)
2: {
3:   if (doit)
4:     {
5:       return(val + incr);
6:     }
7:   return(val);
8: }
```

Если вы нажмете ⟨TAB⟩ в строке 4, повторяется такой же процесс, но с иными данными. Список синтаксических компонентов для этой строки таков:

```
((substatement-open . 43))
```

Здесь первое, что делается для посчета отступа, — ищется символ `substatement-open` в `c-offsets-alist`. Будем считать, что сдвиг для этого символа равен 2. В этом месте промежуточное общее значение равно 2 (`0 + 2 = 2`). Затем к нему добавляется номер строки позиции 43 в буфере, где стоит ‘`i`’ из `if` на строке 3. Этот знак расположен во втором столбце на строке. Итого в сумме получается 4 пробела.

Если при анализе строки появляется синтаксический символ, который отсутствует в `c-offsets-alist`, он игнорируется; и это является ошибкой, если кроме того переменная `c-strict-syntax-p` отлична от `nil`.

### 22.5.5.3 Изменение стиля отступов

Есть два способа настроить стиль отступов для режимов, подобных режиму C. Во-первых, вы можете выбрать один из предопределенных стилей, каждый из которых задает сдвиги для всех синтаксических символов. Для большей гибкости вы можете настоить обработку отдельных синтаксических символов. См. Раздел 22.5.5.4 [Синтаксические символы], с. 214, перечень всех определенных синтаксических символов.

`M-x c-set-style` ⟨RET⟩ *стиль* ⟨RET⟩
: Выбирает предопределенный стиль *стиль*. Чтобы получить перечень поддерживаемых стилей, наберите при вводе стиля знак `?`; чтобы узнать, как выглядит тот или иной стиль, выберите его и примените для фрагмента кода на Си.

`C-c C-o` *символ* ⟨RET⟩ *сдвиг* ⟨RET⟩
: Устанавливает сдвиг для синтаксического символа *символ* (`c-set-offset`). Второй аргумент, *сдвиг*, указывает новую величину сдвига.

Размер отступа для каждого синтаксического символа управляется переменной `c-offsets-alist`. Ее значение — это ассоциативный список, и каждый элемент этого списка имеет форму (*синтаксический-символ* . *сдвиг*). Изменяя сдвиги для разных синтаксических символов, вы можете настраивать отступы в мельчайших подробностях. Чтобы изменить этот ассоциативный список, используйте `c-set-offset` (смотрите ниже).

Значение каждого сдвига в `c-offsets-alist` может быть целым числом, именем функции или переменной, списком или одним их символов `+`, `-`, `++`, `--`, `*` или `/`, обозначающих положительные или отрицательные кратные переменной `c-basic-offset`. Таким образом, если вы хотите поменять уровни отступов с трех пробелов на два пробела, установите `c-basic-offset` в значение 3.

Использование функции в качестве значения сдвига предоставляет полную гибкость в настройке отступов. Эта функция вызывается с одним аргументом, содержащим пару из синтаксического символа и позиции в буфере, если она есть. Функция должна возвращать целое число, равное сдвигу.

Если значением сдвига является список, его элементы обрабатываются в соответствии с описанными выше правилами, пока не найдено отличное от `nil` значение. Тогда это значение добавляется к общему отступу обычным способом. Основное применение этого состоит в сложении результатов нескольких функций.

Команда `C-c C-o` (`c-set-offset`) — это простейший способ установить сдвиги, как интерактивно, так и в вашем файле '`~/.emacs`'. Сначала укажите синтаксический символ, а потом желаемый сдвиг. См. Раздел 22.5.5.4 [Синтаксические символы], с. 214, перечень допустимых синтаксических символов и их значений.

### 22.5.5.4 Синтаксические символы

Это таблица допустимых синтаксических символов для отступов режима C и родственных с ним режимов и их синтаксические значения. Обычно всем этим символам приписывается сдвиг в `c-offsets-alist`.

`string` Внутри строки, занимащей несколько строк в буфере.

`c` Внутри многострочного блочного комментария в стиле Си.

`defun-open`
: На фигурной скобке, которая открывает определение функции.

`defun-close`
: На фигурной скобке, которая закрывает определение функции.

`defun-block-intro`
: На первой строке определения функции верхнего уровня.

`class-open`
: На фигурной скобке, которая открывает определение класса.

`class-close`
: На фигурной скобке, которая закрывает определение класса.

`inline-open`
: На фигурной скобке, которая открывает определяемый внутри класса inline-метод.

`inline-close`
: На фигурной скобке, которая закрывает определяемый внутри класса inline-метод.

`extern-lang-open`
: На фигурной скобке, которая открывает блок на внешнем языке.

`extern-lang-close`
: На фигурной скобке, которая закрывает блок на внешнем языке.

`func-decl-cont`
: На области между списком аргументов в определении функции и открывающей это определение фигурной скобкой (исключая определения функций в стиле K&R). В Си вы не можете писать здесь ничего, кроме пробельных знаков и комментариев; в Си++ и Java в этом контекте могут появляться объявления `throws` и другие вещи.

`knr-argdecl-intro`
: На первой строке объявления аргументов в стиле K&R Си.

`knr-argdecl`
: На одной из последующих строк объявления аргументов в стиле K&R Си.

`topmost-intro`
: На первой строке определения конструкции самого верхнего уровня.

`topmost-intro-cont`
: На остальных строках определения самого верхнего уровня.

`member-init-intro`
: На первой строке списка инициализаций членов.

`member-init-cont`
: На последующих строках списка инициализаций членов.

`inher-intro`
: На первой строке списка множественного наследования.

`inher-cont`
: На одной их последующих строк множественного наследования.

`block-open`
: На открывающей фигурной скобке операторного блока.

`block-close`
: На закрывающей фигурной скобке операторного блока.

`brace-list-open`
: На открывающей фигурной скобке списка массива `enum` или `static`.

`brace-list-close`
: На закрывающей фигурной скобке списка массива `enum` или `static`.

`brace-list-intro`
: На первой строке списка массива `enum` или `static`.

`brace-list-entry`
: На одной их последующих строк списка массива `enum` или `static`.

`brace-entry-open`
: На одной из последующих строк списка массива `enum` или `static`, когда строка начинается с открывающей фигурной скобки.

`statement`
: На обычном операторе.

`statement-cont`
: На строке продолжения оператора.

`statement-block-intro`
: На первой строке нового операторного блока.

`statement-case-intro`
: На первой строке “блока” `case`.

`statement-case-open`
: На первой строке блока `case`, начинающейся с фигурной скобки.

`inexpr-statement`
: На операторном блоке внутри выражения. Это используется для расширения GNU в языке Си и для для специальных функций Pike, которые принимают в качестве аргумента операторный блок.

`inexpr-class`
: На определении класса внутри выражения. Это используется для аноноимных классов и аноноимных инициализаторов массивов в Java.

`substatement`
: На первой строке после `if`, `while`, `for`, `do` или `else`.

`substatement-open`
: На фигурной скобке, открывающей блок подоператора.

`case-label`
: На метке `case` или `default`.

`access-label`
: На метках доступа Си++ `private`, `protected` или `public`.

`label`
: На обычной метке.

`do-while-closure`
: На `while`, который завершает конструкцию `do-while`.

`else-clause`
: На `else` конструкции `if-else`.

`catch-clause`
: На строках `catch` и `finally` в конструкциях `try...catch` в Си++ и Java.

`comment-intro`
: На строке, содержащей только начало комментария.

`arglist-intro`
: На первой строке списка аргументов.

`arglist-cont`
: На одной из последующих строк списка аргументов, когда на строке с открывающей список аргументов круглой скобкой нет ни одного аргумента.

`arglist-cont-nonempty`
: На одной из последующих строк списка аргументов, когда на строке с открывающей список аргументов круглой скобкой есть хотя бы один аргумент.

`arglist-close`
: На закрывающей круглой скобке списка аргументов.

`stream-op`
: На одной строк продолжения конструкции потокового оператора.

`inclass`
: На конструкции, вложенной в определение класса. Отступ относителен открывающей фигурной скобке определения класса.

`inextern-lang`
: На конструкции, вложенной в блок на внешнем языке.

`inexpr-statement`
: На первой строке операторного блока внутри выражения. Это нужно для расширения GCC в языке Си, которое использует синтаксис `({ ... })`. Это также нужно для специальных функций в Pike, принимающих в качестве аргумента операторный блок.

`inexpr-class`
: На первой строке определения класса внутри выражения. Это используется для аноноимных классов и аноноимных инициализаторов массивов в Java.

`cpp-macro`
: На начале макроса препроцессора.

`friend`
: На объявлении Си++ `friend`.

`objc-method-intro`
: На первой строке определения метода Objective-C.

`objc-method-args-cont`
: На одной из строк продолжения определения метода Objective-C.

`objc-method-call-cont`
: На одной из строк продолжения вызова метода Objective-C.

`inlambda`
: Как `inclass`, но применяется внутри лямбда-функций (т.е. анонимных). Используется только в Pike.

`lambda-intro-cont`
: На строке, продолжающей заголовок лямбда-функции, между ключевым словом `lambda` и телом функции. Используется только в Pike.

#### 22.5.5.5 Переменные, управляющие отступами в Си

Этот раздел описывает дополнительные переменные, которые управляют поведением отступов в режиме C и родственных с ним режимах.

`c-offsets-alist`
: Ассоциативный список синтаксических символов и их сдвигов. Вы не должны менять его прямо, делайте это через `c-set-offset`. См. Раздел 22.5.5.3 [Изменение стиля отступов], с. 213, для подробностей.

`c-style-alist`
: Переменная для определения стилей отступов; смотрите ниже.

`c-basic-offset`
: Базовый сдвиг, используемый символами `+` и `-` в `c-offsets-alist`.

`c-special-indent-hook`
: Ловушка для специальных подстроек отступов, определяемых пользователем. Эта ловушка вызывается после того, как в строке уже сделан отступ режимом C или родственным с ним режимом.

Переменная `c-style-alist` задает предопределенные стили отступов. Каждый элемент имеет форму (*имя установка-переменной...*), где *имя* — это имя стиля. Каждая *установка-переменной* имеет форму (*переменная . значение*); *переменная* — это одна из настроечных переменных, используемых режимом C, а *значение* — это значение для этой переменной, когда используется выбранный стиль.

Когда *переменная* равна `c-offsets-alist`, это особый случай: *значение* добавляется в начало значения `c-offsets-alist`, а не замещает его. Следовательно, *значение* не обязано указывать каждый синтаксический символ — можно написать только те, для которых стиль отличен от принимаемого по умолчанию.

Отступы строк, содержащих только комментарии, также подвержены влиянию переменной `c-comment-only-line-offset` (см. Раздел 22.15.5 [Комментарии в Си], с. 239).

### 22.5.5.6 Стили отступов в Си

*Стиль Си* — это набор настроек стиля отступов. Emacs поставляется с несколькими предопределенными стилями отступов для C и родственных режимов, включая `gnu`, `k&r`, `bsd`, `stroustrup`, `linux`, `python`, `java`, `whitesmith`, `ellemtel` и `cc-mode`. По умолчанию применяется стиль `gnu`.

Чтобы выбрать нужный вам стиль, используйте команду `M-x c-set-style`. Задавайте имя стиля в качестве аргумента (регистр не имеет значения). Выбранный стиль применяется только к новым буферам, но не к тем, что вы уже редактируете. Вы также можете установить переменную `c-default-style`, чтобы указать стиль для различных основных режимов. Ее значением должен быть ассоциативный список, где каждый элемент задает один основной режим и стиль отступов, который для него нужно использовать. Например,

```
(setq c-default-style
      '((java-mode . "java") (other . "gnu")))
```

определяет явный выбор для режима Java и велит принимать стиль ‘`gnu`’ по умолчанию для остальных C-подобных режимов.

Чтобы определить новый стиль отступов в Си, вызовите функцию `c-add-style`:

`(c-add-style` *имя значения применить-сразу*`)`

Здесь *имя* — это имя нового стиля (строка), а *значения* — это ассоциативный список, чьи элементы имеют форму (*переменная . значение*). Задаваемые вами переменные должны быть среди описанных в этом руководстве (см. Раздел 22.5.5.5 [Переменные для отступов в Си], с. 217).

Если *применить-сразу* не равна `nil`, `c-add-style` переключает в новый стиль сразу после его определения.

## 22.6 Автоматическое отображение парных скобок

Способность Emacs находить парные скобки предназначается для того, чтобы автоматически показывать, как скобки в тексте соответствуют друг другу. Всякий раз, когда вы набираете самовставляющийся знак, который является закрывающим ограничителем, курсор на мгновение передвигается в положение соответствующего открывающего ограничителя, при условии, что он находится на экране. Если его нет на экране, то в эхо-области показывается немного текста, начинающегося с открывающего ограничителя. В любом случае вы можете сказать, какая группа закрывается.

В Лиспе автоматическое соответствие применяется только к круглым скобкам. В Си оно применяется также к фигурным и квадратным скобкам. Emacs узнает, какие знаки рассматривать как парные ограничители, основываясь на синтаксической таблице, которая устанавливается основным режимом. См. Раздел 31.6 [Синтаксис], с. 366.

Если отрывающий и закрывающий ограничители не соответствуют друг другу, как например в ‘[x)’, в эхо-области появляется предупреждающее сообщение. Правильные пары описываются в синтаксической таблице.

Отображением парных скобок управляют три переменные. `blink-matching-paren` включает или выключает эту возможность; `nil` выключает, а значение по умолчанию, равное `t`, включает ее. `blink-matching-delay` говорит, сколько секунд нужно ожидать; по умолчанию это 1, но на некоторых системах полезно задать часть секунды. `blink-matching-paren-distance` указывает, сколько знаков в обратном направлении надо исследовать, чтобы найти парный открывающий ограничитель. Если пара не будет найдена на таком расстоянии, то сканирование останавливается и ничего не отображается. Это делается для того, чтобы избежать больших затрат времени на поиск парного ограничителя в том случае, если пары не существует. По умолчанию она равна 12000.

При использовании X Windows вы можете запросить более мощную альтернативную разновидность автоматического показа парных скобок, включив режим Show Paren. Этот режим выключает обычный способ отображения парных скобок и использует вместо него подсветку совпадений. Когда точка находится после закрывающей скобки, подвечиваются эта закрывающая скобка и парная ей открывающая; иначе, если точка находится перед открывающей скобкой, подсвечивается парная скобка. (Подсвечивать открывающую скобку после точки не нужно, потому что поверх этого знака находится курсор.) Для включения и выключения этого режима используйте команду `M-x show-paren-mode`.

## 22.7 Управление комментариями

Поскольку комментарии являются весьма важной частью программирования, Emacs предоставляет особые команды для редактирования и вставки комментарев.

### 22.7.1 Команды для комментариев

Команды комментариев вставляют, уничтожают и выравнивают комментарии:

`M-;` Вставить или выровнять комментарий в текущей строке (`indent-for-comment`).

`C-x ;` Установить столбец комментария (`set-comment-column`).

`C-u - C-x ;`
Уничтожить комментарий в текущей строке (`kill-comment`).

`C-M-j` Подобна ⟨RET⟩, за которой следует вставка или выравнивание комментария (`indent-new-comment-line`).

`M-x comment-region`
Добавить или удалить ограничители комментариев на всех строках области.

Команда, которая создает комментарии, называется `M-;` (`indent-for-comment`). Если на строке еще нет комментария, то создается новый комментарий, выровненный по особому столбцу, называемому *столбцом комментария*. Комментарий создается вставкой строки, с которой, как думает Emacs, должны начинаться комментарии (значение `comment-start`, смотрите ниже). Точка оставляется за этой строкой. Если текст в строке текста простирается дальше столбца комментария, то делается отступ до подходящей границы (обычно вставляется по крайней мере один пробел). Если основной режим определил строку, завершающую комментарий, то она вставляется после точки, чтобы сохранить правильный синтаксис.

`M-;` может быть использована также и для выравнивания существующего комментария. Если строка уже содержит начало комментария, то `M-;` просто передвигает за него точку и

делает отступ до принятой позиции. Исключение: комментарии, начинающиеся в столбце 0, не сдвигаются.

Некоторые основные режимы имеют особые правила отступа для некоторых видов комментариев в определенных контекстах. Например, в коде на Лиспе, комментарии, начинающиеся с двойной точки с запятой, имеют отступ такой же, как если бы они были строками кода, а не отступ до столбца комментария. Комментарии, начинающиеся с трех точек с запятой, предполагается располагать с левой границы строки. Emacs понимает эти соглашения, выполняя отступ комментария с двойной точкой с запятой, используя ⟨TAB⟩ и не изменяя отступ комментария с тройной точкой с запятой вообще.

```
;; Эта просто пример функции
;;; Здесь годятся и 2, и 3 точки с запятой.
(defun foo (x)
;;; А теперь первая часть функции
  ;; Следующая строка добавляет единицу.
  (1+ x))           ; Эта строка добавляет единицу.
```

Для комментария в коде на Си, которому на его строке предшествуют только пробельные знаки, делается такой же отступ, как для строки кода.

Даже когда существующий комментарий имеет правильный отступ, `M-;` по-прежнему полезна для перехода сразу к началу комментария.

Команда `C-u - C-x ;` (`kill-comment`) уничтожает комментарий в текущей строке, если он там есть. Отступ перед началом комментария также уничтожается. Если на этой строке нет комментария, то ничего не делается. Чтобы перенести комментарий в другую строку, передвиньтесь в конец этой строки, сделайте `C-y` и затем `M-;`, чтобы заново его выровнять. Заметьте, что `C-u - C-x ;` — это не отдельный ключ; это `C-x ;` (`set-comment-column`) с отрицательным аргументом. Эта команда запрограммирована таким образом, что когда она получает отрицательный аргумент, она вызывает `kill-comment`. Однако, `kill-comment` — это допустимая команда, которую вы можете непосредственно привязать к ключу, если вы этого хотите.

### 22.7.2 Многострочные комментарии

Если вы набираете комментарий и обнаруживаете, что хотели бы продолжить его на другой строке, то вы можете использовать команду `C-M-j` (`indent-new-comment-line`). Она завершает набранный вами комментарий, затем создает новую пустую строку и начинает новый комментарий, с отступом под старым комментарием. Когда действует режим Auto Fill, то переход за столбец заполнения во время набора комментария приводит к тому, что комментарий будет продолжаться именно таким образом. Если во время набора `C-M-j` точка находится не в конце строки, то текст в оставшейся части строки становится частью новой строки комментария.

Чтобы превратить существующие строки в строки комментариев, используйте команду `M-x comment-region`. Она добавляет ограничители к строкам, которые начинаются в области, делая их таким образом комментариями. С отрицательным аргуменом, она делает обратное — удаляет ограничители комментариев из строк области.

С положительнм аргументом, `comment-region` повторяет последний знак из добавляемой последовательности, начинающей комментарий. Таким образом, в режиме Lisp, `C-u 2 M-x comment-region` добавит ‘`;;`’ на каждую строку. Повторение ограничителей комментария — это способ привлечения к нему внимания. В Лиспе для получения правильных отступов вы должны использовать аргумент, равный двум, между определениями функций, и трем — внутри определений функций.

Переменная `comment-padding` указывает, сколько пробелов должна вставить `comment-region` в каждую строку между ограничителем комментария и изначальным текстом этой строки. По умолчанию это 1.

### 22.7.3 Параметры управления комментариями

Столбец комментария хранится в переменной `comment-column`. Вы можете явно установить ее на нужное число. Или вы можете использовать команду `C-x ;` (`set-comment-column`), которая устанавливает столбец комментария равным тому столбцу, где находится точка. `C-u C-x ;` устанавливает столбец комментария так, чтобы он соответствовал последнему комментарию перед точкой в этом буфере, и затем делает `M-;`, чтобы выравнять строку текущего комментария под предыдущую. Отметим, что `C-u - C-x ;` запускает функцию `kill-comment`, как описано выше.

Переменная `comment-column` — это собственная переменная каждого буфера: установка ее влияет только на текущий буфер, но существует и значение по умолчанию, которое вы также можете изменить с помощью `setq-default`. См. Раздел 31.2.4 [Локальные переменные], с. 350. Многие основные режимы инициализируют эту переменную для текущего буфера.

Команды работы с комментариями распознают комментарии, основываясь на регулярном выражении, которое является значением переменной `comment-start-skip`. Убедитесь, что это регулярное выражение не соответствует пустой строке. Оно может соответствовать чему-то большему, чем просто ограничителю, начинающему комментарий, в самом строгом значении этого слова; например, в режиме C значение этой переменной равно `"/\\*+ *"`, что соответствует дополнительным звездочкам и пробелам после самого ‘`/*`’. (Обратите внимание, ‘`\\`’ требуется в синтаксисе Лиспа для того, чтобы включить в строку ‘`\`’, которая нужна, чтобы отменить для первой звездочки ее специальное значение в синтаксисе регулярных выражений. См. Раздел 12.5 [Регулярные выражения], с. 91.)

Когда команда для комментариев создает новый комментарий, она вставляет в его начало значение `comment-start`. Значение `comment-end` вставляется после точки, так что оно будет следовать за текстом, который вы вставите в этот комментарий. В режиме C `comment-start` имеет значение `"/* "`, а `comment-end` имеет значение `" */"`.

Переменная `comment-multi-line` управляет тем, как ведет себя `C-M-j` (`indent-new-comment-line`) при использовании внутри комментария. Если `comment-multi-line` равна `nil`, как это обычно и бывает, то комментарий на текущей строке завершается, а на новой строке начинается новый комментарий. Если `comment-multi-line` отлична от `nil`, то новая следующая строка подготавливается как часть того же самого комментария, который находился на первой строке. Это выполняется следующим образом: в старой строке не вставляется ограничитель комментария, и в новую строку не вставляеся начало комментария. В тех языках, где работают многострочные комментарии, выбор значений для этой переменной — дело вашего вкуса.

Переменная `comment-indent-function` должна содержать функцию, которая будет вызываться для подсчета отступа во вновь вставляемом комментарии или для выравнивания существующего комментария. Эта функция вызывается без аргумента, но с точкой в начале комментария или в конце строки, если вставляется новый комментарий. Она должна возвратить номер столбца, в котором должен начинаться комментарий. Например, в режиме Lisp эта функция-ловушка для создания отступа основывает свое решение на том, сколько точек с запятой начинают существующий комментарий, и на коде в предыдущих строках.

## 22.8 Редактирование без разбалансированных скобок

`M-(` Поставить скобки вокруг следующего s-выражения (или s-выражений) (`insert-parentheses`).

`M-)` Передвинуться через следующую закрывающую скобку и сделать новый отступ (`move-past-close-and-reindent`).

Команды `M-(` (`insert-parentheses`) и `M-)` (`move-past-close-and-reindent`) созданы для облегчения такого вида редактирования, при котором скобки всегда остаются сбалансированными. `M-(` вставляет пару скобок, либо вместе, как в '`()`', либо, если задан аргумент, вокруг следующих нескольких s-выражений, и оставляет точку после открытой скобки. Точка остается после открывающей скобки. Команда `M-)` перемещается через закрывающую скобку, удаляя любой предшествующий ей отступ и делая после нее отступ при помощи `C-j`.

Например, вместо набора `( F O O )`, вы можете набрать `M-( F O O`, что имеет тот же самый эффект, за исключением того, что курсор остается перед закрывающей скобкой.

`M-(` может вставлять перед открывающей скобкой пробел в зависимости от синтаксического класса предыдущего знака. Установите `parens-require-spaces` в значение `nil`, если вы хотите подавить это.

## 22.9 Завершение для имен символов

Обычно завершение происходит в минибуфере. Но один из видов завершения доступен во всех буферах: завершение для имен символов.

`M-`⟨TAB⟩ (`lisp-complete-symbol`) запускает команду, завершающую частично набранный символ перед точкой, используя множество имен символов, имеющих смысл в этом контексте. Все дополняющие знаки, определяемые по частичному имени, вставляются в точке.

Если частичное имя в буфере имеет более одного возможного завершения, и у них нет общих дополняющих знаков, в другом окне показывается перечень всех возможных завершений.

В большинстве основных режимов для языков программирования, `M-`⟨TAB⟩ запускает команду `complete-symbol`, которая предоставляет два типа завершения. Обычно она делает завершения, основываясь на таблице тегов (см. Раздел 22.13 [Теги], с. 224); с числовым аргументом (независимо от его зачения), она делает завершение, основываясь на именах, перечисленных в указателе понятий в Info-файле для этого языка. Поэтому чтобы завершить имя символа, определенного в вашей собственной программе, используйте `M-`⟨TAB⟩ без аргумента; чтобы завершить имя стандартной библиотечной функции, используйте `C-u M-`⟨TAB⟩. Конечно, основанное на Info завершение работает, только если есть Info-файл для стандартной библиотеки функций вашего языка, и только если он установлен в вашей системе.

В режиме Emacs-Lisp пространство имен для завершения обычно состоит из нетривиальных символов, присутствующих в данный момент в Emacs — тех, что имеют определение функции, значение или свойства. Однако, если непосредственно перед началом частичного символа есть открывающая скобка, в качестве завершений рассматриваются только символы с определением функции. Команда, реализующая это, называется `lisp-complete-symbol`.

В режиме Text и родственных с ним, `M-`⟨TAB⟩ завершает слова, основываясь на словаре программы проверки правописания. См. Раздел 13.4 [Правописание], с. 102.

## 22.10 Режим Which Function

Режим Which Function — это второстепенный режим, который показывает в строке режима имя текущей функции по мере того, как вы передвигаетесь по буферу.

Чтобы включить (или выключить) режим Which Function, используйте команду `M-x which-function-mode`. Это глобальная команда; она применяется ко всем буферам, как к существующим, так и к тем, что еще будут созданы. Однако, это затрагивает только определенные основные режимы, перечисленные в значении `which-func-modes`. (Если это

значение `t`, то режим Which Function применяется ко всем основным режимам, которые знают, как это поддерживается — к основным режимам, поддерживающим Imenu.)

## 22.11 Команды документации

Когда вы редактируете код на Лиспе, предназначенный для запуска в Emacs, вы можете использованы команды `C-h f` (`describe-function`) и `C-h v` (`describe-variable`) для печати документации о функциях и переменных, которые вы хотите вызвать. Эти команды используют минибуфер для считывания имени функции или переменной и показывают документацию в окне.

Для большего удобства эти команды предоставляют аргументы по умолчанию, основанные на коде в окрестности точки. `C-h f` устанавливает значение по умолчанию равным функции, вызванной в списке самого глубокого уровня, содержащем точку. `C-h v` использует в качестве значения по умолчанию имя символа, находящегося вокруг или рядом с точкой.

Для кода на Emacs Lisp вы также можете использовать режим Eldoc. Этот второстепенный режим постоянно показывает в эхо-области список аргументов для функции, которая вызывается в точке. (Другими словами, он находит вызов функции, который содержит точку, и показывает список аргументов этой функции.) Режим Eldoc применим только к режимам Emacs Lisp и Lisp Interaction. Для включения и выключения этого режима используйте команду `M-x eldoc-mode`.

Для Си, Лиспа и других языков вы можете использовать `C-h C-i` (`info-lookup-symbol`), чтобы просмотреть документацию Info по какому-то символу. Вы задаете символ в минибуфере; по умолчанию берется символ, находящийся в буфере в точке. Где искать документацию по символам — в каких Info-файлах и каких именных указателях — определяет основной режим. Вы можете также использовать `M-x info-lookup-file` для нахождения документации для имени файла.

Вы можете прочитать “страницу man” для команды операционной системы, библиотечной функции или системного вызова с помощью команды `M-x manual-entry`. Для форматирования страницы она запускает программу `man` и, если позволяет ваша операционная система, делает это асинхронно, чтобы вы могли продолжать редактирование, пока страница форматируется. (MS-DOS и MS-Windows 3 не допускают асинхронных подпроцессов, так что на этих системах вы не можете редактировать, когда Emacs ожидает, пока `man` закончит работу.) Результат направляется в буфер с именем ‘`*Man` *тема*`*`’. Эти буферы используют особый основной режим, режим Man, который облегчает прокрутку и просмотр других страниц man. Для получения подробностей наберите `C-h m` в буфере страницы man.

Для длинных страниц правильная установка начертаний может занять значительное время. По умолчанию Emacs использует в страницах man начертания, если может показывать разные шрифты или цвета. Вы можете выключить использование разных начертаний в страницах man, установив переменную `Man-fontify-manpage-flag` равной `nil`.

Если вы вставите текст страницы man в буфер Emacs каким-то другим способом, вы можете использовать команду `M-x Man-fontify-manpage`, чтобы произвести те же преобразования, что делает `M-x manual-entry`.

Проект GNU надеется когда-нибудь заменить большинство страниц man на лучше организованные руководства, которые вы можете просматривать с помощью Info. См. Раздел 7.7 [Другие справки], с. 60. Поскольку этот процесс завершен лишь частично, читать страницы man все еще полезно.

## 22.12 Журналы изменений

Команда Emacs `C-x 4 a` добавляет в журнал изменений новую запись для файла, который вы редактируете (`add-change-log-entry-other-window`).

Файл журнала изменений содержит хронологическое описание того, почему и когда вы изменяли программу, состоящее из последовательности записей, описывающих отдельные изменения. Как правило оно хранится в файле с именем '`ChangeLog`' в том же самом каталоге, в котором находится файл, который вы редактируете, или в одном из его родительских каталогов. Единственный файл '`ChangeLog`' может записывать изменения для всех файлов в его каталоге и во всех его подкаталогах.

Запись в журнале изменений начинается со строки заголовка, которая содержит ваше имя, ваш адрес электронной почты (получаемый из переменной `user-mail-address`) и текущую дату и время. Кроме этих строк заголовка, каждая строка в журнале изменений начинается с пробела или табуляции. Основная часть записи состоит из *пунктов*, каждый из которых начинается со строки, начинающейся с пропуска и звездочки. Вот пример двух записей, обе датированы маем 1993 года и обе содержат два пункта:

```
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.
```

(Предыдущие версии Emacs использовали другой формат даты.)

Одна запись может описывать несколько изменений; каждое изменение должно описываться в отдельном пункте. Обычно между пунктами должна быть пустая строка. Когда пункты связаны между собой (части одного изменения в разных местах), группируйте их, не оставляя между ними пустую строку. Вторая запись выше содержит два пункта, сгруппированных таким способом.

`C-x 4 a` обращается к файлу журнала изменений и создает новую запись, если только последний по времени пункт не датирован сегодняшним днем и не несет ваше имя. Также она создает новый пункт для текущего файла. Для многих языков она может даже предположить имя измененной функции или объекта.

К файлу журнала изменений обращаются в режиме Change Log. В этом основном режиме каждая связка сгруппированных пунктов считается одним абзацем, а каждая запись считается страницей. Это облегчает редактирование записей. `C-j` и автоматическое заполнение делают в каждой новой строке такой же отступ, как в предыдущей; это удобно для ввода содержимого записей.

Системы управления версиями дают другой способ отслеживания изменений в вашей программе и ведения журнала изменений. См. Раздел 14.7.3.3 [Буфер журнала], с. 119.

## 22.13 Таблицы тегов

*Таблица тегов* — это описание того, как многофайловая программа разбивается на файлы. Она перечисляет имена файлов-компонентов и имена и позиции функций (или других

именованных подъединиц) в каждом файле. Объединение связанных файлов делает возможным поиск или замену во всех файлах с помощью одной команды. Запись имен функций и позиций делает возможной команду `M-.`, которая находит определение, отыскивая сведения о том, в каком файле оно находится.

Таблицы тегов хранятся в файлах, именуемых *файлами таблиц тегов*. Общепринятое имя для файла таблицы тегов — '`TAGS`'.

Каждый элемент в таблице тегов записывает имя одного тега, имя файла, в котором этот тег определен (явно), и местоположение определения тега в этом файле.

Какие именно имена из описанных файлов записываются в таблице тегов, зависит от языка программирования описанного файла. Обычно они включают все функции и подпрограммы, могут также включать глобальные переменные, типы данных и что-нибудь еще относящееся к делу. Каждое записанное имя называется *тегом*.

### 22.13.1 Синтаксис тегов исходного файла

В наиболее популярных языках синтаксис тегов определяется следующим образом:

- В программе на Си, любая функция Си или typedef — это тег, тегом являются и определения `struct`, `union` и `enum`. Определения макросов (`#define`) и констант (`enum`) также являются тегами, если только вы не задали при создании таблицы тегов ключ '`-no-defines`'. Аналогично, тегами являются глобальные переменные, если только вы не задали ключ '`-no-globals`'. Использование '`-no-globals`' и '`-no-defines`' может сделать файлы таблиц тегов гораздо меньше.
- В коде на Си++, помимо всех тегов кода Си распознаются также функции-члены и, возможно, переменные-члены, если вы используете ключ '`-members`'. Теги для переменных и функций в классах именуются как '*класс*`::`*переменная*' и '*класс*`::`*функция*'.
- В коде на Java, теги включают все конструкции, распознаваемые в Си++ плюс конструкции `extends` и `implements`. Теги для переменных и функций в классах именуются как '*класс*`.`*переменная*' и '*класс*`.`*функция*'.
- В тексте для LaTeX, тегами служат аргументы каждой из команд `\chapter`, `\section`, `\subsection`, `\subsubsection`, `\eqno`, `\label`, `\ref`, `\cite`, `\bibitem`, `\part`, `\appendix`, `\entry` или `\index`.

  Другие команды тоже могут создавать теги, если вы укажете их в переменной среды `TEXTAGS` перед вызовом `etags`. Значением этой переменной среды должен быть разделенный двоеточиями список имен команд.

  ```
  TEXTAGS="def:newcommand:newenvironment"
  export TEXTAGS
  ```

  задает (с использованием синтаксиса Bourne shell), что команды '`\def`', '`\newcommand`' и '`\newenvironment`' также определяют теги.
- В коде на Лиспе любая функция, определенная через `defun`, любая переменная, определенная через `defvar` или `defconst`, и вообще первый аргумент любого выражения, которое начинается с '`(def`' в нулевом столбце, являются тегом.
- В коде на Scheme теги включают все определяемое с помощью `def` или конструкции, чье имя начинается с '`def`'. Они также включают переменные, установленные с помощью `set!` на верхнем уровне файла.

Поддерживаются также несколько других языков:

- В коде ассемблера, теги — это метки, появляющиеся в начале строки, за которыми идет двоеточие.
- Во входных файлах Bison или Yacc каждое правило определяет конструируемый им нетерминал как тег. Части файла, содержащие код на Си, анализируются как код Си.

- В коде на Cobol тегами служат имена параграфов; то есть любые слова, которые начинаются в столбце 8, и после которых стоит точка.
- В коде на Erlang тегами служат определенные в файле функции, записи и макросы.
- В Фортран-коде тегами являются функции, подпрограммы и блоки данных.
- В коде на Паскале тегами будут определенные в файле функции и процедуры.
- В коде на Perl тегами являются процедуры, определяемые ключевым словом `sub`.
- В коде на Postscript тегами являются функции.
- В коде на Прологе теги появляются на левой границе.

Вы также можете генерировать теги, основываясь на сопоставлении регулярных выражений (см. Раздел 22.13.2 [Создание таблицы тегов], с. 226), чтобы обработать другие форматы и языки.

### 22.13.2 Создание таблицы тегов

Для создания файла таблицы тегов используется программа `etags`. Она знает несколько языков, как описано в предыдущем разделе. `etags` запускается следующим образом:

`etags` *входные-файлы*...

Программа `etags` считывает указанные файлы и записывает таблицу тегов под именем '`TAGS`' в текущем рабочем каталоге. `etags` распознает язык, используемый во входном файле, основываясь на имени этого файла и его содержании. Вы можете указать язык с помощью ключа '`-language=`*имя*', описанного ниже.

Если данные таблицы тегов становятся устаревшими из-за изменений в описанных в таблице файлах, то таблица тегов обновляется тем же способом, что был применен для ее начального создания. Нет необходимости делать это часто.

Если таблица тегов не в состоянии записать тег или записывает его не для того файла, то Emacs может не найти его определение. Однако, если позиция, записанная в таблицу тегов, становится немного неверной (из-за некоторого редактирования в файле, в котором находится определение этого тега), то единственным следствием будет слегка замедленный поиск тега. Даже если хранящаяся позиция совсем неправильна, Emacs все-таки найдет тег, но для этого он должен будет обследовать весь файл.

Таким образом, вам нужно обновлять таблицу тегов, когда вы определяете новые теги, которые вы хотите внести в список, или когда вы перемещаете определения тегов из одного файла в другой, или когда изменения становятся существенными. Обычно нет нужды обновлять таблицу тегов после каждого редактирования или даже каждый день.

Одна таблица тегов может как бы включать другую. Имя включаемого файла тегов указывается с помощью ключа '`-include=`*файл*' при создании включающего файла. Последний файл затем ведет себя так, как если бы он содержал все файлы, заданные во включенном файле, так же как и те файлы, которые он содержит непосредственно.

Если при запуске `etags` вы зададите исходные файлы по относительным именам, файл тегов будет содержать имена файлов, относительные к каталогу, в котором этот файл тегов был изначально записан. Тогда вы сможете переместить все дерево каталогов, содержащее и файл тегов, и исходные файлы, и файл тегов все равно будет правильно ссылаться на исходные файлы.

Если в качестве аргументов `etags` вы зададите абсолютные имена файлов, то файл тегов будет содержать абсолютные имена. Тогда файл тегов будет так же ссылаться на те же исходные файлы, даже если вы переместите его, до тех пор, пока исходные файлы остаются на старом месте. Абсолютные имена файлов начинаются с '`/`', или с '*устройство*`:/`' в MS-DOS и MS-Windows.

Когда вы хотите создать таблицы тегов для очень большого числа файлов, у вас могут возникнуть проблемы с их перечислением в командной строке, поскольку некоторые системы накладывают ограничение на ее длину. Простейший способ обойти это ограничение — сказать `etags` считывать имена файлов со стандартного ввода, набрав дефис на месте имен файлов, как здесь:

```
find . -name "*.[chCH]" -print | etags -
```

Используйте ключ '`-language=`*имя*' для явного указания языка. Вы можете перемешивать эти ключи с именами файлов; каждый относится к имена файла, которое следует за ним. Задайте '`-language=auto`', чтобы велеть `etags` продолжать самой предполагать язык по имени и содержимому файла. Задайте '`-language=none`', чтобы полностью выключить специфичную для языка обработку; тогда `etags` распознает теги только по сопоставлению с регулярным выражением. '`etags -help`' печатает перечень языков, которые знает `etags`, и правила предположения языка по имени файла.

Ключ '`-regex`' предоставляет общий способ распознавания тегов, основаный на сопоставлении с регулярным выражением. Вы можете свободно перемешивать эти ключи с именами файлов. Каждый ключ '`-regex`' добавляется к предшествующим и применяется только к последующим файлам. Синтаксис таков:

`-regex=/`*regexp-тег*`[/`*regexp-имя*`]/`

где *regexp-тег* используется для нахождения строк тегов. Оно всегда зацепленное, то есть ведет себя так, как если бы в начале стояло '`^`'. Если вы хотите учесть отступы, просто назовите совпадением произвольное количество пропусков, начав ваше регулярное выражение с '`[ \t]*`'. Знак '`\`' в регулярных выражениях экранирует следующий знак, а '`\t`' обозначает символ табуляции. Обратите внимание, `etags` не обрабатывает другие управляющие последовательности Си для специальных знаков.

`etags` придерживается того же синтаксиса регулярных выражений, что и Emacs, но с введением *оператора интервала*, который работает как в `grep` и `ed`. Синтаксис оператора интервала такой: '`\{`*m*`,`*n*`\}`', это означает, что нужно найти совпадение с предыдущим выражением по меньшей мере *m* раз и вплоть до *n* раз.

*regexp-тег* не должно совпадать с большим числом знаков, чем это необходимо для распознавания нужного вам тега. Если соответствие таково, что *regexp-тег* неизбежно совпадает с большим, чем нужно, числом знаков, вы можете найти полезным добавить *regexp-имя*, чтобы сузить область тега. Вы можете найти примеры ниже.

Ключ '`-R`' удаляет все регулярные выражения, определенные ключами '`-regex`'. Он применяется к следующим за ним именам файлов, как вы можете видеть из следующего примера:

```
etags -regex=/reg1/ voo.doo -regex=/reg2/ \
    bar.ber -R -lang=lisp los.er
```

Здесь `etags` выбирает язык для анализа '`voo.doo`' и '`bar.ber`' в соответствии с их содержимым. `etags` также использует *reg1* для распознавания дополнительных тегов в '`voo.doo`' и оба выражения *reg1* и *reg2* для распознавания дополнительных тегов в '`bar.ber`'. Для распознавания тегов в '`los.er`' `etags` использует правила тегов для Лиспа и не использует регулярные выражения.

Вот еще несколько примеров. Регулярные выражения взяты в кавычки, чтобы оболочка не интерпретировала их по-своему.

- Сделать теги для макроса `DEFVAR` в исходных файлах Emacs:

  ```
  -regex='/[ \t]*DEFVAR_[A-Z_ \t(]+"\([^"]+\)"/'
  ```

- Сделать теги для VHDL-файлов (этот пример — одна строка, разбитая здесь для правильного форматирования):

  ```
  -language=none
  -regex='/[ \t]*\(ARCHITECTURE\|CONFIGURATION\) +[^ ]* +OF/'
  ```

```
-regex='/[ \t]*\(ATTRIBUTE\|ENTITY\|FUNCTION\|PACKAGE\
\( BODY\)?\|PROCEDURE\|PROCESS\|TYPE\)[ \t]+\([^ \t(]+\)/\3/'
```

- Сделать теги для файлов на Tcl (этот последний пример показывает использование аргумента *regexp-имя*):

```
-lang=none -regex='/proc[ \t]+\([^ \t]+\)/\1/'
```

Чтобы получить перечень других доступных ключей `etags`, выполните `etags --help`.

### 22.13.3 Выбор таблицы тегов

Emacs хранит в каждый момент одну *выбранную* таблицу тегов, и все команды для работы с таблицами тегов используют эту выбранную таблицу. Чтобы выбрать таблицу тегов, наберите `M-x visit-tags-table`, которая считает имя файла таблицы тегов как аргумент. Имя '`TAGS`' в каталоге по умолчанию используется как имя файла по умолчанию.

Все, что делает эта команда, — сохраняет имя файла в переменной `tags-file-name`. Emacs фактически не считывает содержимое таблицы тегов до тех пор, пока вы не попытаетесь использовать его. Самостоятельная установка этой переменной так же хороша, как и использование `visit-tags-table`. Начальное значение переменной равно `nil`; это значение сообщает всем командам для работы с таблицами тегов, что они должны запрашивать, какое имя файла таблицы тегов надо использовать.

Использование `visit-tags-table`, когда таблица тегов уже загружена, дает вам выбор: вы можете добавить новую таблицу тегов к текущему списку таких таблиц или начать новый список. Команды работы с тегами используют все таблицы тегов в текущем списке. Если вы начинаете новый список, новая таблица тегов используется *вместо* остальных. Если вы добавляете новую таблицу тегов к текущему списку, она используется *вместе* с остальными. Когда команды работы с тегами сканируют список таблиц тегов, они не всегда начинают с начала списка; они начинают с первой таблицы, которая описывает текущий файл (если такая есть), проходят далее до конца списка и затем просматривают список с начала до тех пор, пока в нем не будут проверены все таблицы.

Вы можете явно задать список таблиц тегов, установив переменную `tags-table-list` в значение списка строк, как показано:

```
(setq tags-table-list
      '("~/emacs" "/usr/local/lib/emacs/src"))
```

Это заставляет команды, работающие с тегами, просматривать файлы '`TAGS`' в каталогах '`~/emacs`' и '`/usr/local/lib/emacs/src`'. Порядок зависит от того, в каком файле вы сейчас находитесь и какая таблица тегов упоминает этот файл, как обяснено выше.

Не установливайте переменные `tags-file-name` и `tags-table-list` одновременно.

### 22.13.4 Поиск определения тега

Самая важная вещь, которую вам позволяют делать таблицы тегов, — это поиск определения конкретного тега.

`M-.` *тег* ⟨RET⟩
: Найти первое определение *тега* (`find-tag`).

`C-u M-.`
: Найти следующее по очереди определение последнего заданного тега.

`C-u - M-.`
: Вернуться к предыдущему найденному тегу.

`C-M-.` *образец* ⟨RET⟩
: Найти тег, чье имя совпадает с *образцом* (`find-tag-regexp`).

`C-u C-M-.`
: Найте следующий тег, чье имя совпадает с последним использованным образцом.

`C-x 4 .` *тег* ⟨RET⟩
: Найти первое определение *тега*, но показать его в другом окне (`find-tag-other-window`).

`C-x 5 .` *тег* ⟨RET⟩
: Найти первое определение *тега* и создать новый фрейм для выбора буфера (`find-tag-other-frame`).

`M-*`
: Вернуться к тому месту, где вы ранее вызвали `M-.` и товарищей.

`M-.` (`find-tag`) — это команда для поиска определения заданного тега. Она ищет его по таблице тегов как строку и затем использует эту информацию из таблицы тегов для того, чтобы определить файл, в котором находится определение, и приблизительную положение определения в файле. Затем `find-tag` обращается к этому файлу, передвигает точку в приблизительную позицию и начинает поиск определения на постоянно возрастающем расстоянии.

Если задается пустой аргумент (просто ⟨RET⟩), то в качестве имени тега, который надо найти, используется s-выражение, находящееся в буфере перед или вокруг точки. Для получения информации о s-выражениях смотрите Раздел 22.2 [Списки], с. 206,

Аргумент для `M-.` не обязан быть полным именем тега; достаточно части. Это возможно, потому что `M-.` находит в таблице теги, которые содержат *тег* как построку. Однако, она предпочитает точное совпадение совпадению лишь построки. Чтобы найти другие теги, которые соответствуют той же подстроке, следует дать `find-tag` числовой аргумент, как в `C-u M-.`; эта команда не считываает имя тега, но продолжает поиск по тексту таблицы тегов другого тега, содержащего самую последнюю использованную подстроку. Если у вас есть настоящая клавиша ⟨META⟩, то `M-0 M-.` может служить более простой альтернативой `C-u M-.`.

Подобно большинству команд, которые могут переключать буферы, `find-tag` имеет вариант, который показывает новый буфер в другом окне, и еще один, который создает новый фрейм. Первая команда — это `C-x 4 .`, которая вызывает функцию `find-tag-other-window`. Вторая, `C-x 5 .`, вызывает `find-tag-other-frame`.

Чтобы вернуться к местам, где вы недавно находили теги, используйте `C-u - M-.`; в более общем виде, `M-.` с отрицательным числовым аргументом. Эта команда может перенести вас в другой буфер. `C-x 4 .` с отрицательным аргументом находит предыдущее положение тега в другом окне.

Так же, как вы можете вернуться к местам, где вы недавно находили теги, вы можете вернуться к местам, *откуда* вы их нашли. Используйте для этого `M-*`, что вызывает команду `pop-tag-mark`. Типичное применение этих команд — найти и изучить определение чего-то с помощью `M-.` и затем вернуться к тому месту, где вы были, с помощью `M-*`.

И `C-u - M-.`, и `M-*` позволяют вам пошагово проходить назад до глубины, определяемой переменной `find-tag-marker-ring-length`.

Команда `C-M-.` (`find-tag-regexp`) обращается к тегам, соответствующим заданному регулярному выражению. Она похожа на `M-.`, но производит сопоставление с регулярным выражением, а не со строкой.

### 22.13.5 Поиск и замена при помощи таблиц тегов

Команды этого раздела обращаются и просматривают все файлы, перечисленные в выбранной таблице тегов, один за другим. Таблица тегов служит для этих команд только для того, чтобы определить последовательность поиска в файлах.

`M-x tags-search` ⟨RET⟩ *regexp* ⟨RET⟩
: Поиск *regexp* во всех файлах в выбранной таблице тегов.

`M-x tags-query-replace` ⟨RET⟩ *regexp* ⟨RET⟩ *замена* ⟨RET⟩
: Осуществить `query-replace-regexp` в каждом файле в выбранной таблице тегов.

`M-,`
: Перезапустить одну из вышеупомянутых команд из текущего положения точки (`tags-loop-continue`).

`M-x tags-search` считывает регулярное выражение, используя минибуфер, затем ищет это регулярное выражение по очереди в каждом файле из выбранной таблицы тегов. Она показывает имя файла, который в данный момент просматривается, таким образом, вы можете следить за ходом поиска. Как только определяется местонахождение, `tags-search` возвращается.

Найдя одно соответствие, вы, вероятно, захотите найти все остальные. Чтобы найти еще одно соответствие, наберите `M-,` (`tags-loop-continue`), это возобновит `tags-search`. Эта команда просматривает остаток текущего буфера и затем оставшиеся файлы таблицы тегов.

`M-x tags-query-replace` осуществляет во всех файлах в таблице тегов единую замену регулярного выражения с подтверждением. Она считывает регулярное выражение, которое следует искать, и строку для замены, точно так же, как обычная `M-x query-replace-regexp`. Она ищет очень похоже на `M-x tags-search`, но с повторами, обрабатывая совпадения согласно вашему вводу. См. Раздел 12.7 [Замена], с. 95, более подробную информацию о замене с подтверждением.

Можно пройти по всем файлам в таблице тегов с помощью едиственного вызова `M-x tags-query-replace`. Но иногда бывает полезно временно выйти, что вы можете сделать с помощью любого события ввода, не имеющего особого смысла при замене с подтверждением. Вы можете впоследствии возобновить замену с подтверждением, набрав `M-,`; эта команда возобновляет последнюю сделанную вами команду поиска или замены тегов.

Команды этого раздела приводят к гораздо более широкому поиску, чем семейство `find-tag`. Команды `find-tag` ищут только определения тегов, совпадающих с вашей подстрокой или регулярным выражением. Команды `tags-search` и `tags-query-replace` находят каждое вхождение регулярного выражения, как делают в текущем буфере обычные команды поиска и замены.

Эти команды создают буферы только временно, для файлов, в которых они должны делать поиск (для тех, к которым уже не обращается какой-нибудь буфер Emacs). Буферы, в которых нет совпадений, быстро уничтожаются; остальные продолжают существовать.

Вас, возможно, поразило, что `tags-search` очень похожа на `grep`. Вы можете также запустить саму `grep` как подчиненную Emacs, и Emacs покажет вам совпадающие строки одну за другой. Это работает во многом похоже на запуск компиляции; обращение к тем позициям в исходных файлах, где `grep` нашла совпадения, работает как обращение к ошибкам компиляции. См. Раздел 23.1 [Компиляция], с. 247.

### 22.13.6 Запросы к таблице тегов

`M-x list-tags` ⟨RET⟩ *файл* ⟨RET⟩
: Показать список тегов, определенных в заданном *файле* с текстом программы.

`M-x tags-apropos` ⟨RET⟩ *regexp* ⟨RET⟩
: Показать список всех тегов, соответствующих регулярному выражению *regexp*.

`M-x list-tags` считывает имя одного из файлов, описанных в выбранной таблице тегов, и показывает список всех тегов, определенных в этом файле. Аргумент “имя файла” фактически является просто строкой для сравнения с именами, записанными в таблице тегов; он считывается как строка, а не как имя файла. Поэтому завершение и значение по умолчанию невозможны, и вы должны вводить имя файла в том же самом виде, в котором

оно появляется в таблице тегов. Не включайте каталог как часть имени файла, если имя файла, записанного в таблице тегов, не включает каталог.

`M-x tags-apropos` похожа на `apropos` для тегов (см. Раздел 7.3 [Контекстная справка], с. 57). Она считывает регулярное выражение, затем находит все теги в выбранной таблице тегов, чьи вхождения соответствуют этому регулярному выражению, и показывает найденные имена тегов.

Вы также можете производить в буфере завершение в пространстве имен, составленном из имен тегов текущих таблиц. См. Раздел 22.9 [Завершение символов], с. 222.

## 22.14 Объединение файлов с помощью Emerge

Нередко программисты перебегают друг другу дорогу и изменяют одну и ту же программу в двух разных направлениях. Чтобы справиться с этой путаницей, вам необходимо объединить две эти версии. Emerge упрощает это. Смотрите также Раздел 14.9 [Сравнение файлов], с. 132, о командах для сравнения файлов более ручным методом, и раздел “Emerge” в *The Ediff Manual*.

### 22.14.1 Обзор Emerge

Чтобы запустить Emerge, выполните одну из этих четырех команд:

`M-x emerge-files`
: Объединяет два заданных файла.

`M-x emerge-files-with-ancestor`
: Объединяет два заданных файла, со ссылкой на общего предка.

`M-x emerge-buffers`
: Объединяет два буфера.

`M-x emerge-buffers-with-ancestor`
: Объединяет два буфера со ссылкой на общего предка в третьем буфере.

Команды Emerge сравнивают два файла или буфера и отображают результат сравнения в трех буферах: по одному на каждый входной файл (*буфер A* и *буфер B*) и один (*буфер объединения*), где объединение и происходит. Буфер объединения показывает весь объединяемый текст, а не только различия. Везде, где буферы различаются, вы можете выбрать тот, из которого вы хотите внести фрагмент.

Команды Emerge, которые принимают ввод из существующих буферов, используют только их доступные части, если эти буферы сужены (см. Раздел 30.8 [Сужение], с. 335).

Если доступна общая начальная версия, от которой происходят оба сливаемых текста, Emerge может использовать ее, чтобы вывести предположение о том, какая из альтернатив правильна. Когда одна из текущих версий находится в согласии с предком, Emerge предполагает, что другая текущая версия — это обдуманное изменение, которое должно сохраниться в объединенной версии. Если вы хотите указать общий начальный текст, используйте команды ‘`with-ancestor`’. Эти команды считывают три файла или имени буфера — вариант A, вариант B и их общего предка.

После того как сравнение завершено, и буферы подготовлены, начинается интерактивное объединение. Вы можете управлять им, набирая особые *команды объединения* в буфере объединения. Этот буфер показывает вам полный объединенный текст, а не только различия. Для каждого промежутка различий между входными текстами вы можете сохранить любой или отредактировать их вместе.

В буфере объединения используется особый основной режим, режим Emerge, с командами для принятия таких решений. Но вы также можете редактировать этот буфер с помощью обычных команд Emacs.

В любой момент времени внимание Emerge сосредоточено на одном конкретном изменении, называемом *выделенным*. Это изменение помечается в трех буферах таким образом:

```
vvvvvvvvvvvvvvvvvvvv
различающийся текст
^^^^^^^^^^^^^^^^^^^^
```

Emerge последовательно нумерует все различия, и строка режима всегда показывает номер выделенного различия.

Обычно буфер объединения изначально содержит версию A текста. Но когда версия A изменения согласуется с общим предком, для этого изменения предпочтение отдается версии B.

Когда вы выходите, Emerge оставляет объединенный текст в буфере объединения. Тогда вы можете сохранить его с помощью `C-x C-w`. Если вы задали `emerge-files` или `emerge-files-with-ancestor` числовой аргумент, эти команды считает имя выходного файла в минибуфере. (Это последнее имя, которое они считывают.) Тогда при выходе из Emerge объединенный текст сохраняется в выходном файле.

Обычно команды Emerge сохраняет выходной буфер, когда вы выходите. Если вы прервете Emerge с помощью `C-]`, команда Emerge не сохранит выходной буфер, но вы можете записать его сами, если хотите.

### 22.14.2 Подрежимы Emerge

Вы можете выбирать из двух режимов для отдания команд объединения: режим Fast и режим Edit. В режиме Fast основные команды объединения — это одиночные знаки, но обычные команды Emacs выключены. Это удобно, если вы используете только команды объединения. В режиме Edit все команды объединения начинаются с префикса `C-c C-c`, и доступны также обычные команды Emacs. Это позволяет редактировать буфер объединения, но замедляет операции Emerge.

Используйте `e`, чтобы переключится в режим Edit, и `C-c C-c f`, чтобы переключится в режим Fast. Строка режима изображает режимы Edit и Fast при помощи ‘`E`’ и ‘`F`’.

Emerge имеет два дополнительных подрежима, которые затрагивают способ работы определенных команд объединения: режим Auto Advance и режим Skip Prefers.

Если действует режим Auto Advance, команды `a` и `b` продвигают к следующему различию. Это позволяет вам быстрее сделать объединение, поскольку вы просто выбираете одну из входных альтернатив. Строка режима изображает режим Auto Advance как ‘`A`’.

Если действует режим Skip Prefers, команды `n` и `p` пропускают различия в состояниях prefer-A и prefer-B (см. Раздел 22.14.3 [Состояние различия], с. 232). Таким образом, вы видите только те различия, для которых ни одна из версий не предполагается “правильной”. Строка режима изображает режим Skip Prefers с помощью ‘`S`’.

Используйте команду `s a` (`emerge-auto-advance-mode`), чтобы установить или сбросить режим Auto Advance. Используйте `s s` (`emerge-skip-prefers-mode`), чтобы установить или сбросить режим Skip Prefers. Эти команды включают режим, если им задан положительный аргумент, выключают при отрицательном аргументе и переключают режим, если аргумент не задан.

### 22.14.3 Состояние различия

Различия в буфере объединения помечаются строками знаков ‘`v`’ и ‘`^`’. Каждое различие имеет одно из семи следующих состояний:

A Различие показывает версию A. Команда `a` всегда производит это состояние; строка режима отображает его как ‘`A`’.

B
: Различие показывает версию B. Команда `b` всегда производит это состояние; строка режима отображает его как ‘`B`’.

default-A
default-B
: Различие показывает версию A или B по умолчанию, потому что вы не сделали выбор. Все различия изначально имеют состояние default-A (и таким образом, буфер объединения — это копия буфера A), кроме тех, для которых “предпочтительна” другая альтернатива (смотрите ниже).

  Когда вы выбираете различие, его состояние заменяется из default-A или default-B на простое A или B. Таким образом, выбранное различие никогда не находится в состоянии default-A или default-B, и эти состояния никогда не отображаются в строке режима.

  Команда `d a` выбирает в качестве состояния по умолчанию default-A, а `d b` выбирает default-B. Это состояние по умолчанию применяется ко всем различиям, которые вы никогда не выбирали и для которых нет предпочтительной альтернативы. Если вы продвигаетесь последовательно, то различия, которые вы не выбирали, — это те, что находятся после выбранного. Таким образом, продвигаясь последовательно, вы можете в результате сделать A версией по умолчанию для некоторых фрагментов буфера объединения, а B — версией по умолчанию для остальных фрагментов, используя между различиями `d a` и `d b`.

prefer-A
prefer-B
: Различие показывает состояние A или B, потому что оно *предпочтительно*. Это означает, что вы не сделали явного выбора, но похоже на то, что верна одна из альтернатив, так как вторая согласуется с общим предком. Значит, когда буфер A находится в согласии с общим предком, предпочтительна версия B, потому что скорее всего это и есть действительное изменение.

  Эти состояния показываются в строке режима как ‘`A*`’ и ‘`B*`’.

combined
: Различие показывает комбинацию состояний A и B, как результат команд `x c` или `x C`.

  Когда различие имеет это состояние, команды `a` и `b` не делают для него ничего, если только вы не зададите им числовой аргумент.

  Строка режима показывает это состояние как ‘`comb`’.

### 22.14.4 Команды объединения

Это команды объединения для режима Fast; в режиме Edit предваряйте их набором `C-c C-c`:

`p`
: Выбирает предыдущее различие.

`n`
: Выбирает следующее различие.

`a`
: Выбирает версию A этого различия.

`b`
: Выбирает версию B этого различия.

`C-u` *n* `j`
: Выбирает различие номер *n*.

`.`
: Выбирает различие, содержащее точку. Вы можете использовать эту команды в буфере объединения или в буферах A и B.

`q`
: Выход — завершает объединение.

`C-]`
: Прерывание — выходит и не сохраняет вывод.

`f`
: Переход в режим Fast. (В режиме Edit это на самом деле `C-c C-c f`.)

| | |
|---|---|
| `e` | Переход в режим Edit. |
| `l` | Центрирует (как `C-l`) все три окна. |
| `-` | Задает часть префиксного числового аргумента. |
| *цифра* | Также задает часть префиксного числового аргумента. |
| `d a` | Выбирает A как версию по умолчанию для буфера объединения начиная с этого места. |
| `d b` | Выбирает B как версию по умолчанию для буфера объединения начиная с этого места. |
| `c a` | Копирует версию A этого различия в список уничтожений. |
| `c b` | Копирует версию B этого различия в список уничтожений. |
| `i a` | Вставляет версию A этого различия в точке. |
| `i b` | Вставляет версию B этого различия в точке. |
| `m` | Помещает точку и метку вокруг этого различия. |
| `^` | Прокручивает все три окна вниз (как `M-v`). |
| `v` | Прокручивает все три окна вверх (как `C-v`). |
| `<` | Прокручивает все три окна влево (как `C-x <`). |
| `>` | Прокручивает все три окна вправо (как `C-x >`). |
| `\|` | Переустанавливает горизонтальную прокрутку во всех трех окнах. |
| `x 1` | Сужает окно объединения до одной строки. (Используйте `C-u l`, чтобы восстановить его полный размер.) |
| `x c` | Комбинирует две версии этого различия (см. Раздел 22.14.6 [Комбинирование в Emerge], с. 235). |
| `x f` | Показывает имена файлов/буферов, с которыми работает Emerge, в окне справки. (Используйте `C-u l`, чтобы восстановить окна.) |
| `x j` | Сцепляет это различие со следующим. (`C-u x j` сцепляет это различие с предыдущим.) |
| `x s` | Разбивает это различие на два. До того, как применить эту команду, расположите точку в каждом их буферов в том месте, где вы хотите разбить различие. |
| `x t` | Выбрасывает одинаковые строки сверху и снизу различия. Такие строки появляются, когда версии A и B идентичны, но отличаются от первоначальной версии. |

### 22.14.5 Выход из Emerge

Команда `q` (`emerge-quit`) завершает объединение, записывая результаты в выходной файл, если вы его задали. Она восстанавливает правильное содержимое буферов A и B или уничтожает их, если они были созданы Emerge, и вы не изменяли их. Она также выключает в буфере объединения команды Emerge, поскольку выполнение их теперь может повредить содержимое различных буферов.

`C-]` прерывает объединение. Это означает выход без записи выходного файла. Если вы не указали выходной файл, то между прерыванием и завершением объединения на самом деле нет разницы.

Если команды Emerge были вызваны из другой программы на Лиспе, то в случае успешного завершения возвращается значение `t`, а если вы прервали объединение, возвращается `nil`.

### 22.14.6 Комбинирование двух версий

Иногда вы хотите сохранить *оба* варианта некоторого изменения. Чтобы сделать так, используйте `x c`, которая редактирует буфер объединения следующим образом:

```
#ifdef NEW
версия из буфера A
#else /* not NEW */
версия из буфера B
#endif /* not NEW */
```

Хотя этот пример показывает условные конструкции препроцессора Си, разделяющие два альтернативные версии, вы можете задать используемые строки, устанавливая переменную `emerge-combine-versions-template` по вашему выбору. В этой строке '`%a`' говорит, где нужно помещать версию A, а '`%b`' говорит, говорит помещать версию B. Установка по умолчанию, которая выдает результат, показанный выше, выглядит следующим так:

```
"#ifdef NEW\n%a#else /* not NEW */\n%b#endif /* not NEW */\n"
```

### 22.14.7 Тонкие вопросы, связанные с Emerge

В процессе объединения вы не должны пытаться редактировать буферы A и B сами. Emerge временно изменяет их, но в конце возвращает в исходное состояние.

Вы можете производить любое число объединений одновременно — только не используйте один и тот же буфер в качестве входного более чем для одного объединения, так как временные изменения, сделанные в этих буферах, столкнулись бы друг с другом.

Запуск Emerge может занять продолжительное время, поскольку ей требуется полностью сравнить файлы. Emacs не можете ничего делать, пока `diff` не завершится. Возможно, в будущем кто-то изменит Emerge так, что она будет делать сравнение в фоновом режиме, когда входные файлы велики — тогда вы могли бы продолжать в Emacs другие дела, пока Emerge действительно не будет готова принимать команды.

После подготовки объединения Emerge запускает ловушку `emerge-startup-hook` (см. Раздел 31.2.3 [Ловушки], с. 349).

## 22.15 Режим C и родственные с ним

Этот раздел описывает особые средства, доступные в режимах C, C++, Objective-C, Java, CORBA IDL и Pike. Когда мы говорим “режим C и родственные с ним”, мы имеем в виду эти режимы.

### 22.15.1 Команды перемещения в режиме C

Этот раздел описывает команды для перемещения точки в режиме C и родственных с ним режимах.

`C-c C-u` Перемещает точку назад к содержащей ее условной констукции препроцессора, оставляя метку в текущей позиции. Префиксный аргумент работает в качестве счетчика повторов. С отрицательным аргументом, перемещает точку вперед к концу этой условной констукции препроцессора. При проходе назад, `#elif` рассматривается как `#else`, за которой стоит `#if`. При проходе вперед `#elif` игнорируется.

`C-c C-p` Перемещает точку назад через условную констукцию препроцессора, оставляя метку в текущей позиции. Префиксный аргумент служит в качестве счетчика повторов. С отрицательным аргументом, перемещает вперед.

`C-c C-n` Перемещает точку вперед через условную констукцию препроцессора, оставляя метку в текущей позиции. Префиксный аргумент служит в качестве счетчика повторов. С отрицательным аргументом, перемещает назад.

`M-a` Перемещает точку к началу самого внутреннего оперетора Си (`c-beginning-of-statement`). Если точка уже находится в начале оператора, перемещает к началу предыдущего. С префиксным аргументом $n$, перемещает назад на $n - 1$ операторов.

Если точка находится внутри строки или комментария или после комментария (между ними только пропуск), эта команда перемещает по предложениям, а не по операторам.

Если эта функция вызвана из программы, она принимает три необязательных аргумента: префиксный числовой аргумент, предел позиции в буфере (не перемещаться назад далее этой позиции) и флаг, который говорит, нужно ли двигаться по предложениям внутри комментария.

`M-e` Перемещает точку к концу самого внутреннего оперетора Си; как `M-a`, но перемещает в противоположном направлении (`c-end-of-statement`).

`M-x c-backward-into-nomenclature`
Перемещает точку назад к началу секции или слова в нотации Си++. С префиксным аргументом $n$, перемещает $n$ раз. Если $n$ отрицательно, перемещает вперед. Нотация Си++ обозначает запись имен символов в стиле ИменованиеСимволовВСмешанномРегистреИБезПодчерков; каждая заглавная буква начинает секцию или слово.

В проекте GNU мы рекомендуем использовать для разделения слов в идентификаторах Си или Си++ подчерки, а не изменение регистра.

`M-x c-forward-into-nomenclature`
Перемещает точку назад к концу секции или слова в нотации Си++. С префиксным аргументом $n$, перемещает $n$ раз.

### 22.15.2 Электрик-знаки в Си

В режиме Си и родственных с ним некоторые печатные знаки являются “электрическими” — помимо вставки самих себя, они также обновляют отступ в текущей строке и могут вставлять переводы строк. Это средство управляется переменной `c-auto-newline`. “Электрик”-знаки — это `{`, `}`, `:`, `#`, `;`, `,`, `<`, `>`, `/`, `*`, `(` и `)`.

Электрик-знаки вставляют переводы строк, только если включено средство *auto-newline* (это отображается в строке режима как ‘`/a`’ после имени режима). Это средство управляется переменной `c-auto-newline`. Вы можете включить или выключить его командой `C-c C-a`:

`C-c C-a` Перекючает автоматическую вставку переводов строк (`c-toggle-auto-state`). С префиксным аргументом, эта команда включает такую возможность, если аргумент положителен, и выключает, если аргумент отрицателен.

Двоеточие — это электрик-знак, поскольку это подходит для одного двоеточия. Но это неудобно, когда вы хотите вставить двойное двоеточие в Си++. Вы можете вставить двойное двоеточие в Си++ без вставки отступа или перевода строки, набирая `C-c :`.

`C-c :` Вставляет в точке оператор области видимости, двойное двоеточие, не изменяя отступ строки и не добавляя новых строк (`c-scope-operator`).

Электрик-ключ `#` обновляет отступ строки, если он оказался в начале директивы препроцессора. Это происходит, когда значение `c-electric-pound-behavior` равно

`(alignleft)`. Вы можете выключить эту возможность, устанавливая `c-electric-pound-behavior` в `nil`.

Переменная `c-hanging-braces-alist` управляет вставкой переводов строк до и после вставленных фигурных скобок. Это ассоциативный список с элементами в такой форме: (*синтаксический-символ . список-пс*). Большинство синтаксических символов, перечисленных в `c-offsets-alist`, имеют смысл и здесь.

Список *список-пс* может содержать один из символов `before` и `after`, либо оба, или это может быть `nil`. Когда вставляется фигурная скобка, в `c-hanging-braces-alist` ищется определяемый ей синтаксический контекст; если он найден, используется *список-пс* для выяснения того, где нужно вставить перевод строки: перед фигурной скобкой, после нее или и до, и после. Если ничего не найдено, по умолчанию вставляет перевод строки до и после фигурных скобок.

Переменная `c-hanging-colons-alist` управляет вставкой переводов строк до и после вставленных двоеточий. Это ассоциативный список, чьи элементы имеют форму (*синтаксический-символ . список-пс*). Список *список-пс* может содержать любые из символов `before` или `after`, либо оба, или это может быть `nil`.

Когда вставляется двоеточие, в этом списке ищется определяемый им синтаксический символ, и если он найден, используется *список-пс* для выяснения того, где нужно вставить перевод строки: перед двоеточия, после него или и там, и там. Если этот символ не найден в списке, переводы строк не вставляется.

Электрик-знаки могут также автоматически удалять переводы строк, когда включено средство для их автоматической вставки. Это делает автоматическую вставку переводов строк более приемлимой, путем удаления переводов строк в большинстве случаев, когда это было бы желательно; устанавливая переменную `c-cleanup-list`, вы можете указать *в каких* случаях это происходить. Значение этой переменной — это список символов, каждый из которых описывает один случай возможного удаления перевода строки. Вот перечень воспринимаемых символов и их значений:

`brace-catch-brace`
: Сжимает конструкцию ‘`}` `catch` (*условие*) `{`’, помещаяя ее целиком на одну строку. Сжатие происходит, когда вы набираете ‘`{`’, если между фигурными скобками нет ничего, кроме `catch` и *условия*.

`brace-else-brace`
: Сжимает конструкцию ‘`}` `else` `{`’, помещаяя ее целиком на одну строку. Сжатие происходит, когда вы набираете ‘`{`’ после `else`, но только если между фигурными скобками и `else` нет ничего, кроме пропусков.

`brace-elseif-brace`
: Сжимает конструкцию ‘`}` `else if (...)` `{`’, помещаяя ее целиком на одну строку. Сжатие происходит, когда вы набираете ‘`{`’, если между ‘`}`’ и ‘`{`’ нет ничего, кроме пропусков, не считая эти ключевые слова и условие для `if`.

`empty-defun-braces`
: Сжимает фигурные скобки пустого определения функции, помещая их на одну строку. Сжатие происходит, когда вы набираете закрывающую фигурную скобку.

`defun-close-semi`
: Сжимает двоеточие и `struct` или подобный тип объявления, помещая двоеточие на ту же строку, где стоит закрывающая фигурная скобка. Сжатие происходит, когда вы вводите двоеточие.

`list-close-comma`
: Сжимает запятые, следующие после фигурных скобок в массивах и сложных инициализациях. Сжатие происходит, когда вы набираете запятые.

`scope-operator`
: Сжимает двойное двоеточие, которое может обозначать оператор области видимости в Си++, помещая эти двоеточия вместе. Сжатие происходит, когда вы набираете второе двоеточие, но только если они разделены только пропуском.

### 22.15.3 Средство голодного удаления в Си

Когда включено средство *голодного удаления* (это показывается в строке режима как '`/h`' или '`/ah`' после имени режима), одна команда ⟨DEL⟩ удаляет весь предшествующий пропуск, а не только одни пробел. Чтобы включать и выключать эту возможность, используйте `C-c C-d`:

`C-c C-d` Включает или выключает средство голодного удаления (`c-toggle-hungry-state`). С префиксным аргументом, эта команда включает такую возможность, если аргумент положителен, и выключает, если аргумент отрицателен.

`C-c C-t` Переключает средства автоматической вставки перевода строки и голодного удаления одновременно (`c-toggle-auto-hungry-state`).

Переменная `c-hungry-delete-key` говорит, включено ли средство голодного удаления.

### 22.15.4 Другие команды режима C

`C-M-h` Помещает метку в конце определения функции, а точку в начале (`c-mark-function`).

`M-q` Заполняет абзац, обрабатывая комментарии Си и Си++ (`c-fill-paragraph`). Если какая-либо часть текущей строки является комментарием или находится внутри комментария, эта команда заполняет этот комментарий или его абзац, сохраняя отступы и ограничители комментария.

`C-c C-e` Запускает препроцессор Си для текста в области и показывает результат, который включает раскрытия всех вызовов макросов (`c-macro-expand`). Текст буфера, написанный перед областью, также передается препроцессору, так как там могут быть определения макросов, но вывод для этой части не показывается.

Когда вы отлаживаете использующий макросы код на Си, бывает трудно точно понять, как раскрываются макросы. С этой командой вам не нужно это понимать, вы можете видеть раскрытия.

`C-c C-\` Вставляет или выравнивает знаки '\' в концах строк области (`c-backslash-region`). Это полезно после написания или редактирования определения макроса Си.

Если строка уже завершается знаком '\', эта команда подстраивает размер пропуска перед ним. В противном случае она вставляет новый '\'. Однако, последняя строка области рассматривается особо; в нее не вставляется '\', а если если этот знак там стоит, то он удаляется.

`M-x cpp-highlight-buffer`
: Подсвечивает части текста в соответствии с условными конструкциями препроцессора. Эта команда показывает еще один буфер с именем '`*CPP Edit*`', который служит в качестве графического меню для выбора способа отображения конкретных видов условных конструкций и их содержимого. После изменения различных установок щелкните на '`[A]pply these settings`' (или перейдите в этот буфер и нажмите `a`), чтобы соответственно обновить подветку в буфере с режимом C.

`C-c C-s` Показывает информацию о синтаксисе текущей исходной строки (`c-show-syntactic-information`). Это та информация, которая управляет отступом строки.

### 22.15.5 Комментарии в режимах C

Режим C и родственные режимы используют несколько переменных для управления форматом комментариев.

`c-comment-only-line-offset`
: Дополнительный сдвиг для строки, которая содержит только начало комментария. Это может быть либо число, либо пара в форме (*не-привязанный-сдвиг* . *привязанный-сдвиг*), где *не-привязанный-сдвиг* — это размер сдвига, придаваемый полнострочным комментариям, начинающимся не в нулевом столбце, а *привязанный-сдвиг* — это размер сдвига, даваемый полнострочным комментариям, начинающимся в нулевом столбце. Простое число в качестве значения эквивалентно (*значение* . 0).

`c-comment-start-regexp`
: Эта локальная для буфера переменная указывает, как распознавать начало комментария.

`c-hanging-comment-ender-p`
: Если эта переменная равна `nil`, `c-fill-paragraph` оставляет завершающую строку для блока комментария на отдельной строке. Значение по умолчанию равно `t`, что помещает закрывающий ограничитель комментария ‘`*/`’ в конце последней строки текста комментария.

`c-hanging-comment-starter-p`
: Если эта переменная равна `nil`, `c-fill-paragraph` оставляет начинающий ограничитель блока комментария на отдельной строке. Значение по умолчанию равно `t`, что помещает открывающий ограничитель комментария ‘`/*`’ в начале первой строки текста комментария.

## 22.16 Режим Fortran

Режим Fortran предоставляет специальные команды движения для операторов и подпрограмм на Фортране и команды отступов, которые понимают фортрановские соглашения о вложенности, номера строк и операторы продолжения. Режим Fortran имеет свой собственный режим Auto Fill, который обрывает длинные строки на правильные с точки зрения Фортрана строки продолжения.

Предусматриваются специальные команды для комментариев, так как комментарии в Фортране не похожи на комментарии в других языках. Возможны встроенные сокращения, которые убыстряют набор, когда вы вставляете ключевые слова Фортрана.

Используйте `M-x fortran-mode`, чтобы переключиться в этот режим. Эта команда запускает ловушку `fortran-mode-hook` (см. Раздел 31.2.3 [Ловушки], с. 349).

### 22.16.1 Команды движения

Режим Fortran обеспечивает специальные команды для движения через подпрограммы (функции и процедуры) и через операторы. Есть также команда для установки области вокруг подпрограмм, удобная для их уничтожения и перемещения.

`C-M-a` Переместить точку на начало подпрограммы (`beginning-of-fortran-subprogram`).

`C-M-e` Переместить точку на конец подпрограммы (`end-of-fortran-subprogram`).

`C-M-h` Поставить точку в начале подпрограммы, а метку в ее конце (`mark-fortran-subprogram`).

`C-c C-n` Перейти на начало текущего или следующего оператора (`fortran-next-statement`).

`C-c C-p` Перейти на конец текущего или предыдущего оператора (`fortran-previous-statement`).

### 22.16.2 Отступы в Фортране

Для создания отступов в программах на Фортране необходимы специальные команды и средства, чтобы быть уверенным в том, что различные синтаксические единицы (номера строк, указатели комментариев и флаги продолжения строк) появляются в тех столбцах, которые требует стандарт Фортрана.

#### 22.16.2.1 Команды отступа в Фортране

⟨TAB⟩ Сделать отступ текущей строки (`fortran-indent-line`).

`C-j` Сделать отступ текущей строки и начать новую строку с отступом (`fortran-indent-new-line`).

`C-M-j` Прервать текущую строку и подготовить строку продолжения.

`M-^` Соединить эту строку со следующей.

`C-M-q` Сделать отступ во всех строках подпрограммы, в которой находится точка (`fortran-indent-subprogram`).

Режим Fortran переопределяет ⟨TAB⟩ так, чтобы он делал новый отступ в текущей строке для Фортрана (`fortran-indent-line`). Номера строк и маркеры продолжения имеют отступ до требуемых столбцов, а тело оператора получает независимый отступ, основанный на его вложенности в программе.

Ключ `C-j` запускает команду `fortran-indent-new-line`, которая заново делает отступ в текущей строке, а затем создает новую строку и деалет отступ в ней. Эта команда полезна для создания нового отступа в закрывающем операторе циклов ‘`do`’ и других блоков перед началом новой строки.

Ключ `C-M-q` запускает `fortran-indent-subprogram`, команду для создания отступа во всех строках фортрановской подпрограммы (функции и процедуры), содержащей точку.

Ключ `C-M-j` запускает `fortran-split-line`, которая разрывает строку соответствующим Фортрану способом. В строке, которая не является комментарием, вторая половина становится строкой продолжения и имеет соответственный отступ. В строке комментария обе половины становятся отдельными строками комментария.

`M-^` запускает команду `fortran-join-line`, которая более или менее является обращением `fortran-split-line`. Она объединяет текущую и предшествующую строки подходящим для Фортрана способом.

#### 22.16.2.2 Строки продолжения

Большиство современных компиляторов Фортрана разрешают два способа написания строк продолжения. Если первый непробельный знак на строке находится в столбце 5, то

эта строка считается продолжением предыдущей. Мы называем это *фиксированным форматом*; (В GNU Emacs мы всегда остчитываем столбцы от нуля.) Переменная `fortran-continuation-string` указывает, какой знак надо помещать в столбец 5. Строка, начинающаяся со знака табуляции, за которым стоит любая цифра, кроме '`0`', также является строкой продолжения. Этот стиль продолжения мы называем *табулированным форматом*.

Режим Fortran может делать строки продолжения в обоих стилях, но вы должны указать, какой вы предпочитаете. Этим выбором управляет значение преременной `indent-tabs-mode`: `nil` означает фиксированный формат, а отличное от `nil` — табулированный. Вы можете судить о действующим в данный момент формате по наличию или отсутствию в строке режима слова '`Tab`'.

Если текст на строке начинается с принятого в Фортране маркера продолжения '`$`' или с непробельного знака в столбце 5, режим Fortran считает эту строку строкой продолжения. Когда вы делаете в строке продолжения отступ с помощью ⟨TAB⟩, эта строка приводится к текущему стилю продолжения. Когда вы разбиваете фортрановский оператор с помощью `C-M-j`, на новой строке создается маркер продолжения в соотвествии с этим стилем.

Установка стиля продолжения затрагивает некоторые другие аспекты редактирования в режиме Fortran. При фиксированном формате, минимальный номер столбаца для тела оператора равен шести. Строки внутри фортрановских блоков, отступ в которых больше этого числа, всегда используют для пропусков только пробелы. При табулированном формате, минимальный номер столбца для тела оператора равен восьми, и пропуск перед столбцом 8 всегда состоит из одного знака табуляции.

Когда вы включаете режим Fortran для существующего файла, он старается вычислить подходящий стиль продолжения автоматически, исходя из содержимого этого файла. Выбор определяет первая строка, которая начинается с табуляции или шести пробелов. Переменная `fortran-analyze-depth` определяет, сколько строк нужно рассмотреть (от начала файла); если ни одна их этих строк не укажет стиль, то он определяется по переменной `fortran-tab-mode-default`. Если она равна `nil`, то используется фиксированный формат, отличное от `nil` значение велит использовать табулированный формат.

### 22.16.2.3 Номера строк

Если первым непробельным текстом на строке является число, режим Fortran предполагает, что это номер строки, и перемещает его к столбцам от 0 до 4. (В GNU Emacs столбцы всегда отсчитываются от нуля.)

Номера строк из четырех и менее цифр обычно имеют отступ на один пробел. Это управляется переменной `fortran-line-number-indent`, значение которой является максимальным отступом, который может иметь номер строки. Номера строк получают такой отступ, чтобы они корректно оканчивались в четвертом столбце, если при этом не требуется отступ больше максимального. По умолчанию значение переменной равно 1.

Простая вставка номера строки достаточна для того, чтобы отступ у него соответствовал этим правилам. Как только вставляется каждая цифра, отступ пересчитывается. Чтобы выключить это свойство, установите переменную `fortran-electric-line-number` в `nil`. Тогда вставка номеров строк будет похожа на вставку всего остального.

### 22.16.2.4 Синтаксические соглашения

Режим Fortran предполагает, что вы следуете определенным соглашениям, которые упрощают задачу понимания программ на Фортране в достаточной степени, чтобы делать в них правильный отступ:

- Два вложенных цикла '`do`' никогда не имеют общего оператора '`continue`'.
- Ключевые слова Фортрана, такие как '`if`', '`else`', '`then`', '`do`' и другие, написаны без внутренних пробелов и разрывов строк.

Компиляторы Фортрана обычно игнорируют все пробельные знаки вне строковых констант, но режим Fortran не распознает эти ключевые слова, если они разорваны. Конструкции вроде ‘`else if`’ или ‘`end do`’ допустимы, но второе слово должно быть на той же строке, что и первое, а не на строке продолжения.

Если вы не следуете этим соглашениям, команды отступа могут сделать отступ в некоторых строках неэстетично. Однако, правильная программа на Фортране будет сохранять свое значение при новых отступах, даже если эти соглашения не соблюдались.

#### 22.16.2.5 Переменные для управления отступами

Несколько дополнительных переменных управляют тем, как работает отступ в Фортране:

`fortran-do-indent`
: Дополнительный отступ в пределах каждого уровня оператора ‘`do`’ (по умолчанию 3).

`fortran-if-indent`
: Дополнительный отступ в пределах каждого уровня оператора ‘`if`’ (по умолчанию 3). Это же значение используется для дополнительного отступа каждого уровня оператора Фортрана90 ‘`where`’.

`fortran-structure-indent`
: Дополнительный отступ в пределах каждого уровня операторов ‘`structure`’, ‘`union`’ или ‘`map`’ (по умолчанию 3).

`fortran-continuation-indent`
: Дополнительный отступ для тел строк продолжения (по умолчанию 5).

`fortran-check-all-num-for-matching-do`
: Если это `nil`, команды отступа считают, что каждый оператор ‘`do`’ кончается на операторе ‘`continue`’. Поэтому при вычислении отступа для оператора, отличного от ‘`continue`’, они могут сократить время, не выполняя в этом месте проверку окончания оператора ‘`do`’. Если это не `nil`, то команды отступа для любого пронумерованного оператора должны проверять, не заканчивается ли там ‘`do`’. По умолчанию значение равно ‘`nil`’.

`fortran-blink-matching-if`
: Если это `t`, создание отступа для оператора ‘`endif`’ на мгновение перемещает курсор к парному оператору ‘`if`’, чтобы вы видели, где он находится. По умолчанию `nil`.

`fortran-minimum-statement-indent-fixed`
: Минимальный отступ для операторов Фортрана при использовании фиксированного формата для строк продолжения. Тела операторов никогда не получают отступ менее этого. По умолчанию это 6.

`fortran-minimum-statement-indent-tab`
: Минимальный отступ для операторов Фортрана при использовании табулированного формата строк продолжения. Тела операторов никогда не получают отступ менее этого. По умолчанию это 8.

### 22.16.3 Комментарии в Фортране

Обычные команды Emacs для комментариев предполагают, что комментарии могут следовать за строкой кода. В Фортране стандартный синтаксис комментариев требует отведения строки целиком только под комментарий. Поэтому режим Fortran заменяет стандартные команды комментариев в Emacs и определяет некоторые новые переменные.

Режим Fortran также может обрабатывать нестандартный синтаксис комментариев, когда комментарии начинаются с ‘!’ и могут следовать за другим текстом. Так как только некоторые компиляторы Фортрана признают такой синтаксис, режим Fortran не вставляет такие комментарии, если вы не потребовали этого заранее. Чтобы сделать это, установите переменной `comment-start` значение ‘`"!"`’ (см. Раздел 31.2 [Переменные], с. 343).

`M-;` Выровнять комментарий или вставить новый комментарий (`fortran-comment-indent`).

`C-x ;` Применяется только к нестандартным комментариям ‘`!`’.

`C-c ;` Превратить все строки области в комментарии или (с аргументом) превратить их обратно в реальный код (`fortran-comment-region`).

`M-;` в режиме Fortran переопределяется на `fortran-comment-indent`. Как и обычная команда `M-;`, она распознает любой вид существующих комментариев и соответственно выравнивает его текст; если существующего комментария нет, то комментарий вставляется и выравнивается. Но вставка и выравнивание комментариев в режиме Fortran не такие, как в других режимах.

Когда должен быть вставлен новый комментарий, то, если текущая строка пустая, вставляется полная строка комментария. В непустой строке вставляется нестандартный комментарий с ‘`!`’, если вы сказали, что хотите их использовать. В противном случае в новую строку перед текущей вставляется полная строка комментария.

Нестандартные комментарии с ‘`!`’ выравниваются, как комментарии в других языках, но полнострочные комментарии выравниваются иначе. В стандартном полнострочном комментарие сам ограничитель комментария должен всегда появляться в нулевом столбце. Что может выравниваться, так это текст в пределах комментария. Вы можете выбирать из трех возможных видов выравнивания, устанавливая переменную `fortran-comment-indent-style` в одно из этих значений:

`fixed` Текст выравнивается по фиксированному столбцу, который является суммой `fortran-commenrt-line-column` и минимального отступа оператора. Это значение принимается по умолчанию.

Минимальный отступ операторов — это `fortran-minimum-statement-indent-fixed` для стиля продолжения с фиксированным форматом и `fortran-minimum-statement-indent-tab` для стиля с табулированным форматом.

`relative` Текст выравнивается так, как если бы он был строкой кода, но с дополнительными `fortran-comment-line-column` столбцами отступа.

`nil` Текст в полнострочных комментариях не перемещается автоматически.

Кроме того, вы можете определить знак, который используется для отступа в пределах полнострочных комментариев, устанавливая переменной `fortran-comment-indent-char` значение, равное строке из одного знака, который вы хотите использовать.

В режиме Fortran вводятся две переменные, `comment-line-start` и `comment-line-start-skip`, которые играют для полнострочных комментариев ту же роль, что и `comment-start` и `comment-start-skip` для обычных, следующих за текстом комментариев. Обычно они устанавливаются правильно режимом Fortran, так что их не нужно менять.

Обычная команда Emacs для создания комментария `C-x ;` переопределена. Если вы используете комментарии с ‘`!`’, эта команда может быть использована с ними. Иначе она бесполезна в режиме Fortran.

Команда `C-c ;` (`fortran-comment-region`) превращает все строки области в комментарии, вставляя ‘`C$$$`’ в начале каждой из строк. С числовым аргументом, она превращает область обратно в реальный код, удаляя ‘`C$$$`’ из начала каждой строки в этой области. Строка, используемая для этих комментариев, может управляться установкой переменной

`fortran-comment-region`. Заметим, что здесь мы имеем пример команды и переменной с одним и тем же именем. Эти два варианта использования имени никогда не конфликтуют, так как в Лиспе и в Emacs всегда понятно по контексту, какое из них имеется в виду.

### 22.16.4 Режим Fortran Auto Fill

Режим Fortran Auto Fill — это второстепенный режим, который автоматически разбивает фортрановские операторы, когда они становятся слишком широкими по мере того, как вы их вставляете. Разбиение оператора влечет создание строки продолжения с использованием `fortran-continuation-string` (см. Раздел 22.16.2.2 [Строки продолжения в Фортране], с. 240). Разбиение происходит, когда вы набираете ⟨SPC⟩, ⟨RET⟩ или ⟨TAB⟩, а также в командах для отступов в Фортране.

`M-x fortran-auto-fill-mode` включает режим Fortran Auto Fill, если он был выключен, или выключает, если он был включен. Эта команда работает так же, как работает `M-x auto-fill-mode` для обычного режима Auto Fill (см. Раздел 21.5 [Заполнение], с. 185). Положительный аргумент включает режим Fortran Auto Fill, а отрицательный выключает. Вы можете узнать, действует ли режим Fortran Auto Fill, по наличию слова ‘`Fill`’ в строке режима в круглых скобках. Режим Fortran Auto Fill — это второстепенный режим, включаемый и выключаемый в каждом буфере отдельно. См. Раздел 31.1 [Второстепенные режимы], с. 341.

Режим Fortran Auto Fill разрывает строки на пробелах или разделителях, когда строки становятся длиннее желаемой ширины (значения `fill-column`). Разделителями, на которых режим Fortran Auto Fill может разорвать строку, являются ‘`,`’, ‘`'`’, ‘`+`’, ‘`-`’, ‘`/`’, ‘`*`’, ‘`=`’ и ‘`)`’. Разрыв происходит после разделителя, если переменная `fortran-break-before-delimiters` равна `nil`. Иначе (и по умолчанию) разрыв делается перед разделителем.

По умолчанию режим Fortran Auto Fill не задействован. Если вы хотите, чтобы это средство было включено постоянно, добавьте к `fortran-mode-hook` функцию-ловушку, которая выполнит `(fortran-auto-fill-mode 1)`. См. Раздел 31.2.3 [Ловушки], с. 349.

### 22.16.5 Проверка столбцов в Фортране

`C-c C-r` Кратковременно показать “линейку столбцов” над текущей строкой (`fortran-column-ruler`).

`C-c C-w` Временно разделить текущее окно по горизонтали таким образом, чтобы оно стало шириной в 72 столбца. Это может помочь вам избежать выхода за лимит в 72 столбца, который накладывают некоторые компиляторы Фортрана (`fortran-window-create-momentarily`).

Команда `C-c C-r` (`fortran-column-ruler`) кратковременно показывает над текущей строкой линейку столбцов. Линейка столбцов — это две строки текста, которые показывают вам позиции столбцов, имеющих специальные значения в Фортран-программах. Квадратные скобки показывают границы столбцов для номеров строк, а фигурные скобки показывают границы столбцов для тела оператора. Над ними показаны номера столбцов.

Заметьте, что номера столбцов считаются от нуля, как всегда в GNU Emacs. В связи с этим номера могут на единицу меньше, чем те, к которым вы привыкли; но указываемые ими позиции в строке стандартны для Фортрана.

Текст, используемый для показа линейки столбцов, зависит от значения переменной `indent-tabs-mode`. Если `indent-tabs-mode` равна `nil`, то в качестве линейки столбцов используется значение переменной `fortran-column-ruler-fixed`. Иначе показывается переменная `fortran-column-ruler-tab`. Изменяя эти переменные, вы можете изменить вид линейки столбцов.

Для еще большей помощи используйте команду `C-c C-w` (`fortran-window-create`), которая разделяет теущее окно по горизонтали, делая его ширину равной 72 столбцам. При редактировании в этом окне вы можете непосредственно видеть, когда вы сделали строку слишком длинной, чтобы она была правильной с точки зрения Фортрана.

### 22.16.6 Сокращения ключевых слов Фортрана

Режим Fortran обеспечивает множество встроенных сокращений для часто встречающихся ключевых слов и объявлений. Это те же виды сокращений, которые вы можете определить сами. Чтобы использовать их, вы должны включить режим Abbrev. См. Глава 24 [Сокращения], с. 257.

Встроенные сокращения необычны в одном: все они начинаются с точки с запятой. Обычно вы не можете использовать точку с запятой в сокращениях, но режим Fortran делает это возможным, изменяя синтаксис точки с запятой на “составную часть слова”.

Например, одно встроенное фортрановское сокращение — это ‘`;c`’ для ‘`continue`’. Если вы вставите ‘`;c`’ и затем поставите знаки пунктуации, например пробел или перевод строки, то ‘`;c`’ автоматически изменится на ‘`continue`’, при условии, что включен режим Abbrev.

Наберите ‘`;?`’ или ‘`;C-h`’, чтобы просмотреть все встроенные сокращения для Фортрана и то, чему они соответствуют.

### 22.16.7 Другие команды режима Fortran

`C-x n d` Сужает до текущей подпрограммы Фортрана.

Режим Fortran переопределяет ключ `C-x n d` для запуска команды `fortran-narrow-to-subprogram`, которая служит фортрановским аналогом обычного определения этого ключа. Она сужает буфер до подпрограммы, содержащей точку.

## 22.17 Режим Asm

Режим Asm — это основной режим для редактирования файлов на ассемблерном коде. Он определяет следующие команды:

⟨TAB⟩ `tab-to-tab-stop`.

`C-j` Вставляет перевод строки и делает отступ, используя `tab-to-tab-stop`.

`:` Вставляет двоеточие и затем удаляет отступ перед меткой, предшествующей двоеточию. Затем делает `tab-to-tab-stop`.

`;` Вставляет или выравнивает комментарий.

Переменная `asm-comment-char` определяет, какой знак начинает комментарий в синтаксисе ассемблера.

# 23 Сборка и тестирование программ

В предыдущей главе обсуждались команды Emacs, полезные для внесения изменений в программы. Эта глава имеет дело с командами, которые помогают в обширном процессе разработки и сопровождения программ.

## 23.1 Запуск компиляторов в Emacs

Emacs может запускать компиляторы для недиалоговых языков, таких как Си и Фортран, как подчиненные процессы, подавая протокол ошибок в буфер Emacs. Он также может произвести разбор сообщений об ошибках и показать вам строки исходных текстов, где произошла ошибка.

`M-x compile`
: Асинхронно запускает компилятор под управлением Emacs, выводя сообщения об ошибках в буфер '`*compilation*`'.

`M-x grep`
: Асинхронно запускает `grep` под управлением Emacs, перечисляя совпавшие строки в буфере '`*grep*`'.

`M-x grep-find`
: Запускает `grep` через `find` с предоставленными пользователем аргументами, направляя вывод в буфер '`*grep*`'.

`M-x kill-compilation`
`M-x kill-grep`
: Уничтожает работающие подпроцессы компиляции или `grep`.

Чтобы запустить `make` или другую команду компиляции, выполните `M-x compile`. Эта команда считывает командную строку оболочки, используя минибуфер, и затем выполняет эту командную строку в подчиненной оболочке, помещая вывод в буфер с именем '`*compilation*`'. В качестве рабочего каталога для выполнения этой команды используется каталог по умолчанию текущего буфера, следовательно, компиляция производится в этом каталоге.

Когда считывается командная строка оболочки, появляется минибуфер, содержащий командную строку по умолчанию; это команда, которую вы использовали при последнем применении `M-x compile`. Если вы наберете просто ⟨RET⟩, то снова будет использована та же самая командная строка. Для первой `M-x compile` по умолчанию используется '`make -k`'. Значение по умолчанию берется из переменной `compile-command`; если соответствующая команда компиляции для файла является чем-то другим, не '`make -k`', то может быть полезно иметь для этого файла локальное значение `compile-command` (см. Раздел 31.2.5 [Переменные файла], с. 351).

Запуск компиляции показывает буфер '`*compilation*`' в другом окне, но не выбирает его. Строка режима этого буфера сообщает вам, закончилась ли компиляция, при помощи слов '`run`' или '`exit`' в круглых скобках. Вы не обязаны держать этот буфер видимым, компиляция продолжается в любом случае. Пока компиляция продолжается, в строках режима всех буферов появляется слово '`Compiling`'. Если это слово исчезает, компиляция закончена.

Если вы хотите видеть протокол компиляции по мере его появления, переключитесь в буфер '`*compilation*`' и переместите точку в его конец. Когда точка расположена в конце, новый вывод процесса компиляции вставляется перед точкой, и она остается в конце. Если точка не находится в конце этого буфера, она остается на своем месте, тогда как дальнейший вывод компиляции добавляется в конец буфера.

Если вы установите переменную `compilation-scroll-output` в значение, отличное от `nil`, то буфер компиляции всегда прокручивается, чтобы показывать вывод по мере его появления.

Чтобы прекратить процесс компиляции, выполните команду `M-x kill-compilation`. Когда процесс компиляции будет прерван, строка режима буфера '`*compilation*`' изменится, и в ней будет слово '`signal`' вместо '`run`'. Запуск новой компиляции также уничтожает любую работающую компиляцию, так как в одно время может существовать только одна. Однако `M-x compile` требует подтверждения перед фактическим уничтожением уже запущенной компиляции.

## 23.2 Поиск с Grep под Emacs

Точно так же, как вы запускаете из Emacs компилятор, и затем обращаетесь к строкам, где были ошибки компиляции, вы можете запустить `grep` и затем обратиться к строкам, где были найдены совпадения. Это работает путем интерпретации сообщений о совпадениях от `grep` как сообщений об "ошибках".

Чтобы сделать это, наберите `M-x grep` и введите командную строку, указывающую, как нужно запускать `grep`. Используйте те же аргументы, которые вы дали бы `grep` при обычном запуске: регулярное выражение в формате `grep` (обычно в одиночных кавычках, чтобы отменить особый смысл специальных символов оболочки), за которым следуют имена файлов, в которых можно использовать шаблоны. Вывод из `grep` идет в буфер '`*compilation*`'. Вы можете обратиться к совпавшимстрокам при помощи `C-x ‘` и ⟨RET⟩, как к ошибкам компиляции.

Если вы зададите для `M-x grep` префиксный аргумент, она найдет в окрестности точки тег (см. Раздел 22.13 [Теги], с. 224) и поместит его в команду `grep` по умолчанию.

Команда `M-x grep-find` похожа на `M-x grep`, но предлагает другую командную строку по умолчанию — строку, которая запускает `find` и `grep`, так что поиск производится в каждом файле дерева каталогов. Смотрите также команду `find-grep-dired`, Раздел 28.15 [Поиск в Dired], с. 300.

## 23.3 Режим Compilation

В буфере '`*compilation*`' используется особый основной режим, режим Compilation, основная цель которого — предоставить удобный способ просмотреть строку исходного текста, где случилась ошибка.

`C-x ‘` Обратиться к позиции следующего сообщения об ошибке компиляции или совпадения, найденного `grep`.

⟨RET⟩ Обратиться к позиции сообщения об ошибке, в которой находится точка. Эта команда применяется в буфере компиляции.

`Mouse-2` Обратиться к позиции сообщения об ошибке, на котором вы щелкнули.

Вы можете обратиться к исходному тексту для любого конкретного сообщения об ошибке, переместив точку в буфере '`*compilation*`' к этому сообщению и нажав ⟨RET⟩ (`compile-goto-error`). Или щелкните на этом сообщении об ошибке `Mouse-2`; тогда вам не обязательно сначала переключаться в буфер '`*compilation*`'.

Чтобы последовательно сделать грамматический разбор сообщений компилятора об ошибках, набирайте `C-x ‘` (`next-error`). Знак, стоящий после `C-x` — это обратная кавычка или "акцент грав", а не обычная одиночная кавычка. Эта команда доступна во всех буферах, а не только в буфере '`*compilation*`'; она показывает следующее сообщение об ошибке вверху одного окна и текст, в котором находится эта ошибка, в другом окне.

Когда `C-x ‘` используется первый раз после начала компиляции, она передвигается к положению первой ошибки. Последующие использования `C-x ‘` продвигают вниз к следующим ошибкам. Если вы обратились к файлу по какому-то сообщению об ошибке с помощью ⟨RET⟩ или `Mouse-2`, последующие команды `C-x ‘` продвигаются с этого места. Когда `C-x ‘` доходит до конца буфера и не может найти больше сообщений, она завершается неуспехом и Emacs выдает ошибку.

`C-u C-x ‘` начинает просмотр буфера ‘`*compilation*`’ сначала. Это один из способов еще раз обработать один и тот же набор ошибок.

Режим Compilation также переопределяет ключи ⟨SPC⟩ и ⟨DEL⟩ для прокрутки по целому экрану, а `M-n` и `M-p` — для перемещения к следующему или предыдущему сообщению об ошибке. Вы также можете использовать `M-{` и `M-}` для перемещения вверх и вниз к сообщению об ошибке для другого исходного файла.

Возможности режима Compilation также доступны во второстепенном режиме, называемом Compilation Minor. Он позволяет вам разбирать сообщения об ошибках в любом буфере, а не только в обычном буфере для вывода протокола компиляции. Для включения этого второстепенного режима наберите `M-x compilation-minor-mode`. Это определит ключи ⟨RET⟩ и `Mouse-2` как в основном режиме Compilation.

Второстепенный режим Compilation работает в любом буфере, если содержимое этого буфера имеет понятный ему формат. В буфере Rlogin (см. Раздел 30.2.6 [Удаленная машина], с. 329), второстепенный режим Compilation автоматически получает удаленные исходные файлы по FTP (см. Раздел 14.1 [Имена файлов], с. 105).

## 23.4 Подоболочки для компиляции

Emacs использует для команды компиляции оболочку, но ей указывается, что она должна быть неинтерактивной. В частности, это означает, что оболочка начинается без подсказки. Если вы обнаружите, что буфер ‘`*compilation*`’ уродуют ваши обычные подсказки оболочки, то это значит, что вы сделали ошибку в вашем файле инициализации оболочки, установив подсказку, не учитывая условий, когда она не должна появляться. (Файл инициализации может называться ‘`.bashrc`’, ‘`.profile`’, ‘`.cshrc`’, ‘`.shrc`’ или еще как-нибудь в зависимости от используемой вами оболочки.) Файл инициализации оболочки должен устанавливать подсказку, только если подсказка уже есть. Покажем, как это нужно делать в csh:

```
if ($?prompt) set prompt = ...
```

А так это делается в bash:

```
if [ "${PS1+set}" = set ]
then PS1=...
fi
```

Могут быть и другие вещи, которые вы должны делать только в интерактивной оболочке. Для проверки условия интерактивного запуска вы можете использовать такой же метод.

“Операционная система” MS-DOS не поддерживает асинхронные подпроцессы; чтобы как-то обойти этот недостаток, `M-x compile` в MS-DOS запускает команду компиляции синхронно. Как следствие, вы должны дождаться завершения этой команды до того, как сможете сделать что-то в Emacs. См. Приложение C [MS-DOS], с. 403.

## 23.5 Запуск отладчиков в Emacs

Библиотека GUD (Grand Unified Debugger[1]) предоставляет интерфейс к различным символьным отладчикам из Emacs. Мы рекомендуем отладчик GDB, который распространя-

[1] Единый отладчик. *(Прим. переводчика)*

ется свободно, но вы также можете запускать DBX, SDB или XDB, если они у вас есть. GUD может также служить интерфейсом к отладочному режиму Perl, отладчику Python PDB и JDB, отладчику Java.

### 23.5.1 Запуск GUD

Существует несколько команд для запуска отладчика, каждая соответствует конкретной программе-отладчику.

`M-x gdb` ⟨RET⟩ *файл* ⟨RET⟩
: Запускает GDB как подпроцесс Emacs. Эта команда создает буфер для ввода и вывода GDB и переключает в него. Если буфер GDB уже существует, она просто переключает в этот буфер.

`M-x dbx` ⟨RET⟩ *файл* ⟨RET⟩
: Аналогично, но запускает DBX вместо GDB.

`M-x xdb` ⟨RET⟩ *файл* ⟨RET⟩
: Аналогично, но запускает XDB, а не GDB. Используйте переменную `gud-xdb-directories` для задания каталогов поиска исходных файлов.

`M-x sdb` ⟨RET⟩ *файл* ⟨RET⟩
: Аналогично, но запускает SDB, а не GDB.

  Некоторые версии SDB не называют в своих сообщениях имена исходных файлов. Когда вы используйте их, у вас должна быть создана правильная таблица тегов (см. Раздел 22.13 [Теги], с. 224), чтобы GUD мог найти функции в исходных файлах. Если вы не обращались к таблице тегов или таблица тегов не содержит одну из функций, вы получите сообщение, говорящее ‘`The sdb support requires a valid tags table to work`’.[2] Если это случилось, создайте в рабочем каталоге правильную таблицу тегов и попробуйте снова.

`M-x perldb` ⟨RET⟩ *файл* ⟨RET⟩
: Запускает интерпретатор Perl в отладочном режиме для отладки *файла*, программы на Perl.

`M-x jdb` ⟨RET⟩ *файл* ⟨RET⟩
: Запускает для отладки *файла* отладчик Java.

`M-x pdb` ⟨RET⟩ *файл* ⟨RET⟩
: Запускает для отладки *файла* отладчик Python.

Каждая из этих команд принимает один аргумент: командную строку для вызова отладчика. В простейшем случае, задайте просто имя исполняемого файла, который вы хотите отлаживать. Вы также можете использовать ключи, поддерживаемые вашим отладчиком. Однако шаблоны и переменные оболочки недопустимы. GUD предполагает, что первый аргумент, не начинающийся с ‘`-`’, является именем исполняемого файла.

Emacs может запустить только один отладочный процесс в одно время.

### 23.5.2 Управление отладчиком

Когда вы запустили отладчик с помощью GUD, он использует буфер Emacs для обычного ввода и вывода. Этот буфер называется буфером GUD. Отладчик показывает строки исходных файлов, обращаясь к ним в буферах Emacs. Стрелка (‘`=>`’) в одном из буферов указывает на исполняемую в данный момент строку. Перемещение точки в буфере не изменяет положения стрелки.

[2] Для работы поддержки sdb требуется правильная таблица тегов. *(Прим. переводчика)*

Вы можете в любое время начать редактировать исходные файлы в тех буферах, которые их показывают. Стрелка не является частью текста файла; она появляется лишь на экране. Если вы действительно изменяете исходный файл, помните, что удаление или вставка строк собьет положение стрелки; GUD не может определить, какая строка строка соответствовала номеру строки в сообщении отладчика до вашего изменения. Кроме того, чтобы ваши изменения нашли отражение в таблицах отладчика, вам обычно придется перекомпилировать и перезапустить программу.

Если вы захотите, вы можете полностью управлять процессом отладчика через его буфер, который использует вариант режима Shell. Доступны все обычные команды вашего отладчика, и вы можете использовать команды истории режима Shell для их повторения. См. Раздел 30.2.3 [Режим Shell], с. 325.

### 23.5.3 Команды GUD

Буфер диалога с GUD использует вариант режима Shell, так что вам доступны команды этого режима (см. Раздел 30.2.3 [Режим Shell], с. 325). Режим GUD также предоставляет команды для установки и сброса контрольных точек, для выбора фреймов стека и для пошагового прохода по программе. Эти команды доступны как в буфере GUD, так и глобально, но с разными привязками.

Команды управления контрольными точками обычно используются в буферах, обращающихся к файлам, потому что они дают способ указать где поставить или убрать контрольную точку. Вот глобальные привязки для установки контрольных точек:

`C-x` ⟨SPC⟩ Устанавливает контрольную точку в исходной строке, где находится точка.

Это другая специальная команда, предоставляемая GUD. Ключи, начинающиеся с `C-c`, доступны только в буфере диалога с GUD. Привязки ключей, начинающиеся с `C-x C-a`, доступны и в буфере диалога с GUD, и в исходных файлах.

`C-c C-l`
`C-x C-a C-l`
: Отображает в другом окне последнюю строку, на которую появилась ссылка в буфере GUD (то есть строку, указанную в последнем сообщении о позиции в программе). Это запускает команду `gud-refresh`.

`C-c C-s`
`C-x C-a C-s`
: Исполняет одну строку кода (`gud-step`). Если строка содержит вызов функции, выполнение останавливается после входа в нее.

`C-c C-n`
`C-x C-a C-n`
: Исполняет одну строку кода, проходя через вызовы функций без остановки (`gud-next`).

`C-c C-i`
`C-x C-a C-i`
: Исполняет одну машинную инструкцию (`gud-stepi`).

`C-c C-r`
`C-x C-a C-r`
: Продолжает исполнение с неопределенной точкой останова. Программа продолжит работу до тех пор, пока не попадет на контрольную точку, завершится или получит сигнал, проверяемый отладчиком (`gud-cont`).

`C-c C-d`
`C-x C-a C-d`
: Удаляет контрольную точку (одну или несколько) в текущей строке исходного текста, если они поставлены (`gud-remove`). Если вы вызовите эту команду в

буфере диалога с GUD, она применяется к строке, на который в последний раз остановилась программа.

`C-c C-t`
`C-x C-a C-t`
: Устанавливает временную контрольную точку на текущей исходной строке, если такая есть в данный момент. Если вы вызовите эту команду в буфере диалога с GUD, она применяется к строке, на которой программа остановилась в последний раз.

Перечисленные выше команды относятся ко всем поддерживаемым отладчикам. Если вы пользуетесь GDB или DBX (некоторыми версиями), доступны такие дополнительные команды:

`C-c <`
`C-x C-a <` Выбирает следующий внешний фрейм стека (`gud-up`). Это эквивалентно команде ‘`up`’.

`C-c >`
`C-x C-a >` Выбирает следующий внутренний фрейм стека (`gud-down`). Это эквивалентно команде ‘`down`’.

Если вы пользуетесь GDB, доступны следующие привязки:

⟨TAB⟩ Завершает имя символа (`gud-gdb-complete-command`). Этот ключ доступен только в буфере диалога с GUD и требует GDB версии 4.13 или более поздней.

`C-c C-f`
`C-x C-a C-f`
: Исполняет программу до тех пор, пока не произойдет возврат из выбранного фрейма стека (или пока программа не остановится по другой причине).

Эти команды интерпретируют числовой аргумент как счетчик повторений, если это имеет смысл.

Так как ⟨TAB⟩ служит командой завершения, вы не можете использовать ее для набора знака табуляции в качестве ввода для программы, которую вы отлаживаете в GDB. Вместо этого, чтобы ввести знак табуляции, набирайте `C-q` ⟨TAB⟩.

### 23.5.4 Настройка GUD

Во время запуска GUD выполняет одну из следующих ловушек: `gdb-mode-hook`, если вы пользуетесь GDB; `dbx-mode-hook`, если вы пользуетесь DBX; `sdb-mode-hook`, если вы пользуетесь SDB; `xdb-mode-hook`, если вы пользуетесь XDB; `perldb-mode-hook` для отладочного режима Perl; `jdb-mode-hook` для PDB; `jdb-mode-hook` для JDB. Вы можете использовать эти ловушки для определения ваших собственных привязок ключей для буфера диалога с отладчиком. См. Раздел 31.2.3 [Ловушки], с. 349.

Вот удобный способ определить команду, которая посылает отладчику некоторую командную строку и устанавливает для нее привязку в буфере диалога с отладчиком:

`(gud-def` *функция командная-строка привязка док-строка*`)`

Это определит команду с именем *функция*, которая посылает процессу отладчика *командную-строку*, и даст ей строку описания *док-строка*. Вы можете использовать определенную таким образом команду в любом буфере. Если *привязка* не равна `nil`, `gud-def` также привязывает эту команду к `C-c` *привязка* в режиме буфера GUD и к `C-x C-a` *привязка* глобально.

Командная строка может содержать определенные ‘`%`’-последовательности, которые обозначают данные, подставляемые во время вызова *функции*:

‘%f’ Имя текущего исходного файла. Если текущим является буфер GUD, то “текущий исходный файл” — это файл, в котором программа остановилась.

‘%l’ Номер текущей исходной строки. Если текущим является буфер GUD, то “текущая исходная строка” — это строка, на которой остановилась программа.

‘%e’ Текст lvalue языка Си или выражения вызова функции, в котором или рядом с которым находится точка.

‘%a’ Текст шестнадцатиричного адреса, в котором или рядом с которым находится точка.

‘%p’ Числовой аргумент вызванной функции в виде десятичного числа. Если эта команда используется без числового аргумента, ‘%p’ будет пустой строкой.

Если вы не использовали ‘%p’ в командной строке, определяемая вами команда будет игнорировать любой числовой аргумент.

## 23.6 Исполнение лисповских выражений

В Emacs есть несколько основных режимов для языков Лисп и Scheme. В них используются одни и те же команды редактирования, но разные команды для исполнения выражений Лиспа. У каждого режима свое предназначение.

Режим Emacs-Lisp
: Режим для редактирования исходных файлов программ, запускаемых в Emacs Lisp. Этот режим определяет `C-M-x` для вычисления текущего определения функции. См. Раздел 23.7 [Библиотеки Лиспа], с. 253.

Режим Lisp Interaction
: Режим для диалогового сеанса с Emacs Lisp. Он определяет `C-j` для вычисления s-выражения перед точкой и вставки его значения в буфер. См. Раздел 23.9 [Диалог с Лиспом], с. 255.

Режим Lisp
: Режим для редактирования исходных файлов программ, которые запускают в Лиспах, отличных от Emacs Lisp. Этот режим определяет `C-M-x` так, чтобы он посылал текущее определение функции подчиненному Лисп-процессу. См. Раздел 23.10 [Внешний Лисп], с. 256.

Режим Inferior Lisp
: Режим для диалогового сеанса с подчиненным процессом Лиспа. Этот режим объединяет специальные средства режима Lisp и режима Shell (см. Раздел 30.2.3 [Режим Shell], с. 325).

Режим Scheme
: Подобен режиму Lisp, но для программ на Scheme.

Режим Inferior Scheme
: Режим для диалогового сеанса с подчиненным процессом Scheme.

Большинство команд редактирования для работы с программами на Лиспе на самом деле доступны глобально. См. Глава 22 [Программы], с. 205.

## 23.7 Библиотеки Лисп-программ для Emacs

Лисп-программы для команд редактирования хранятся в файлах, чьи имена традиционно оканчиваются на ‘.el’. Это окончание сообщает Emacs, что редактировать их следует в режиме Emacs-Lisp (см. Раздел 23.6 [Исполнение Лиспа], с. 253).

Чтобы выполнить файл с кодом на Emacs Lisp, используйте `M-x load-file`. Эта команда считывает имя файла, используя минибуфер, и затем выполняет содержимое этого файла как Лисп-программу. Для этого нет необходимости сначала обращаться к файлу, в любом случае эта команда считывает файл таким, каким он находится на диске, а не как текст в буфере Emacs.

Если файл Лисп-программы установлен в каталогах с библиотеками Emacs Lisp, пользователи могут загрузить его, используя `M-x load-library`. Программы могут загрузить его, вызывая `load-library`, или с помощью `load`, более низкоуровневой функции, которая похожа на эту, но допускает некоторые дополнительные аргументы.

`M-x load-library` отличается от `M-x load-file` тем, что она просматривает последовательность каталогов и пробует три имени в каждом из них. Предположим, ваш аргумент — это *lib*; три этих имени — это '*lib*`.elc`', '*lib*`.el`' и наконец просто '*lib*'. Если существует файл '*lib*`.elc`', по соглашению это файл, получаемый в результате компиляции; лучше загрузить скомпилированный файл, так как он загружается и работает быстрее.

Если `load-library` обнаружит, что '*lib*`.el`' новее, чем '*lib*`.elc`', она напечатает сообщение, потому что это похоже на то, что кто-то внес изменения в '`.el`'-файл и забыл его перекомпилировать.

Так как аргумент для `load-library` обычно не является сам по себе действительным именем файла, то завершение имени файла недоступно. Действительно, используя эту команду, вы не знаете точно, какое имя файла будет использовано.

Последовательность каталогов, просматриваемых `M-x load-library`, определяется переменной `load-path`, списком строк, являющихся именами каталогов. По умолчанию значение этого списка содержит каталог, где хранится Лисп-код самого Emacs. Если у вас есть свои собственные библиотеки, поместите их в один каталог и добавьте этот каталог к `load-path`. `nil` в этом списке означает текущий каталог по умолчанию, но скорее всего, ставить в список `nil` — не самая лучшая идея. Если вы сами решили, что хотите иметь `nil` в этом списке, то наиболее вероятно, что в действительности вы хотите в этот раз использовать `M-x load-file`.

Часто вы не должны давать никакой команды, чтобы загрузить какую-нибудь библиотеку, так как определенные в ней команды настроены таким образом, что выполняют *автозагрузку* этой библиотеки. Попытка запуска любой из этих команд приводит к вызову `load` для загрузки нужной библиотеки. Это меняет определения автозагрузки на действительные определения из библиотеки.

Код на Emacs Lisp может быть откомпилирован в байт-код, который загружается быстрее, занимает после загрузки меньше памяти и быстрее выполняется. См. раздел "Byte Compilation" в *the Emacs Lisp Reference Manual*. По соглашению, скомпилированный код библиотеки помещается в отдельный файл, чье имя состоит из имени исходного файла библиотеки с добавленным '`c`'. Таким образом, скомпилированный код для '`foo.el`' попадает в '`foo.elc`'. Поэтому `load-library` сначала ищет '`.elc`'-файлы.

## 23.8 Вычисление выражений Emacs-Lisp

Программы на Лиспе, предназначенные для запуска в Emacs, нужно редактировать в режиме Emacs-Lisp. Это происходит автоматически для файлов, чьи имена кончаются на '`.el`'. В противоположность этому, сам режим Lisp используется для редактирования Лисп-программ, предназначенных для других Лисп-систем. Чтобы перейти в режим Emacs-Lisp, используете команду `M-x emacs-lisp-mode`.

Для проверки программ на Лиспе, предназначенных для запуска в Emacs, часто полезно вычислять какую-нибудь часть программы в том виде, как она находится в буфере Emacs. Например, после изменения текста определения лисповской функции, вычисление определения устанавливает это изменение для будущих вызовов функции. Вычисление лисповских

выражений также удобно при любом редактировании для запуска неинтерактивных функций (функций, которые не являются командами).

`M-:` Считывает одно лисповское выражение в минибуфере, вычисляет его и печатает его значение в эхо-области (`eval-expression`).

`C-x C-e` Вычисляет лисповское выражение, находящееся перед точкой, и печатает его значение в эхо-области (`eval-last-sexp`).

`C-M-x` Вычисляет определение функции, содержащее точку или находящееся после нее, и печатает его значение в эхо-области (`eval-defun`).

`M-x eval-region`
Вычисляет все лисповские выражения в области.

`M-x eval-current-buffer`
Вычисляет все лисповские выражения в этом буфере.

`M-:` (`eval-expression`) — это самая основная команда для интерактивного вычисления лисповских выражений. Она считывает выражение, используя минибуфер, так что вы можете выполнить любое выражение в любом буфере, независимо от того, что этот буфер содержит. Когда выражение вычислено, то текущим буфером опять является тот, который был текущим, когда вы набирали `M-:`.

В режиме Emacs-Lisp ключ `C-M-x` привязан к команде `eval-defun`, которая анализирует определение функции, следующее за точкой или содержащее ее, как выражение Лиспа и вычисляет его. Значение печатается в эхо-области. Эта команда удобна для установки в среде Лиспа изменений, которые вы только что сделали в тексте определения функции.

`C-M-x` обрабатывает выражения `defvar` особо. Обычно вычисление выражения `defvar` не делает ничего, если определяемая им переменная уже имеет значение. Но `C-M-x` всегда переустанавливает эту переменную в начальное значение, заданное выражением `defvar`. Эта особенность удобна для отладки программ на Лиспе.

Команда `C-x C-e` (`eval-last-sexp`) вычисляет лисповское выражением, находящееся в этом буфере перед точкой, и показывает его значение в эхо-области. Она доступна во всех основных режимах, а не только в режиме Emacs-Lisp. Она не обрабатывает `defvar` особо.

Если командам `C-M-x`, `C-x C-e` или `M-:` задан числовой аргумент, то они вставляют значение в текущий буфер в позиции точки, а не в печатают эхо-области. Значение аргумента не играет роли.

Самой общей командой для вычисления Лисп-выражений из буфера является `eval-region`. `M-x eval-region` анализирует текст из области как одно или несколько лисповских выражений, вычисляя их одно за другим. `M-x eval-current-buffer` похожа на предыдущую, но вычисляет весь буфер. Это оправданный способ установки Лисп-кода из файла, который вы как раз готовы проверить. После нахождения и исправления ошибки используйте `C-M-x` в каждой функции, которую вы изменяете. Это сохранит соответствие между миром Лиспа и исходным файлом.

## 23.9 Буферы диалога с Лиспом

Буфер '`*scratch*`', выбираемый в момент запуска Emacs, предназначен для интерактивного вычисления Лисп-выражений внутри Emacs.

Простейший способ использования буфера '`*scratch*`' — вставлять лисповские выражения, оканчивая каждое из них вводом `C-j`. Эта команда считывает Лисп-выражение перед точкой, вычисляет его и вставляет его значение в печатном представлении перед точкой. Результатом будет полный протокол вычисленных вами выражений и их значений.

Основной режим буфера '`*scratch*`' называется Lisp Interaction; он во всем эквивалентен режиму Emacs-Lisp, за исключением привязки для `C-j`.

Целесообразность этого свойства состоит в том, что у Emacs должен быть буфер в момент запуска, но этот буфер неудобен для редактирования файлов, так как новый буфер создается для каждого файла, к которому вы обращаетесь. Я думаю, что самое полезное, что может делать начальный буфер, — это вести протокол интерпретатора Лиспа. `M-x lisp-interaction-mode` переведет любой буфер в режим Lisp Interaction.

Альтернативный способ интерактивного вычисления выражений Emacs Lisp предоставляет режим Inferior Emacs-Lisp, который имеет интерфейс, похожий скорее на интерфейс режима Shell (см. Раздел 30.2.3 [Режим Shell], с. 325). Наберите `M-x ielm`, чтобы создать буфер '`*ielm*`', использующий этот режим.

## 23.10 Запуск внешнего Лиспа

Emacs имеет средства для запуска программ в других Лисп-системах. Вы можете запустить Лисп-процесс как подчиненный процесс Emacs и передавать ему выражения, которые нужно вычислить. Вы также можете передать подчиненному Лисп-процессу измененные определения функций непосредственно из буферов Emacs, в которых вы редактируете программы на Лиспе.

Чтобы запустить подчиненный Лисп-процесс, наберите `M-x run-lisp`. Это запускает программу с именем `lisp`, ту же программу, которую бы вы запустили, набрав `lisp` как команду оболочки, с выводом и вводом, идущими через буфер Emacs с именем '`*lisp*`'. Следует сказать, что любой "терминальный вывод" от Лиспа пойдет в этот буфер, передвигая вперед точку, и любой "терминальный ввод" для Лиспа приходит из текста в буфере. (Вы можете изменить имя исполняемого файла Лиспа, установив переменную `inferior-lisp-program`.)

Чтобы дать ввод в Лисп, отправьте сь в конец буфера и наберите нужный текст, завершив его вводом ⟨RET⟩. Буфер '`*lisp*`' находится в режиме Inferior Lisp, режиме, который объединяет специальные характеристики режима Lisp и большую часть свойств режима Shell (см. Раздел 30.2.3 [Режим Shell], с. 325). Определение ⟨RET⟩ как команды, посылающей строку подпроцессу, — это одна из особенностей режима Shell.

Для запуска исходных файлов программ во внешних Лиспах используйте режим Lisp. Этот режим можно выбирать при помощи `M-x lisp-mode`; он используется автоматически для файлов, имена которых оканчиваются на '`.l`', '`.lsp`' или '`.lisp`', какие и применяются в большинстве Лисп-систем.

Когда вы редактируете функцию в программе на Лиспе, которая работает в данный момент, простейшим способом пересылки измененного определения в подчиненный Лисп-процесс является ключ `C-M-x`. В режиме Lisp при этом запускается функция `lisp-send-defun`, которая находит определение функции рядом или следом за точкой и посылает его как ввод в Лисп-процесс. (Emacs может послать ввод в любой подчиненный процесс независимо от того, какой буфер является текущим.)

Сравним значения `C-M-x` в режиме Lisp (для редактирования программ, запускаемых в другой Лисп-системе) и режиме Emacs-Lisp (для редактирования программ на Лиспе, предназначенных для работы в Emacs): в обоих режимах она имеет действие установки определения, в котором находится точка, но способ выполнения этого различается, согласно тому, где находится соответствующая среда Лиспа. См. Раздел 23.6 [Исполнение Лиспа], с. 253.

# 24 Сокращения

*Сокращение* — это слово, которое при вставке *расшифровывается* в какой-нибудь другой текст. Расшифровки сокращений определяются пользователем. Например, вы можете определить ‘`foo`’ как сокращение, расшифровывающееся в ‘`find outer otter`’. Определив такое сокращение, вы получите возможность вставлять в буфер ‘`find outer otter`’, набирая `f o o` ⟨SPC⟩.

Второй вид работы с сокращениями называется *динамической расшифровкой сокращений*. Вы используете динамическую расшифровку сокращений с помощью явной команды, раскрывающей буквы перед точкой путем поиска в этом буфере других слов, которые начинаются с таких же букв. См. Раздел 24.6 [Динамические сокращения], с. 260.

## 24.1 Понятия о сокращениях

*Сокращение* — это слово, которое было определено как *расшифровывающееся* в заданную *расшифровку*. Когда вы вставляете после сокращения знак, разделяющий слова, сокращение раскрывается, заменяясь на расшифровку. Например, если ‘`foo`’ определено как сокращение, расшифровывающееся в ‘`find outer otter`’, то вы можете вставить в буфер ‘`find outer otter.`’, напечатав `f o o .`.

Сокращения расшифровываются, только когда включен режим Abbrev (второстепенный режим). Выключение режима Abbrev не вызывает забвения определений сокращений, но они не расшифровываются, пока режим Abbrev не будет снова включен. Команда `M-x abbrev-mode` переключает режим Abbrev. С числовым аргументом она включает этот режим, если аргумент положительный, в противном случае — выключает его. См. Раздел 31.1 [Второстепенные режимы], с. 341. Символ `abbrev-mode` также является переменной; режим Abbrev включается, когда это переменная отлична от `nil`. Переменная `abbrev-mode` автоматически становится локальной для текущего буфера, как только вы ее устанавливаете.

Определения сокращений могут быть *зависящими от режима*, то есть действовать только в одном основном режиме. Сокращения могут также иметь *глобальные* определения, которые являются активными во всех основных режимах. Одни и те же сокращения могут иметь глобальное определение и различные зависящие от режима определения для разных основных режимов. Зависящее от режима определение для текущего основного режима перекрывает глобальное определение.

Сокращения можно определять интерактивно во время сеанса редактирования. Списки определений сокращений могут также храниться в файлах и перезагружаться в последующих сеансах. Некоторые пользователи держат обширные списки сокращений и загружают их в каждом сеансе.

## 24.2 Определение сокращений

`C-x a g` Определить сокращение, используя одно или несколько слов перед точкой в качестве его расшифровки (`add-global-abbrev`).

`C-x a l` Аналогично, но определить сокращение, доступное только в текущем основном режиме (`add-mode-abbrev`).

`C-x a i g` Определить слово в буфере как сокращение (`inverse-add-global-abbrev`).

`C-x a i l` Определить слово в буфере как свойственное для режима сокращение (`inverse-add-mode-abbrev`).

`M-x kill-all-abbrevs`
Эта команда сбрасывает все действующие в данный момент определения сокращений, начиная всё сначала.

Обычный способ определить сокращение — войти в текст, который вы хотите зашифровывать, установить после него точку и набрать `C-x a g` (`add-global-abbrev`). При этом само сокращение считывается в минибуфере, и затем оно определяется как сокращение для одного или более слов перед точкой. Используйте числовой аргумент для указания числа слов перед точкой, которые должны браться в качестве расшифровки. Например, чтобы определить сокращение '`foo`' как описано выше, вставьте текст '`find outer otter`' и затем наберите `C-u 3 C-x a g f o o` ⟨RET⟩.

Нулевой аргумент для `C-x a g` означает, что в качестве расшифровки определяемого сокращения следует использовать содержимое области.

Команда `C-x a l` (`add-mode-abbrev`) похожа, но определяет сокращение, свойственное для режима. Свойственные для режима сокращения активны только в отдельном основном режиме. `C-x a l` определяет сокращение для основного режима, действующего в момент набора этой команды. Аргументы работают так же, как и для `C-x a g`.

Если находящийся в буфере текст является сокращением, а не его расшифровкой, используйте команду `C-x a i g` (`inverse-add-global-abbrev`) вместо `C-x a g` или `C-x a i l` (`inverse-add-mode-abbrev`) вместо `C-x a l`. Эти команды называются "инверсными", так как они обращают смысл двух используемых текстовых строк (одной из буфера и второй, считываемой в минибуфере).

Чтобы изменить определение сокращения, просто добавьте новое определение. Вас спросят о подтверждении, если такое сокращение уже определено.

Чтобы уничтожить определение сокращения, дайте команде определения сокращений отрицательный аргумент: `C-u - C-x a g` или `C-u - C-x a l`. Первая удаляет глобальное определение, а вторая — свойственное для режима.

`M-x kill-all-abbrevs` уничтожает все имеющиеся определения сокращений, как глобальные, так и локальные.

## 24.3 Управление расшифровкой сокращения

Сокращение расшифровывается всякий раз, когда оно присутствует в буфере непосредственно перед точкой, и вы набираете самовставляющийся пробельный знак или знак пунктуации (⟨SPC⟩, запятую и тому подобное). Более точно, любой знак, не являющийся частью слова, раскрывает сокращение, а любой знак, который может быть частью слова, может быть частью сокращения. Наиболее общий способ использования сокращения — вставить его и ввести затем знак препинания.

Расшифровка сокращения сохраняет регистр букв; таким образом, '`foo`' расшифровывается в '`find outer otter`'; '`Foo`' в '`Find outer otter`', а '`FOO`' в '`FIND OUTER OTTER`' или в '`Find Outer Otter`' в зависимости от значения переменной `abbrev-all-caps` (значение, не равное `nil`, выбирает первую из двух расшифровок).

Для управления расшифровкой сокращений используются следующие команды:

`M-'` Отделить приставку от последующего сокращения, которое будет расшифровываться позднее (`abbrev-prefix-mark`).

`C-x a e` Расшифровать сокращение перед точкой (`expand-abbrev`). Это действует, даже когда отключен режим Abbrev.

`M-x expand-region-abbrevs`
Расшифровать некоторые или все сокращения в области.

Вы можете захотеть расшифровать сокращение с присоединенной приставкой; например, если '`cnst`' раскрывается в '`construction`', вы могли бы захотеть использовать его для ввода '`reconstruction`'. Набор '`recnst`' не работает, так как это не обязательно определенное сокращение. В этом случае сработает применение команды `M-'` (`abbrev-prefix-mark`) между приставкой '`re`' и сокращением '`cnst`'. Первым вставьте '`re`'. Затем

наберите `M-'`; эта команда вставит в буфер дефис, чтобы показать, что она выполнила свою работу. Затем вставьте сокращение '`cnst`'. Теперь буфер содержит '`re-cnst`'. Теперь вставьте знак, не являющийся частью слова, чтобы расшифровать сокращение '`cnst`' в '`construction`'. На этом шаге раскрытия дефис, оставшийся после использования `M-'`, удаляется. Результирующим текстом будет желаемое слово '`reconstruction`'.

Если вы на самом деле хотите получить в буфере текст сокращения, а не его расшифровку, вы можете достичь желаемого, вставив последующую пунктуацию с помощью `C-q`. Таким образом, `foo C-q ,` оставляет в буфере '`foo,`'.

Если вы расшифровали сокращение по ошибке, то вы можете отменить расшифровку и вернуть само сокращение, напечатав для отмены `C-_` (см. Раздел 4.4 [Отмена], с. 37). Это также отменяет вставку знака, не являющегося частью слова, который развернул сокращение. Если вы хотите получить завершающий знак, не входящий в слово, плюс нераскрытое сокращение, то вы должны заново вставить завершающий знак, экранировав его командой `C-q`. Вы также можете использовать команду `M-x unexpand-abbrev` для отмены последнего раскрытия без удаления завершающего знака.

`M-x expand-region-abbrevs` ищет в области определенные сокращения и предлагает заменить каждое найденное сокращение на его расшифровку. Это команда удобна, если вы набрали текст, используя сокращения, но забыли перед этим включить режим Abbrev. Она также может быть полезной вместе со специальным набором определений сокращений для выполнения нескольких глобальных замен за один раз. Эта команда действует, даже если выключен режим Abbrev.

Расшифровка сокращения запускает ловушку `pre-abbrev-expand-hook` (см. Раздел 31.2.3 [Ловушки], с. 349).

## 24.4 Проверка и редактирование сокращений

`M-x list-abbrevs`
: Показать перечень всех определений сокращений.

`M-x edit-abbrevs`
: Редактировать перечень сокращений; вы можете добавить, изменить или удалить определения.

Вывод `M-x list-abbrevs` выглядит так:

```
(lisp-mode-abbrev-table)
"dk"        0    "define-key"
(global-abbrev-table)
"dfn"        0    "definition"
```

(Некоторые пустые строки, не имеющие смысловой значимости, и некоторые другие таблицы сокращений были опущены.)

Строка, содержащая имя в круглых скобках, — это заголовок для сокращений из конкретной таблицы; `global-abbrev-table` содержит все глобальные сокращения, а другие таблицы сокращений, которые именуются в соответствии с основными режимами, содержат сокращения, специфичные для режима.

Каждая непустая строка в таблице определяет одно сокращение. Слово в начале строки — это само сокращение. Число, стоящее далее, говорит, сколько раз сокращение было расшифровано. Emacs отслеживает это, чтобы помочь вам увидеть, какие сокращения вы действительно используете, на случай, если вы решите уничтожить те, что не используются достаточно часто. Цепочка знаков в конце строки — это и есть расшифровка.

`M-x edit-abbrevs` позволяет вам добавить, изменить или уничтожить определения сокращений при помощи редактирования их списка в буфере Emacs. Этот список имеет тот же самый формат, что и описанный выше. Буфер сокращений называется '`*Abbrevs*`'

и находится в режиме Edit-Abbrevs. Напечатайте в этом буфере `C-c C-c`, чтобы установить, как указано в нем, определения сокращений и удалить все не перечисленные в нем определения.

Команда `edit-abbrevs` — это фактически то же самое, что и `list-abbrevs`, за исключением того, что она выбирает буфер '`*Abbrevs*`', тогда как `list-abbrevs` просто показывает его в другом окне.

## 24.5 Сохранение сокращений

Эти команды позволяют вам сохранять определения сокращений между сеансами редактирования.

`M-x write-abbrev-file` ⟨RET⟩ *файл* ⟨RET⟩
: Записать файл *файл*, описывающий все определенные сокращения.

`M-x read-abbrev-file` ⟨RET⟩ *файл* ⟨RET⟩
: Считать файл *файл* и определить сокращения так, как там описано.

`M-x quietly-read-abbrev-file` ⟨RET⟩ *файл* ⟨RET⟩
: То же самое, но не показывать сообщения о происходящем.

`M-x define-abbrevs`
: Определить сокращения из определений в текущем буфере.

`M-x insert-abbrevs`
: Вставить все сокращения и их расшифровки в текущий буфер.

`M-x write-abbrev-file` считывает имя файла, используя минибуфер, и записывает в этот файл описание всех текущих определений сокращений. Это используется для того, чтобы сохранить определения для использования в дальнейших сеансах. Хранимый в таком файле текст — это последовательность лисповских выражений, которые при выполнении определяют такие же сокращения, какие у вас есть в данный момент.

`M-x read-abbrev-file` запрашивает имя файла, используя минибуфер, и затем считывает этот файл, определяя сокращения согласно его содержимому. `M-x quietly-read-abbrev-file` — такая же команда, за исключением того, что она не показывает в эхо-области сообщение о своей работе; в действительности это удобно главным образом в файле '`.emacs`'. Если любой из этих функций передается пустой аргумент, то в качестве имени файла используется значение переменной `abbrev-file-name`, которая по умолчанию равна `"~/.abbrev_defs"`.

Emacs автоматически предложит записать сокращения, если вы изменили какое-либо из них, всякий раз, когда он предлагает сохранить все файлы (для `C-x s` или `C-x C-c`). Это свойство может быть отключено установкой переменной `save-abbevs` в значение `nil`.

Команды `M-x insert-abbrevs` и `M-x define-abbrevs` похожи на предыдущие команды, но работают над текстом в буфере Emacs. `M-x insert-abbrevs` вставляет текст в текущий буфер перед точкой, описывая все текущие определения сокращений; `M-x define-abbrevs` полностью анализирует текущий буфер и соответственно определяет сокращения.

## 24.6 Динамическая расшифровка сокращений

Описанные выше средства для работы с сокращениями действует автоматически по мере вставки текста, но все сокращения должны быть определены явно. В противоположность этому, *динамические сокращения* позволяют определять значения сокращений автоматически из содержимого буфера, но динамическая расшифровка происходит, только если вы запросите ее явно.

`M-/` Расшифровать слово в буфере перед точкой как *динамическое сокращение* при помощи поиска в буфере слов, начинающихся с этого сокращения (`dabbrev-expand`).

`C-M-/` Завершить слово перед точкой как динамическое сокращение (`dabbrev-completion`).

Например, если буфер содержит ‘`does this follow`’ и вы наберете `f o M-/`, то результатом будет вставка ‘`follow`’, потому что это последнее слово в буфере, которое начинается с ‘`fo`’. Числовой аргумент для `M-/` говорит, что следует брать вторую, третью и так далее отличающуюся расшифровку, найденную при просмотре в обратном направлении от точки. Повтор `M-/` ищет альтернативную расшифровку путем дальнейшего просмотра назад. После того, как будет просмотрен весь текст перед точкой, просматривается текст после точки. Переменная `dabbrev-limit`, если не равна `nil`, указывает, как далеко по буферу нужно искать расшифровку.

После просмотра текущего буфера `M-/` обычно просматривает другие буферы, если вы не установили `dabbrev-check-all-buffers` в значение `nil`.

Отрицательный аргумент для `M-/`, как `C-u - M-/`, говорит, что нужно искать расшифровки сначала после точки, а потом перед ней. Если вы повторяете `M-/` для поиска другого раскрытия, не задавайте аргумент. При этом сначала будут попробованы все расшифровки после точки, а затем все расшифровки перед точкой.

После того, как вы расшифровали динамическое сокращение, вы можете скопировать дополнительные слова, которые идут после расшифровки в оригинальном контексте. Просто печатайте ⟨SPC⟩ `M-/` для каждого слова, которое вы хотите скопировать. Промежутки и пунктуация между словами копируется вместе с ними.

Команда `C-M-/` (`dabbrev-completion`) производит завершение динамического сокращения. Вместо того, чтобы пробовать возможные расшифровки одну за другой, она находит их все, а потом вставляет текст, который является в них общим. Если в них нет ничего общего, `C-M-/` показывает перечень завершений, из которого вы можете выбрать нужное обычным способом. См. Раздел 5.3 [Завершение], с. 47.

Динамическая расшифровка сокращений совершенно не зависит от режима Abbrev; расшифровка слова с помощью `M-/` полностью независима от того, имеет ли оно определение как обыкновенное сокращение.

## 24.7 Настройки для динамических сокращений

Обычно динамическая расшифровка сокращений игнорирует регистр при поиске раскрытий. Это значит, что расшифровка не обязана совпадать по регистру с раскрываемым словом.

Это средство управляется переменной `dabbrev-case-fold-search`. Если она равна `t`, регистр при поиске игнорируется; если она равна `nil`, то слово и расшифровка должны иметь один регистр. Если значение `dabbrev-case-fold-search` равно `case-fold-search`, что верно по умолчанию, то игнорирование регистра во время поиска расшифровок определяется переменной `case-fold-search`.

Обычно динамическая расшифровка сокращений сохраняет образец регистра *сокращения, которое вы напечатали*, преобразуя расшифровку к регистру этого образца.

Переменная `dabbrev-case-replace` указывает, нужно ли сохранять образец регистра сокращения. Если она равна `t`, образец регистра сокращения сохраняется в большинстве случаев; если она `nil`, то расшифровка копируется буквально. Если значение `dabbrev-case-replace` равно `case-replace`, что истинно по умолчанию, то переменная `case-replace` указывает, нужно ли копировать расшифровку буквально.

Однако, если расшифровка содержит сложный набор букв разных регистров, и сокращение совпадает с этим образцом, когда доходит до него, то расшифровка копируется буквально, несмотря на значения этих переменных. Таким образом, если буфер содержит `переменнуюСГлупымОбразцомРегистров`, и вы напечатаете `п е M-/`, она скопирует расшифровку буквально, включая ее образец регистров.

Переменная `dabbrev-abbrev-char-regexp`, если не равна `nil`, указывает, какие знаки считаются частью слова для целей динамической расшифровки. Это регулярное выражение должно совпадать только с одним знаком, но никогда не с двумя или большим числом. Это же регулярное выражение определяет также, какие знаки являются частью расшифровки. Значение `nil` имеет особый смысл: сокращения состоят из знаков, являющихся частью слов, но расшифровки состоят из знаков, являющихся частью слов и символов.

В сценариях командного интерпретатора и Make-файлах к именам переменных иногда приставляется ‘`$`’, а иногда нет. Основные режимы для такого рода текста могут настроить динамическую расшифровку, чтобы она обрабатывала необязательный префикс, установив переменную `dabbrev-abbrev-skip-leading-regexp`. Ее значение должно быть регулярным выражением, совпадающим с необязательным префиксом, который должен игнорироваться динамическими сокращениями.

# 25 Редактирование рисунков

Чтобы создать рисунок, составленный из текстовых знаков (например изображение деления регистра на поля в качестве комментария в программе), используйте команду `M-x edit-picture` для входа в режим Picture.

В режиме Picture редактирование основывается на *квадрантной* модели текста, согласно которой текстовые знаки рассыпаны в области, простирающейся неограничено вправо и вниз. Понятие конца строки не существует в этой модели; самое большое, что вы можете сказать, — это где находится последний непустой знак на строке.

Конечно, в действительности Emacs всегда рассматривает текст как последовательность знаков, и строки на самом деле имеют концы. Но в режиме Picture наиболее часто используемые команды заменяются вариантами, которые воспроизводят квадрантную модель текста. Они делают это, вставляя пробелы или превращая в пробелы знаки табуляции.

Большинство базовых команд редактирования Emacs переопределяются режимом Picture таким образом, чтобы делать в основном те же самые вещи, но квадрантным способом. Кроме того, режим Picture определяет разные ключи, начинающиеся с префикса `C-c`, для запуска специальных команд редактирования изображения.

Один из этих ключей, `C-c C-c`, достаточно важен. Часто рисунок — это часть большего файла, который обычно редактируется в каком-нибудь другом основном режиме. Команда `M-x edit-picture` записывает имя предыдущего основного режима, и затем вы можете использовать команду `C-c C-c` (`picture-mode-exit`), чтобы вернуться в этот режим. Если `C-c C-c` не имеет числового аргумента, она также уничтожает пробелы в концах строк.

Все команды, используемые в режиме Picture, работают и в других режимах (если только загружена библиотека ‘`picture`’), но привязаны к ключам только в режиме Picture. Дальнейшие описания говорят о движении “на один столбец” и так далее, но все команды режима Picture обращаются с числовыми аргументами так же, как и их обычные эквиваленты.

Включение режима Picture вызывает ловушку `picture-mode-hook` (см. Раздел 31.2.3 [Ловушки], с. 349).

## 25.1 Основы редактирования в режиме Picture

Большинство ключей делают в режиме Picture то же самое, что они делают обычно, но в квадрантном стиле. Например, `C-f` перепривязывается на запуск команды `picture-forward-column`, которая передвигает точку на один столбец вправо, вставляя, если это необходимо, пробел, так что действительный конец строки не играет роли. `C-b` перепривязывается для запуска `picture-backward-column`, которая всегда двигает точку влево на один столбец, превращая знак табуляции в несколько пробелов, если это необходимо. `C-n` и `C-p` перепривязываются для запуска `picture-move-down` и `picture-move-up`, которые могут либо вставить пробелы, либо превратить знаки табуляции в пробелы, как необходимо, чтобы гарантировать, что точка остается в том же самом столбце. `C-e` запускает `picture-end-of-line`, которая передвигается за последний непустой знак на строке. Здесь не надо менять `C-a`, так как выбор модели экрана не влияет на начала строк.

Вставка текста приспосабливается к квадрантной модели экрана посредством использования режима Overwrite (см. Раздел 31.1 [Второстепенные режимы], с. 341). Самовставляющиеся знаки заменяют существующий текст, столбец за столбцом, а не выталкивают его вправо. ⟨RET⟩ запускает `picture-newline`, которая просто сдвигает на начало следующей строки, так что новый текст заменит эту строку.

Режим Picture предоставляет стирание текста вместо удаления и уничтожения. ⟨DEL⟩ (`picture-backward-clear-column`) заменяет предыдущий знак пробелом, а не удаляет его;

это смещает точку назад. C-d (picture-clear-column) заменяет следующий знак или знаки пробелами, но не сдвигает точку. (Если вы хотите превратить знаки в пробелы и продвинуться через них, используйте ⟨SPC⟩.) C-k (picture-clear-line) действительно уничтожает содержимое строки, но никогда не удаляет из буфера знаки перевода строки.

Чтобы сделать действительную вставку, вы должны использовать специальные команды. C-o (picture-open-line) создает пустую строку после текущей; она никогда не разбивает строку. C-M-o (split-line) имеет смысл в режиме Picture, поэтому она не изменяется. C-j (picture-duplicate-line) вставляет ниже текущей строки другую строку с тем же самым содержимым.

Чтобы сделать в режиме Picture действительное удаление, используйте C-w или C-c C-d (которая определяется для запуска delete-char, как C-d в других режимах), или с помощью одной из команд прямоугольника для рисунков (см. Раздел 25.4 [Прямоугольники в рисунке], с. 265).

## 25.2 Управление движением после вставки

Так как “самовставляющиеся” знаки в режиме Picture перезаписывают и передвигают точку, то нет существенного ограничения на то, как точка будет передвигаться. Обычно точка перемещается вправо, но вы можете определить любое из восьми ортогональных или диагональных направлений для движения после “самовставляющегося” знака. Это удобно для рисования линий в буфере.

C-c <
: Сдвинуться после вставки влево (picture-movement-left).

C-c >
: Сдвинуться после вставки вправо (picture-movement-right).

C-c ^
: Сдвинуться после вставки вверх (picture-movement-up).

C-c .
: Сдвинуться после вставки вниз (picture-movement-down).

C-c ‘
: Сдвинуться после вставки вверх и влево (на “северо-запад”) (picture-movement-nw).

C-c ’
: Сдвинуться после вставки вверх и вправо (на “северо-восток”) (picture-movement-ne).

C-c /
: Сдвинуться после вставки вниз и влево (на “юго-запад”) (picture-movement-sw).

C-c \
: Сдвинуться после вставки вниз и вправо (на “юго-восток”) (picture-movement-se).

Две команды движения передвигают, основываясь на текущем направлении вставки. Команда C-c C-f (picture-motion) передвигает в том же самом направлении, в каком выполняется движение после текущей “вставки”, тогда как C-c C-b (picture-motion-reverse) двигает в противоположном направлении.

## 25.3 Знаки табуляции в режиме Picture

В режиме Picture предусмотрены два вида действий, подобных табуляции. Для табуляции, основанной на контексте, используйте M-⟨TAB⟩ (picture-tab-search). Без аргумента она передвигает в точку под следующий “интересный” знак, который следует за незначащим знаком в предыдущей непустой строке. “Следующий” означает здесь “появляющийся в более дальней горизонтальной позиции, чем та, с которой точка отправлялась”. С аргументом, как в C-u M-⟨TAB⟩, эта команда переходит к следующему такому интересному знаку в текущей строке. M-⟨TAB⟩ не изменяет текст, она только двигает точку. “Интересные” знаки определяются переменной picture-tab-chars, которая должна задавать набор

знаков. Синтаксис этой переменной похож на синтаксис, используемый внутри '`[...]`' в регулярном выражении, но без '`[`' и '`]`'. Ее значение по умолчанию равно `"!-~"`.

Сама ⟨TAB⟩ запускает `picture-tab`, которая действует, основываясь на установленных текущих позициях табуляции; это эквивалент `tab-to-tab-stop` в режиме Picture. Обычно она просто двигает точку, но с числовым аргументом она стирает текст, через который передвинулась.

Формы табуляции, основанные на контексте и на позициях табуляции, объединяются вместе командой `C-c` ⟨TAB⟩, `picture-set-tab-stops`. Эта команда устанавливает позиции табуляции, которые `M-`⟨TAB⟩ считала бы значимыми в текущей строке. Использование этой команды вместе с ⟨TAB⟩ может дать эффект табуляции, основанной на контексте. Но `M-`⟨TAB⟩ более удобна в тех случаях, когда ее достаточно.

Может оказаться удобным запретить использование настоящих знаков табуляции в рисунках. Например, это помешает `C-x` ⟨TAB⟩ испортить рисунок. Вы можете сделать так, установив переменную `indent-tabs-mode` в значение `nil`. См. Раздел 20.3 [Только пробелы], с. 179.

## 25.4 Команды прямоугольника в режиме Picture

Режим Picture определяет команды для работы с прямоугольными кусками текста таким способом, который подходит для квадрантной модели. Стандартные команды для прямоугольников также могут быть полезны (см. Раздел 9.4 [Прямоугольники], с. 74).

`C-c C-k` Очистить текущую область-прямоугольник (`picture-clear-rectangle`). С аргументом — удалить ее.

`C-c C-w` *r* То же самое, но сначала сохранить содержимое прямоугольника в регистре *r* (`picture-clear-rectangle-to-register`).

`C-c C-y` Скопировать последний уничтоженный прямоугольник в буфер путем перезаписи, при этом левый верхний угол находится в точке (`picture-yank-rectangle`). С аргументом — вставка вместо перезаписи.

`C-c C-x` *r* То же самое, но использовать прямоугольник из регистра *r* (`picture-yank-rectangle-from-register`).

Команды работы с прямоугольниками для рисунков, это `C-c C-k` (`picture-clear-rectangle`) и `C-c C-w` (`picture-clear-rectangle-to-register`), отличаются от стандартных команд прямоугольника тем, что они обычно очищают прямоугольник вместо его удаления; это аналогично тому, как в режиме Picture изменяется `C-d`.

Однако, удаление прямоугольников может быть удобным и в режиме Picture, поэтому эти команды удаляют прямоугольник, если им передан числовой аргумент. `C-c C-k` с числовым аргументом или без него сохраняет прямоугольник для `C-c C-y`.

Команды режима Picture для восстановления прямоугольников отличаются от стандартных тем, что они перезаписывают, а не вставляют. Точно так же вставка другого текста в режиме Picture отличается от вставки в других режимах. `C-c C-y` (`picture-yank-rectangle`) вставляет (при помощи перезаписи) прямоугольник, который был уничтожен самым последним, в то время как `C-c C-x` (`picture-yank-rectangle-from-register`) делает то же самое для прямоугольника, находящегося в указанном регистре.

# 26 Посылка почты

`C-x m` Начать составление сообщения для пересылки (`compose-mail`).

`C-x 4 m` Аналогично, но показать сообщение в другом окне (`compose-mail-other-window`).

`C-x 5 m` Аналогично, но создать новый фрейм (`compose-mail-other-frame`).

`C-c C-s` В режиме Mail, посылает сообщение (`mail-send`).

`C-c C-c` Посылает сообщение и прячет его буфер (`mail-send-and-exit`).

Команда `C-x m` (`compose-mail`) выбирает буфер с именем '`*mail*`' и инициализирует его наброском исходящего сообщения. `C-x 4 m` (`compose-mail-other-window`) выбирает буфер '`*mail*`' в другом окне, оставляя предыдущий текущий буфер видимым. `C-x 5 m` (`compose-mail-other-frame`) создает для буфера '`*mail*`' новый фрейм.

Поскольку буфер сообщения — это обычный буфер Emacs, во время составления письма вы можете переключаться в другие буферы и вернуться обратно позднее (или вообще не возвращаться). Если вы используете команду `C-x m` снова, когда вы составляли другое сообщение, но еще не послали его, от вас потребуют подтверждение, перед тем как удалить старое сообщение. Если вы ответите `n`, буфер '`*mail*`' останется выбранным со своим старым содержимым, так что вы сможете закончить прежнее сообщение и послать его. `C-u C-x m` — это другой способ сделать то же самое. Пересылка сообщения помечает буфер '`*mail*`' как "немодифицированный", что устраняет необходимость подтверждения при следующем использовании `C-x m`.

Если вы составляете сообщение в буфере '`*mail*`' и хотите послать еще одно сообщение до завершения первого, переименуйте буфер '`*mail*`' с помощью `M-x rename-uniquely` (см. Раздел 15.3 [Буферы Разное], с. 136). Затем вы можете использовать `C-x m` или ее варианты, описанные выше, чтобы создать новый буфер '`*mail*`'. Если вы сделаете так, то сможете работать с каждым буфером с сообщением независимо.

## 26.1 Формат буфера с почтовым сообщением

Кроме *текста*, или *тела*, сообщение имеет *поля заголовка*, которые говорят, кто послал его, когда, кому, зачем и так далее. Некоторые поля заголовка, такие как '`Date`' (дата) и '`Sender`' (отправитель), создаются автоматически, когда вы посылаете сообщение. Другие же, например имена получателей, должны быть заданы вами, чтобы сообщение было отослано правильно.

Режим Mail предусматривает несколько команд, помогающих вам отредактировать некоторые поля заголовка, и некоторые поля в этом буфере иногда инициализируются автоматически. Вы можете вставить или отредактировать любые поля заголовка, используя обычные команды редактирования.

Строка в буфере, которая гласит

```
-text follows this line-
```

— это специальный ограничитель, отделяющий заданные вами заголовки от самого текста. Все, что следует за этой строкой, — это текст сообщения, а заголовки предшествуют ей. Сама разделяющая строка не появляется в реальном посылаемом сообщении. Используемый для строки-ограничителя текст задается переменной `mail-header-separator`.

Здесь представлен пример того, как могут выглядеть заголовки и текст в буфере сообщения.

```
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 Поля заголовка сообщения

Каждое поле заголовка в буфере сообщения начинается с имени это поля, оно пишется в начале строки и отделяется двоеточием. Прописные и строчные буквы в именах полей не различаются (и в почтовых адресах тоже). После двоеточия и необязательного пропуска пишется содержимое этого поля.

Вы можете использовать для полей заголовка любые имена, какие вам нравятся, но обычно люди используют только стандартные имена с принятыми значениями. Ниже приведена таблица часто используемых в посылаемых сообщениях полей.

‘`To`’ Это поле содержит почтовые адреса, на которые посылается сообщение. Если вы перечисляете более одного адреса, используйте для их разделения запятые, а не пробелы.

‘`Subject`’ Содержанием поля ‘`Subject`’ должен быть фрагмент текста, который указывает, о чем будет сообщение. Смысл полей ‘`Subject`’ в том, что большинство программ чтения почты могут предоставлять обзорный перечень сообщений, печатая только тему каждого сообщения, а не весь текст.

‘`CC`’ Это поле, как и поле ‘`To`’, содержит дополнительные почтовые адреса, по которым нужно отправить сообщение, но эти читатели не должны рассматривать это сообщение как адресованное им.

‘`BCC`’ Это поле содержит дополнительные почтовые адреса, по которым отправляется сообщение, но они не должны появляться в заголовке фактически посланного сообщения. Посланные таким образом копии называются *слепыми копиями*.

Чтобы вам посылалась слепая копия каждого исходящего сообщения, установите переменную `mail-self-blind` равной `t`.

‘`FCC`’ Это поле содержит имя файла, оно велит Emacs добавить после отправки копию этого сообщения в заданный файл. Если это файл в формате Rmail, Emacs записывает сообщение в формате Rmail; в противном случае — в системном формате почтовых файлов.

Чтобы каждый раз, когда вы начинаете редактирование исходящего сообщения, в поле ‘`FCC`’ помещалось фиксированное имя файла, установите переменную `mail-archive-file-name` равной имени этого файла. Если вы не удалите поле ‘`FCC`’ перед посылкой, сообщение будет записано в этот файл.

‘`From`’ Используйте поле ‘`From`’, чтобы назвать себя, когда вы отправляете почту, войдя в систему под чужим именем. Содержимое поля ‘`From`’ должно быть правильным почтовым адресом, поскольку обычно ответы направляются по этому адресу. Если вы не задали поле ‘`From`’ сами, Emacs использует значение `user-mail-address` в качестве значения по умолчанию.

‘`Reply-to`’

Используйте это поле, чтобы направлять ответы по адресам, отличным от вашего собственного. Большинство программ чтения почты (включая Rmail) автоматически посылают ответы по адресу ‘`Reply-to`’; он имеет приоритет перед адресом ‘`From`’. Добавляя в заголовок поле ‘`Reply-to`’, вы можете избавиться от любых проблем, которые может вызывать при ответе ваш адрес ‘`From`’.

Чтобы какой-то адрес для ‘Reply-to’ помещался в каждое исходящее сообщение, установите переменную mail-default-reply-to равной этом адресу (в виде строки). Тогда mail инициализирует сообщения с заданным адресом ‘Reply-to’. Вы можете удалить или изменить это поле заголовка перед отправкой сообщения, если хотите. Когда Emacs начинает работу, mail-default-reply-to инициализируется по переменной среды REPLYTO, если она установлена.

‘In-reply-to’
: Это поле содержит фрагмент текста, описывающий сообщение, на которое вы отвечаете. Некоторые почтовые системы могут использовать эту информацию для соотношения связанных между собой фрагментов почты. Обычно это поле заполняется самим Rmail, когда вы отвечаете на сообщение из него, и вам никогда не придется думать об этом (см. Глава 27 [Rmail], с. 275).

‘References’
: В этом поле перечисляются ID предыдущих сообщений, связанных с этим. Rmail устанавливает это поле автоматически, когда вы отвечаете на какое-то сообщение.

Поля заголовка ‘To’, ‘CC’, ‘BCC’ и ‘FCC’ могут использоваться любое число раз, и каждое из этих полей может содержать несколько адресов, разделенных запятыми. Поля ‘To’, ‘CC’ или ‘BCC’ могут также иметь строки продолжения: одна или несколько строк, начинающиеся с пробельных знаков и следующие за строкой, на которой начинается поле, рассматриваются как часть этого поля. Вот пример поля ‘To’ со строкой продолжения:

```
To: foo@here.net, this@there.net,
  me@gnu.cambridge.mass.usa.earth.spiral3281
```

При посылке сообщения, если вы не написали поле ‘From’ сами, Emacs сделает это за вас. Формат этого поля управляется переменной mail-from-style:

`nil`
: Только почтовый адрес, как ‘king@grassland.com’.

`parens`
: И почтовый адрес, и полное имя, как ‘king@grassland.com (Elvis Parsley)’.

`angles`
: То же, но как ‘Elvis Parsley <king@grassland.com>’.

`system-default`
: Позволить системе самой вставить поле ‘From’.

## 26.3 Почтовые псевдонимы

Вы можете определить *почтовые псевдонимы* в файле с именем ‘~/.mailrc’. Это короткие мнемонические имена, обозначающие почтовые адреса или группы адресов. Подобно многим другим почтовым программам, Emacs раскрывает псевдонимы, когда они появляются в полях ‘To’, ‘From’, ‘CC’, ‘BCC’ и ‘Reply-to’ и в их вариантах с ‘Resent-’.

Чтобы определить псевдоним в ‘~/.mailrc’, напишите одну строку в таком формате:

```
alias короткий-адрес полные-адреса
```

Здесь *полные-адреса* означает один или более почтовых адресов, в которые раскрывается *короткий-адрес*. Разделяйте адреса пробелами; если адрес содержит пробел, заключайте весь адрес в двойные кавычки.

Например, чтобы сделать так, чтобы maingnu обозначало ваш собственный местный адрес и gnu@gnu.org, поместите такую строку:

```
alias maingnu gnu@gnu.org local-gnu
```

Emacs также распознает в файлах ‘.mailrc’ команды включения. Они выглядят так:

`source` *имя-файла*

Файл '`~/.mailrc`' в основном используется другими программами для чтения почты; он может содержать различные другие команды. Emacs игнорирует все, кроме определений псевдонимов и команд включения.

Есть другой способ определить почтовый псевдоним, но только внутри Emacs — с помощью команды `define-mail-alias`. Она запрашивает псевдоним и затем полный адрес. Вы можете использовать ее для определения псевдонимов в вашем файле '`.emacs`', следующим образом:

```
(define-mail-alias "maingnu" "gnu@gnu.org")
```

`define-mail-alias` записывает псевдонимы, добавляя их к переменной, называемой `mail-aliases`. Если вы умеете обращаться со списками в Лиспе, вы можете установить `mail-aliases` напрямую. Первоначальное значение переменной `mail-aliases` равно `t`, что означает, что для получения правильного значения Emacs должен считать '`.mailrc`'.

Вы можете задать вместо '`~/.mailrc`' файл с другим именем, установив переменную `mail-personal-alias-file`.

Обычно Emacs раскрывает псевдонимы, когда вы отсылаете сообщение. Вам не обязательно раскрывать псевдонимы до отсылки сообщения, но вы можете раскрыть их, если хотите увидеть, куда пойдет это письмо. Чтобы сделать это, используйте команду `M-x expand-mail-aliases`; она раскрывает все почтовые псевдонимы, присутствующие в данный момент в полях заголовка с адресами.

Если хотите, вы можете сделать так, чтобы почтовые псевдонимы раскрывались как сокращения, по мере того как вы их набираете (см. Глава 24 [Сокращения], с. 257). Чтобы включить эту возможность, выполните следующее:

```
(add-hook 'mail-setup-hook 'mail-abbrevs-setup)
```

Это можно написать в файле '`.emacs`'. См. Раздел 31.2.3 [Ловушки], с. 349. Если вы пользуетесь этим методом, вы должны использовать `define-mail-abbrev`, а не `define-mail-alias`; последняя команда не работает с этим пакетом. Обратите внимание, пакет для сокращений почтовых адресов использует переменную `mail-abbrevs` вместо `mail-aliases`, и все псевдонимы преобразуются к нижнему регистру.

Пакет для сокращений почтовых адресов также предоставляет команду `C-c C-a` (`mail-interactive-insert-alias`), которая считывает псевдоним (с завершением) и вставляет его определение в точке. Это полезно при редактировании самого текста сообщения или поля вроде '`Subject`', где Emacs обычно не раскрывает псевдонимы.

Заметьте, что сокращения раскрываются, только если вы после вставили разделитель слов. Однако, вы можете перепривязать `C-n` и `M->` так, чтобы они еще и производили раскрытие. Это можно сделать следующим образом:

```
(add-hook 'mail-setup-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 Режим Mail

Основной режим, используемый в буфере сообщения — это режим Mail, который очень похож на режим Text за исключением того, что в нем предоставляются различные специальные команды на префиксе `C-c`. Все эти команды должны производить различные

операции с редактируемым или посылаемым сообщением. Кроме того, режим Mail определяет знак '%' как разделитель слов; это полезно при использовании команд, работающих со словами, для редактирования почтовых адресов.

Режим Mail обычно используется в буферах, автоматически подготовленных командой `mail` или родственными командами. Однако, вы можете также переключиться в режим Mail в буфере, обращающемся к файлу. Это полезно делать, если вы сохранили в файле черновик сообщения.

### 26.4.1 Отправка почты

В режиме Mail есть две команды для отправки сообщения, которое вы редактировали:

`C-c C-s` Послать это сообщение и оставить буфер с ним выбранным (`mail-send`).

`C-c C-c` Послать это сообщение и выбрать какой-то другой буфер (`mail-send-and-exit`).

`C-c C-s` (`mail-send`) отсылает сообщение и помечает буфер с этим сообщением как немодифицированный, но при этом оставляет этот буфер выбранным, так что вы можете изменить это сообщение (возможно с новыми получателями) и послать его снова.

`C-c C-c` (`mail-send-and-exit`) отсылает сообщение, а затем удаляет окно или переключает в другой буфер. Эта команда устанавливает для буфера с сообщением самый низкий приоритет для повторного выбора, так как вы перестали его использовать. Это обычный способ отправки сообщения.

В буфере, который обращается к файлу, отправка сообщения не сбрасывает флаг измененности, так как это должно делать только сохранение файла. В результате вы не получаете предупреждения, если пытаетесь отправить одно сообщение два раза.

Когда вы отправляете сообщение, которое содержит знаки, не входящие в ASCII, эти знаки необходимо перевести в какую-то систему кодирования (см. Раздел 18.7 [Системы кодирования], с. 165). Обычно она устанавливается вашей языковой средой (см. Раздел 18.3 [Языковые среды], с. 162). Вы можете явно указать систему кодирования для исходящей почты, устанавливая переменную `sendmail-coding-system`.

Если определяемая таким образом система кодирования не может обработать знаки в каком-то сообщении, Emacs просит вас выбрать систему кодирования самим, показывая список возможных вариантов.

### 26.4.2 Редактирование заголовка сообщения

Режим Mail предоставляет специальные команды для передвижения к определенным полям заголовка и для завершения адресов.

`C-c C-f C-t`
Перейти к полю заголовка '`To`', создавая его, если его еще нет (`mail-to`).

`C-c C-f C-s`
Перейти к полю заголовка '`Subject`', создавая его, если его еще нет (`mail-subject`).

`C-c C-f C-c`
Перейти к полю заголовка '`CC`', создавая его, если его еще нет (`mail-cc`).

`C-c C-f C-b`
Перейти к полю заголовка '`BCC`', создавая его, если его еще нет (`mail-bcc`).

`C-c C-f C-f`
Перейти к полю заголовка '`FCC`', создавая его, если его еще нет (`mail-fcc`).

`M-`⟨TAB⟩ Завершить почтовый адрес (`mail-complete`).

Есть пять команд для перемещения точки к отдельным полям заголовка, и все они базируются на префиксе `C-c C-f` (‘`C-f`’ означает “field”.[1] Они перечислены в таблице выше. Если требуемое поле не существует, эти команды создают его. Мы предоставляем специальные команды для перемещения именно к таким полям, потому что чаще всего пользователи хотят редактировать именно их.

При редактировании полей заголовка, которые содержат почтовые адреса, таких как ‘`To:`’, ‘`CC:`’ и ‘`BCC:`’, вы можете завершить адрес, введя `M-`⟨TAB⟩ (`mail-complete`). Эта команда вставляет полное имя, соответствующее этому адресу, если она может его определить. Переменная `mail-complete-style` говорит, нужно ли вставлять полное имя и какой стиль следует использовать; стиль задается как в `mail-from-style` (см. Раздел 26.2 [Заголовки сообщений], с. 268).

При завершении правильными почтовыми адресами считаются имена локальных пользователей и определенные вами почтовые псевдонимы. Вы можете задать дополнительные источники правильных адресов; используйте буфер настройки, чтобы просмотреть возможные варианты.

Если вы напечатаете `M-`⟨TAB⟩ в теле сообщения, она вызовет `ispell-complete-word`, как в режиме Text.

### 26.4.3 Цитирование почты

В режиме Mail также есть команды для восстановления или *цитирования* всего или части сообщения, на которое вы отвечаете. Эти команды активны, только когда вы начали отправку сообщения с использованием команды Rmail.

`C-c C-y` Восстановить выбранное сообщение из Rmail (`mail-yank-original`).

`C-c C-r` Восстановить область из буфера Rmail (`mail-yank-region`).

`C-c C-q` Заполнить все абзацы, процитированные из другого сообщения (`mail-fill-yanked-message`).

Если отправка почты запускается из программы чтения почты Rmail с использованием команды Rmail, то внутри буфера сообщения может использоваться команда `C-c C-y` для вставки текста сообщения, на которое вы отвечаете. Обычно она сдвигает каждую строку этого сообщения на три пробела и удаляет большинство полей заголовка. Числовой аргумент указывает количество пробелов для отступа. Просто `C-u` говорит о том, что делать отступ и удалять что-либо не надо. `C-c C-y` всегда использует текущее сообщение из буфера Rmail, так что можно вставить несколько старых сообщений, выбирая нужное в Rmail, переключаясь в ‘`*mail*`’ и восстанавливая его, а затем снова переключаясь в Rmail, чтобы выбрать еще одно.

Вы можете задать текст, который команда `C-c C-y` будет вставлять в начале каждой строки: установите `mail-yank-prefix` равной желаемой строке. (Значение `nil` означает, что следует делать отступ; это используется по умолчанию.) Однако, `C-u C-c C-y` никогда не добавляет ничего в начало вставляемых строк, несмотря на значение `mail-yank-prefix`.

Чтобы вставить только часть пришедшего сообщения, установите в Rmail область вокруг нужного фрагмента; затем перейдите в буфер ‘`*mail*`’ и напечатайте `C-c C-r` (`mail-yank-region`). В каждой копируемой строке делается отступ или добавляется префикс в соответствии с `mail-yank-prefix`.

После использования `C-c C-y` или `C-c C-r` вы можете набрать `C-c C-q`, чтобы заполнить абзацы восстановленного старого сообщения или сообщений. Однократным использованием `C-c C-q` заполняются все такие абзацы, причем каждый отдельно. Чтобы заполнить

[1] От англ. “поле”. *(Прим. переводчика)*

один абзац процитированного сообщения, используйте `M-q`. Если заполнение не обрабатывает используемый вами стиль префикса для цитат автоматически, попробуйте установить префикс заполнения явно. См. Раздел 21.5 [Заполнение], с. 185.

### 26.4.4 Другие возможности режима Mail

`C-c C-t` Перемещает к началу текста тела сообщения (`mail-text`).

`C-c C-w` Вставляет файл '`~/.signature`' в конец текста сообщения (`mail-signature`).

`C-c C-i` *файл* ⟨RET⟩
: Вставляет содержимое *файла* в конец исходящего сообщения (`mail-attach-file`).

`M-x ispell-message`
: Производит проверку правописания в тексте сообщения, но не в цитатах из других сообщений.

`C-c C-t` (`mail-text`) перемещает точку к позиции сразу после строки-разделителя заголовка, то есть к началу текста тела сообщения.

`C-c C-w` (`mail-signature`) добавляет в конец сообщения стандартный кусок текста, где вы можете подробнее рассказать с себе. Этот текст берется из файла '`~/.signature`' в вашем начальном каталоге. Чтобы подпись вставлялась автоматически, установите переменную `mail-signature` в значение `t`; тогда при создании почтового сообщения содержимое вашего файла '`~/.signature`' будет вставляться автоматически. Если вы не хотите ставить подпись в конкретном сообщении, удалите ее из буфера перед отсылкой.

Вы также можете установить `mail-signature` равной строке; тогда эта строка автоматически вставляется в качестве вашей подписи, когда вы начинаете редактировать сообщение. Если вы установите ее равной какому-то другому лисповскому выражению, это выражение всякий раз вычисляется, а его значение (которое должно быть строкой) определяет подпись.

Вы можете проверить орфографию в тексте написанного сообщения с помощью команды `M-x ispell-message`. Если вы восстанавливали пришедшие сообщения в набросок отправляемого, эта команда пропускает восстановленные места, но проверяет текст, который вставили вы сами. (Она просматривает величину отступа или значение переменной `mail-yank-prefix`, чтобы отличить процитированные строки от введенных вами.) См. Раздел 13.4 [Правописание], с. 102.

Чтобы включить в отправляемое сообщение файл, вы можете использовать `C-x i`, обычную команду для вставки файла в текущий буфер. Но чаще удобнее использовать особую команду, `C-c C-i` (`mail-attach-file`). Эта команда вставляет содержимое заданного файла в конец буфера после подписи, если она есть, с разделяющей строкой, включающей имя этого файла.

Включение режима Mail (что `C-x m` делает автоматически) запускает обычные ловушки `text-mode-hook` и `mail-mode-hook`. Инициализация нового исходящего сообщения запускает обычную ловушку `mail-setup-hook`; используйте эту ловушку, если вы хотите добавить к вашему почтовому заголовку особые поля или сделать другие изменения в представлении буфера сообщения. См. Раздел 31.2.3 [Ловушки], с. 349.

Основное различие между этими ловушками состоит только в том, в какое время они вызываются. Когда вы набираете `M-x mail`, запускается `mail-mode-hook`, как только будет создан буфер '`*mail*`'. Затем функция `mail-setup` помещает в этот буфер его начальное содержимое по умолчанию. После этого запускается `mail-setup-hook`.

## 26.5 Как сбить с толку NSA

`M-x spook` добавляет в отправляемое сообщение строку случайно выбранных ключевых слов. Эти ключевые слова выбираются из списка слов, которые предполагают, что вы обсуждаете что-то .

За этой возможностью стоит идея, что NSA[2] просматривает электронную почту, содержащую ключевые слова, которые они могут найти интересными. (NSA говорит, что они этого не делают, но они *должны* так говорить.) Идея состоит в том, что если многие люди добавляют к своим сообщениям подозрительные слова, NSA будет настолько занято подделками, что вынуждено будет совсем прекратить просмотр.

Вот как автоматически вставлять зловещие ключевые слова, когда вы начинаете набирать сообщение:

```
(add-hook 'mail-setup-hook 'spook)
```

Смущает это NSA или нет, по крайней мере это развлекает людей.

## 26.6 Способы составления сообщений

Эта глава описывает обычный режим Emacs для редактирования и отправки почты — режим Mail. В Emacs есть иные способы для этого, включая режимы MH-E и Message, не описанные в этом руководстве. Вы можете выбрать любой из них в качестве своего предпочтительного способа. Команды `C-x m`, `C-x 4 m` и `C-x 5 m` используют тот агент, который вы задали. Так же делают и другие команды и программы Emacs, посылающие почту.

Чтобы указать ваш способ составления сообщений, установите переменную `mail-user-agent`. На данный момент допустимые значения включают `sendmail-user-agent`, `mh-e-user-agent` и `message-user-agent`.

Если вы выбрали другой способ составления сообщений, информация о буфере ‘`*mail*`’ и режиме Mail из этой главы не относится к вашему случаю; другие методы могут использовать совершенно иные команды с иным форматом в иначе называемом буфере.

[2] National Security Agency, американский аналог ФСБ с ее СОРМ2. *(Прим. переводчика)*

# 27 Чтение почты с помощью Rmail

Rmail — это подсистема Emacs для чтения и размещения получаемой вами почты. Rmail хранит почтовые сообщения в файлах, называемых Rmail-файлами. Чтение сообщений в Rmail-файле осуществляется в специальном основном режиме, режиме Rmail, который переопределяет большинство букв для запуска команд управления почтой. Команда `rmail-mode` используется для входа в режим Rmail, она запускает ловушку `rmail-mode-hook`, как обычно; но не выполняйте эту команду вручную, она не может делать ничего существенного, если этот буфер не обращается к правильному Rmail-файлу.

## 27.1 Основные понятия Rmail

При простейшем использовании Rmail, у вас есть один Rmail-файл '`~/RMAIL`', в котором сохраняется вся ваша почта. Этот файл называется *первичным Rmail-файлом*. Команда `M-x rmail` считывает ваш первичный Rmail-файл, вставляет в него новую почту из ваших входных файлов, отображает первое непрочитанное сообщение и позволяет вам начать чтение. Переменная `rmail-file-name` задает имя первичного Rmail-файла.

Rmail использует сужение, чтобы скрыть из Rmail-файла все сообщения, кроме одного. Показанное сообщение называется *текущим*. Специальные команды режима Rmail могут осуществлять такие вещи, как удаление текущего сообщения, копирование его в другой файл, посылка ответа или перемещение к другому сообщению. Вы также можете создать несколько Rmail-файлов и использовать Rmail для обмена сообщениями между ними.

В пределах Rmail-файла сообщения расположены последовательно в порядке поступления; вы можете использовать другие способы сортировки. Сообщениям присваиваются последовательные целые числа в качестве их *номеров сообщений*. Номер текущего сообщения показывается в строке режима Rmail, за ним идет общее число сообщений в файле. Вы можете перейти на сообщение, задавая его номер с помощью ключа `j` (см. Раздел 27.3 [Перемещение в Rmail], с. 276).

Следуя обычным правилам Emacs, изменения в Rmail-файле становятся постоянными только после сохранения этого файла. Вы можете осуществить это с помощью команды `s` (`rmail-save`), которая также сперва вычеркивает удаляемые сообщения из файла (см. Раздел 27.4 [Удаление сообщений], с. 277). Для записи файла без вычеркивания следует использовать `C-x C-s`. Rmail также сохраняет Rmail-файл при поступлении новой почты из входного почтового файла (см. Раздел 27.5 [Входной почтовый файл], с. 278).

Выйти из Rmail можно с помощью `q` (`rmail-quit`), при этом Rmail-файл очищается и сохраняется, и происходит переход в другой буфер. Но формально нет необходимости 'выходить'. Если вы переключились из Rmail к редактированию в других буферах и больше не возвращались обратно, на самом деле вы вышли. (Команда Rmail `b` делает это для вас.) Достаточно просто убедиться, что вы сохранили Rmail-файл (как и любой другой измененный вами файл). Достаточно удобный способ сделать это предоставляет команда `C-x s` (см. Раздел 14.3 [Сохранение], с. 108).

## 27.2 Прокрутка в сообщении

Когда Rmail показывает сообщение, не помещающееся на экране, вам придется прокручивать это сообщение, чтобы прочитать оставшуюся часть. Вы могли бы сделать это с помощью `C-v`, `M-v` и `M-<`, но в Rmail прокруткой приходится пользоваться настолько часто, что ее команды заслуживают более простого набора.

⟨SPC⟩ Прокручивает вперед (`scroll-up`).

⟨DEL⟩ Прокручивает назад (`scroll-down`).

. Прокручивает к началу сообщения (`rmail-beginning-of-message`).

Так как во время чтения сообщения наиболее частой процедурой является прокрутка по целому экрану, Rmail делает ⟨SPC⟩ и ⟨DEL⟩ синонимами `C-v` (`scroll-up`) и `M-v` (`scroll-down`).

Команда `.` (`rmail-begining-of-message`) прокручивает к началу выбранного сообщения. Это не совсем то же, что и `M-<`: во-первых, она не оставляет метку, а во-вторых, она переустанавливает границы буфера до пределов текущего сообщения, если вы их изменили.

## 27.3 Перемещение по сообщениям

Самое основное, что вы можете сделать с сообщением — прочитать его. В Rmail вы можете прочитать сообщение, сделав его текущим. Обычно при этом последовательно перемещаются по файлу, так как сообщения в нем расположены в порядке получения. При входе в Rmail вы попадаете на первое сообщение, которое никогда еще не было текущим (это первое сообщения, имеющее атрибут ‘`unseen`’; см. Раздел 27.9 [Атрибуты сообщений], с. 281). Чтобы увидеть другие новые сообщения, двигайтесь вперед, для повторного просмотра старых двигайтесь назад.

`n` Перейти к следующему неудаленному сообщению, пропуская все промежуточные удаленные сообщения (`rmail-next-undeleted-message`).

`p` Перейти к предыдущему неудаленному сообщению (`rmail-previous-undeleted-message`).

`M-n` Перейти к следующему сообщению, включая удаленные (`rmail-next-message`).

`M-p` Перейти к предыдущему сообщению, включая удаленные (`rmail-previous-message`).

`j` Перейти к первому сообщению. С аргументом *n* — перейти к сообщению с номером *n* (`rmail-show-message`).

`>` Перейти к последнему сообщению (`rmail-last-message`).

`<` Перейти к первому сообщению (`rmail-first-message`).

`M-s` *regexp* ⟨RET⟩
Перейти к следующему сообщению, содержащему совпадение с регулярным выражением *regexp* (`rmail-search`).

`-` `M-s` *regexp* ⟨RET⟩
Перейти к предыдущему сообщению, содержащему совпадение с *regexp*.

`n` and `p` — это обычный способ перемещения по сообщениям в Rmail. Они перемещают по сообщениям последовательно, но пропускают удаленные сообщения, что обычно вы и хотели бы делать. Эти команды называются `rmail-next-undeleted-message` и `rmail-previous-undeleted-message`. Если вы не хотите пропускать удаленные сообщения, например, если вы хотите переместиться к сообщению, чтобы отменить его удаление, используйте варианты `M-n` и `M-p` (`rmail-next-message` и `rmail-previous-message`). Числовой аргумент в любой из этих команд используется как счетчик повторов.

В Rmail вы можете задать числовой аргумент, набрав только цифру. При этом не требуется сначала набирать `C-u`.

Команда `M-s` (`rmail-search`) — это версия поиска для Rmail. Обычная команда наращиваемого поиска `C-s` работает в Rmail, но она осуществляет поиск только в пределах текущего сообщения. Цель команды `M-s` — поиск другого сообщения. Она считывает регулярное выражение (см. Раздел 12.5 [Регулярные выражения], с. 91) без наращивания и затем осуществляет поиск совпадения, начиная с начала следующего сообщения. Потом она выбирает сообщение, содержащее совпадение. Если *regexp* пусто, `M-s` использует регулярное выражение, заданное при предыдущем поиске.

Чтобы найти в файле другое сообщения в обратном направлении, задайте команде `M-s` отрицательный аргумент. В Rmail это можно сделать как `- M-s`.

Также возможен поиск сообщения по метке. См. Раздел 27.8 [Метки сообщений], с. 281.

Для передвижения к сообщению, задаваемому абсолютным номером, используйте команду `j` (`rmail-show-message`) с номером сообщения в качестве аргумента. Без аргумента, команда `j` выбирает первое сообщение. `<` (`rmail-first-message`) также выбирает первое сообщение. Команда `>` (`rmail-last-message`) выбирает последнее.

## 27.4 Удаление сообщений

Когда вам больше не нужно хранить какое-то сообщение, его можно *удалить*. При этом удаляемое сообщение метится как возможно игнорируемое и некоторые команды Rmail не будут замечать его присутствия, но тем не менее оно продолжает существовать в Rmail-файле и сохраняет свой номер.

*Вычеркивание* в Rmail-файле (или его *очистка*) действительно уничтожает удаленные сообщения. Остальные сообщения последовательно перенумеровываются. Вычеркивание — это единственное действие, кроме разбора дайджеста (см. Раздел 27.15 [Дайджест сообщений], с. 287), которое изменяет номера сообщений.

`d` Удалить текущее сообщение и перейти к следующему неудаленному (`rmail-delete-forward`).

`C-d` Удалить текущее сообщение и перейти к предыдущему неудаленному (`rmail-delete-backward`).

`u` Отменить удаление текущего сообщения или перейти назад к удаленному сообщению и отменить его удаление (`rmail-undelete-previous-message`).

`x` Очистить Rmail-файл (`rmail-expunge`).

Существуют две команды Rmail для удаления сообщений. Обе они удаляют текущее сообщение и выбирают другое. Команда `d` (`rmail-delete-forward`) переходит к следующему сообщению, пропуская уже удаленные, в то время как `C-d` (`rmail-delete-backward`) передвигает к предыдущему неудаленному сообщению. Если же нет неудаленного сообщения, к которому можно перейти в указанном направлении, то текущим остается сообщение, которое было только что удалено. Числовой аргумент меняет направление движения после удаления.

Всякий раз, когда Rmail удаляет сообщение, он вызывает функции, перечисленные в `rmail-delete-message-hook`. Когда вызываются функции этой ловушки, сообщение уже помечено как удаленное, но все еще является текущим в этом буфере Rmail.

Для того чтобы все удаленные сообщения окончательно исчезли из Rmail-файла, надо набрать `x` (`rmail-expunge`). Пока это не сделано, есть возможность *отмены удаления* сообщений. Команда отмены удаления, `u` (`rmail-undelete-previous-message`), предназначена для отмены действия команды `d` в большинстве случаев. Она отменяет удаление текущего сообщения, если оно было удалено. В противном случае она двигается к предыдущему сообщению до тех пор, пока не будет найдено удаленное сообщение, и производит отмену удаления этого сообщения.

Обычно вы можете отменить действие `d` с помощью команды `u`, так как `u` передвигает назад и отменяет удаление сообщения, произведенное командой `d`. Но это не работает, когда `d` пропускает несколько уже удаленных сообщений, которые следуют за удаляемым сообщением; в этом случае команда `u` будет отменять удаление последнего сообщения из тех, что были пропущены. Не существует совершенного способа обойти эту проблему. Однако, повторяя команду `u`, можно в конце концов вернуться к сообщению, для которого вы собирались отменить удаление. Можно также добраться до этого сообщения с помощью команды `M-p` и затем набрать `u`.

Удаленное сообщение имеет атрибут ‘deleted’, и как результат при удалении текущего сообщения в строке режима появляется слово ‘deleted’. На самом деле, удаление или отмена удаления сообщения — это не более чем добавление или уничтожение этого атрибута. См. Раздел 27.9 [Атрибуты сообщений], с. 281.

## 27.5 Rmail-файлы и входные почтовые ящики

Операционная система помещает приходящую вам почту в файл, который мы называем вашим *входным почтовым ящиком*. Когда вы запускаете Rmail, он выполняет написанную на Си программу `movemail`, чтобы скопировать новые сообщения из входного почтового ящика в первичный Rmail-файл, который также содержит другие сообщения, сохранившиеся от предыдущих сеансов Rmail. Именно в этом файле находится та корреспонденция, которую вы читаете с помощью Rmail. Эта процедура называется *получением новой почты*. В любой момент она может быть повторена в Rmail с помощью команды `g`.

Переменная `rmail-primary-inbox-list` содержит список имен файлов, являющихся входными почтовыми ящиками вашего первичного Rmail-файла. Если вы не установили эту переменную явно, она инициализируется значением переменной среды `MAIL`, или, в крайнем случае, устанавливается в значение `nil`, что означает использование входного почтового ящика по умолчанию; это могут быть файлы ‘`/var/mail/`*имя-пользователя*’, ‘`/usr/mail/`*имя-пользователя*’ или ‘`/usr/spool/mail/`*имя-пользователя*’ в зависимости от вашей операционной системы.

Чтобы узнать значение по умолчанию для вашей системы, используйте `C-h v rmail-primary-inbox` ⟨RET⟩. Вы можете указать входной файл (или файлы) для любого Rmail-файла с помощью команды `set-rmail-inbox-list`; смотрите Раздел 27.6 [Rmail-файлы], с. 278.

Есть две причины для разделения Rmail-файлов и входных почтовых ящиков.

1. Формат входных почтовых файлов изменяется от одной операционной системы к другой и зависит от других почтовых программ. Только одна часть Rmail должна знать об этих альтернативах и должна лишь понимать, как преобразовывать их все к собственному формату Rmail.
2. Обеспечение доступа в файл входного почтового ящика без возникновения опасности потери почты — очень громоздкая вещь, поскольку требует взаимных блокировок с доставкой почты. Кроме того, различные операционные системы используют разную технику блокировок. Стратегия перемещения почты из входного почтового ящика раз и навсегда в отдельный Rmail-файл устраняет необходимость блокировки всего остального в Rmail, так как только Rmail работает с Rmail-файлом.

Rmail был написан с использованием Babyl в качестве внутреннего формата. С тех пор мы поняли, что обычный для систем Unix и GNU формат входных почтовых ящиков подходит для этой цели, и мы планируем изменить Rmail, чтобы он использовал этот формат. Однако, Rmail-файл все так же будет отдельным от входного файла, даже на системах, где их форматы одинаковы.

## 27.6 Множество почтовых файлов

По умолчанию Rmail действует в вашем *первичном Rmail-файле*, называемом ‘`~/RMAIL`’, и получает вашу приходящую почту из системного входного почтового файла. Но вы можете также иметь другие Rmail-файлы и редактировать их с помощью Rmail. Эти файлы могут получать почту через их собственные файлы входных почтовых ящиков, или вы можете перемещать в них сообщения с помощью явных команд Rmail (см. Раздел 27.7 [Вывод из Rmail], с. 279).

`i` *файл* ⟨RET⟩
: Считать *файл* в Emacs и запустить в нем Rmail (`rmail-input`).

`M-x set-rmail-inbox-list` ⟨RET⟩ *файлы* ⟨RET⟩
: Задать имена входных почтовых файлов, откуда будет получать почту текущий Rmail-файл.

`g`
: Получить новую почту из входных почтовых ящиков текущего Rmail-файла (`rmail-get-new-mail`).

`C-u g` *файл* ⟨RET⟩
: Получить новую почту из входного почтового ящика *файл*.

Чтобы запустить Rmail для файла, отличного от вашего первичного почтового файла, можно использовать в Rmail команду `i` (`rmail-input`). Она обращается к этому файлу в режиме Rmail. Вы также можете использовать команду `M-x rmail-input`, даже не находясь в Rmail.

Файл, считываемый с помощью команды `i`, как правило, должен быть правильным Rmail-файлом. Если это не так, Rmail пытается преобразовать его в поток сообщений в нескольких известных форматах. Если это удается, он преобразует весь этот файл в Rmail-файл. Если вы задали имя несуществующего файла, `i` инициализирует новый буфер для создания нового Rmail-файла.

Вы также можете выбрать Rmail-файл из меню. Сначала выберите пункт меню Classify, из меню Classify выберите пункт Input Rmail File; затем выберите нужный вам файл. Переменные `rmail-secondary-file-directory` и `rmail-secondary-file-regexp` указывают, какие файлы предлагает это меню: первая переменная говорит, в каком каталоге их искать; вторая говорит, какие файлы в этом каталоге предлагать (все, чьи имена соответствуют регулярному выражению). Эти переменные также относятся к выбору файла для вывода (см. Раздел 27.7 [Вывод из Rmail], с. 279).

Каждый Rmail-файл может содержать список имен файлов входных почтовых ящиков; вы можете задать этот список с помощью `M-x set-rmail-inbox-list` ⟨RET⟩ `files` ⟨RET⟩. Аргумент может содержать любое число имен файлов, разделенных запятыми. Он может быть также пустым, и это означает, что этот файл не должен иметь входных почтовых ящиков. Как только указан список входных почтовых ящиков, Rmail-файл запоминает его и сохраняет до тех пор, пока он явно не будет изменен.

Как особое исключение, если ваш первичный Rmail-файл не указывает входных почтовых файлов, он использует стандартный системный.

Команда `g` (`rmail-get-new-mail`) вносит почту в текущий Rmail-файл из его входных файлов. Если у этого Rmail-файла нет входных файлов, `g` ничего не делает. Команда `M-x rmail` также вносит новую почту в ваш первичный Rmail-файл.

Чтобы перенести почту из файла, не являющегося обычным входным почтовым ящиком, задайте ключу `g` числовой аргумент, как в `C-u g`. Тогда он считает имя файла и перенесет почту из него. Когда `g` используется с аргументом, файл входного почтового ящика ни в коем случае не удаляется и не изменяется. Поэтому это основной способ переноса одного файла сообщений в другой.

## 27.7 Копирование сообщений в файлы

Эти команды копируют сообщения из Rmail-файла в другой файл.

`o` *файл* ⟨RET⟩
: Добавить копию текущего сообщения в конец *файла*, по умолчанию в формате Rmail-файлов (`rmail-output-to-rmail-file`).

`C-o` *файл* ⟨RET⟩
: Добавить копию текущего сообщения в конец *файла*, по умолчанию в формате системных почтовых ящиков (`rmail-output`).

`w` *файл* ⟨RET⟩
: Вывести только тело сообщения в *файл*, по умолчанию имя файла берется из поля ‘`Subject`’.

Команды `o` и `C-o` копируют текущее сообщение в указанный файл. Это может быть Rmail-файл или файл в формате системных почтовых ящиков; команды вывода выясняют формат этого файла и записывают копируемое сообщение в этом формате.

При копировании сообщения в файл в формате Unix, эти команды включают все поля заголовков, которые видимы в данный момент. Если хотите, используйте сначала команду `t`, чтобы указать, какие заголовки показывать (и копировать).

Команды `o` и `C-o` различаются с двух сторон: каждая предлагает свое имя файла по умолчанию и каждая задает свой выбор формата, когда файл еще не существует. Команда `o` использует при создании нового файла формат Rmail, тогда как `C-o` использует для нового файла системный формат. По умолчанию `o` использует имя файла, заданное последней `o`, а `C-o` — последней `C-o`.

Если к выходному Rmail-файлу в данный момент обращается какой-нибудь буфер, команды вывода копируют сообщение в этот буфер. Нужно ли действительно сохранять буфер в его файл решаете вы.

Иногда вы можете получить сообщение, чье тело несет содержимое файла. Вы можете сохранить его тело в файл (исключая заголовки сообщения) с помощью команды `w` (`rmail-output-body-to-file`). Часто эти сообщения содержат имя целевого файла в поле ‘`Subject`’, поэтому команда `w` использует это поле как имя выходного файла по умолчанию. Однако, имя файла считывается из минибуфера, поэтому при желании вы можете указать другое имя.

Вы также можете вывести сообщение в Rmail-файл, выбранный из меню. Сначала выберите пункт меню Classify, из меню Classify выберите пункт Output Rmail File; затем выберите нужный вам пункт меню. Это выведет текущее сообщение в указанный файл, как команда `o`. Переменные `rmail-secondary-file-directory` и `rmail-secondary-file-regexp` указывают, какие файлы предлагает это меню: первая переменная говорит в каком каталоге их искать; вторая говорит, какие файлы в этом каталоге предлагать (все, чьи имена соответствуют регулярному выражению).

Копирование сообщения придает его исходной копии атрибут ‘`filed`’, так что когда такое сообщение становится текущим, в строке режима появляется слово ‘`filed`’. Если вы хотите хранить только одну копию каждого почтового сообщения, установите переменную `rmail-delete-after-output` равной `t`; тогда команды `o` и `C-o` после копирования удаляют оригинал. (Впоследствии вы можете отменить удаление оригинального сообщения, если захотите.)

При копировании в файлы в системном формате почтовых ящиков используются поля заголовка, которые показаны в это время в Rmail. Таким образом, если вы используете команду `t` для просмотра всего заголовка и затем скопируете сообщение, скопируется весь заголовок. См. Раздел 27.13 [Отображение сообщений], с. 286.

Переменная `rmail-output-file-alist` позволяет вам указать интеллектуальные значения по умолчанию для выходного файла, основанные на содержимом текущего сообщения. Ее значением должен быть список, чьи элементы имеют такую форму:

```
(regexp . имя)
```

Если в текущем сообщении есть совпадение с *regexp*, то по умолчанию именем выходного файла будет *имя*. Если совпадения найдены для нескольких элементов, то имя файла по умолчанию определяется первым совпавшим элементом. Подвыражение *имя* может быть

константной строкой, дающей имя файла, или, в более общем случае, любым лисповским выражением, возвращающим имя файла в виде строки. `rmail-output-file-alist` относится как к `o`, так и к `C-o`.

## 27.8 Метки

У каждого сообщения могут быть различные *метки*, приписываемые ему в качестве средства классификации. Метка имеет имя; разные имена означают разные метки. Любая данная метка либо присутствует, либо отсутствует в конкретном сообщении. Ряд имен меток имеют стандартные значения и присваиваются сообщениям в Rmail автоматически в нужный момент; такие специальные метки называются *атрибутами*.

Все другие метки приписываются пользователем.

`a` *метка* ⟨RET⟩
: Приписать *метку* текущему сообщению (`rmail-add-label`).

`k` *метка* ⟨RET⟩
: Удалить *метку* из текущего сообщения (`rmail-kill-label`).

`C-M-n` *метки* ⟨RET⟩
: Передвинуться на следующее сообщение, которое имеет одну из *меток* (`rmail-next-labeled-message`).

`C-M-p` *метки* ⟨RET⟩
: Передвинуться на предыдущее сообщение, которое имеет одну из *меток* (`rmail-previous-labeled-message`).

`C-M-l` *метки* ⟨RET⟩
: Сделать резюме всех сообщений, содержащих какую-либо из *меток* (`rmail-summary-by-labels`).

Команды `a` (`rmail-add-label`) и `k` (`rmail-kill-label`) позволяют вам приписывать или удалять любую метку из текущего сообщения. Если аргумент *метка* пустой, то это означает приписывание или удаление той самой метки, которая была приписана или удалена самой последней.

Как только вы присвоили сообщениям метки, чтобы классифицировать их так, как вам хочется, появляются два способа использования этих меток: в перемещении и в резюме.

Команда `C-M-n` *метки* ⟨RET⟩ (`rmail-next-labeled-message`) передвигает к следующему сообщению, которое имеет одну из *меток*. Аргумент *метки* — это одно или несколько имен меток, разделенных запятыми. `C-M-p` (`rmail-previous-labeled-message`) — аналогичная команда, но передвигает назад к предыдущим сообщениям. Числовой аргумент в этих командах работает как счетчик повторов.

Команда `C-M-l` *метки* ⟨RET⟩ (`rmail-summary-by-labels`) показывает резюме, содержащее только сообщения, имеющие по крайней мере одну из описанного набора меток. Аргумент *метки* — это одно или более имен меток, разделенных запятыми. См. Раздел 27.11 [Резюме сообщений], с. 284, для получения информации о резюме.

Если аргумент *метки* для `C-M-n`, `C-M-p` или `C-M-l` пустой, то это предполагает использование последнего набора меток, указанного для какой-либо из этих команд.

## 27.9 Атрибуты в Rmail

Некоторые метки, такие, как ‘`deleted`’ и ‘`filed`’, имеют встроенные значения и приписываются сообщениям или удаляются из них автоматически в соответствующее время; эти метки называются *атрибутами*. Ниже приводится список атрибутов Rmail.

‘unseen’ Означает, что сообщение никогда не было текущим. Приписывается сообщениям, когда они поступают из файла входного почтового ящика, и удаляется, когда сообщение становится текущим. Когда вы запускаете Rmail, он изначально показывает первое сообщение с этим атрибутом.

‘deleted’ Означает, что сообщение удаляется. Приписывается командами удаления и уничтожается командами отмены удаления (см. Раздел 27.4 [Удаление сообщений], с. 277).

‘filed’ Означает, что сообщение было скопировано в какой-нибудь другой файл. Приписывается командами вывода файла (см. Раздел 27.6 [Rmail-файлы], с. 278).

‘answered’
Означает, что вы послали ответ на это сообщение. Приписывается командой `r` (`rmail-reply`). См. Раздел 27.10 [Посылка ответов], с. 282.

‘forwarded’
Означает, что вы перенаправляли сообщение другим. Приписывается командой `f` (`rmail-forward`). См. Раздел 27.10 [Посылка ответов], с. 282.

‘edited’ Означает, что вы редактировали текст сообщения внутри Rmail. См. Раздел 27.14 [Редактирование сообщений], с. 287.

‘resent’ Означает, что вы пересылали это сообщение. Приписывается командой `M-x rmail-resend`. См. Раздел 27.10 [Посылка ответов], с. 282.

Все другие метки приписываются или удаляются только пользователем и не имеют стандартного смысла.

## 27.10 Посылка ответов

Rmail имеет ряд команд, которые используют режим Mail для посылки исходящей корреспонденции. См. Глава 26 [Посылка почты], с. 267, для справок по использованию режима Mail, включая некоторые средства, предназначенные для работы с Rmail. А в этом разделе описываются специальные команды Rmail для входа в режим Mail. Следует отметить, что обычные команды для посылки почты, `C-x m` и `C-x 4 m`, применимы в режиме Rmail и работают точно так же, как обычно.

`m` Послать сообщение (`rmail-mail`).

`c` Продолжить редактирование уже начатого выходного сообщения (`rmail-continue`).

`r` Послать ответ на текущее сообщение Rmail (`rmail-reply`).

`f` Перенаправить текущее сообщение другим пользователям (`rmail-forward`).

`C-u f` Переслать сообщение другим пользователям (`rmail-resend`).

`M-m` Попробовать послать отскочившее письмо еще раз (`rmail-retry-failure`).

Самая распространенная причина посылки сообщения во время пребывания в Rmail — это необходимость ответа на сообщение, которое вы читаете. Чтобы сделать это, наберите `r` (`rmail-reply`). Это вызовет появление буфера ‘`*mail*`’ в другом окне, что очень похоже на `C-x 4 m`, но при этом заранее инициализируются поля заголовка ‘`Subject`’, ‘`To`’, ‘`CC`’ и ‘`In-reply-To`’, основываясь на сообщении, на которое формируется ответ. Полю ‘`To`’ присваивается имя отправителя этого сообщения, а в поле ‘`CC`’ помещаются имена всех остальных его получателей.

Вы можете предотвратить появление некоторых адресов в поле ‘`CC`’, используя переменную `rmail-dont-reply-to-names`. Ее значением должно быть регулярное выражение

(в виде строки): все получатели, чьи адреса совпадают с этим регулярным выражением, будут исключены из поля ‘CC’. По умолчанию она соответствует вашему собственному имени и любому имени, начинающемуся с ‘info-’. (Эти имена исключаются, потому что есть соглашение об использовании их для больших списков рассылки для широкой публикации анонсов.)

Чтобы полностью опустить поле ‘CC’ в каком-то ответе, введите команду ответа с числовым аргументом: `C-u r` или `1 r`.

После того, как буфер ‘*mail*’ проинициализирован, редактирование и отправка почты идет как обычно (см. Глава 26 [Посылка почты], с. 267). Если с вашей точки зрения ранее подготовленные поля заголовка неправильны, то их можно отредактировать. Вы также можете использовать команды режима Mail (см. Раздел 26.4 [Режим Mail], с. 270), включая `C-c C-y`, которая вставляет сообщение, на которое вы отвечаете. Вы можете вернуться в буфер Rmail, выбрать там другое сообщение, переключиться назад и вставить новое текущее сообщение.

Иногда сообщения не доходят по назначению. Почтовые программы обычно возвращают вам это письмо, заключив его в *сообщение об отказе*. Команда Rmail `M-m` (`rmail-retry-failure`) подготавливает этого же сообщение к повторной посылке: она создает буфер ‘*mail*’ с тем же телом и полями заголовка, какие были раньше. Если вы сразу нажмете `C-c C-c`, вы пошлете сообщение точно таким же, каким оно было в первый раз. Иначе, вы можете отредактировать его текст или заголовки и затем отправить. Переменная `rmail-retry-ignored-headers`, в том же формате, что и `rmail-ignored-headers` (см. Раздел 27.13 [Отображение сообщений], с. 286), контролирует, какие заголовки будут удалены из сообщения об отказе при попытке повторной отправки; по умолчанию она равна `nil`.

Еще одна достаточно частая причина отправки почты в Rmail заключается в необходимости *перенаправить* текущее сообщение другим пользователям. Команда `f` (`rmail-forward`) облегчает это, заранее инициализируя буфер ‘*mail*’ текстом текущего сообщения и указывая в поле ‘Subject’, что это перенаправленное письмо. От вас требуется только записать получателей и отправить. Когда вы перенаправляете сообщение, адресаты получают его “от вас”, а его содержимым является оригинальное сообщение.

Перенаправленное сообщение заключается между двумя разделительными строками. Кроме этого, все строки в нем, начинающиеся с дефисов, изменяются добавлением ‘- ’ в начало. Когда вы получаете перенаправленное сообщение, если оно содержит что-то кроме обычного текста — исходный код программы, например — вы можете счесть удобным проделать обратное преобразование. Вы можете сделать это, выбрав перенаправленное сообщение и напечатав `M-x unforward-rmail-message`. Эта команда извлекает оригинальное сообщение, удаляя вставленные строки ‘- ’, и вставляет его в Rmail-файл как отдельное сообщение сразу после текущего.

*Пересылка* — это вариант, похожий на перенаправление; разница в том, что при пересылке письмо отправляется “от начального посылателя”, таким же, каким оно пришло к вам, но с добавлением полей заголовка ‘Resent-from’ и ‘Resent-to’, чтобы обозначить, что оно исходит от вас. Чтобы переслать сообщение в Rmail, используйте `C-u f`. (`f` запускает `rmail-forward`, которая запрограммирована так, чтобы вызывать `rmail-resend`, если вы задали числовой аргумент.)

Команда `m` (`rmail-mail`) используется для начала редактирования исходящего сообщения, которое не является ответом. Она оставляет поля заголовка пустыми. Единственное отличие ее от `C-x 4 m` состоит в том, что она обеспечивает доступность буфера Rmail для `C-c C-y`, так же, как это делает `r`. Таким образом, `m` может быть использована для отправки ответа на сообщение или для перенаправления; она может делать все, что делают команды `r` и `f`.

Команда `c` (`rmail-continue`) возобновляет редактирование буфера '`*mail*`', чтобы вы могли завершить сообщение, которое уже составляли, а также для изменения сообщения, которое вы отправили.

Если вы установите переменную `rmail-mail-new-frame` в значение, не равное `nil`, то все команды Rmail для начала отправки сообщения будут создавать для его редактирования новый фрейм. Этот фрейм удаляется, когда вы отсылаете сообщение, или когда вы используете пункт '`Don't Send`' в меню '`Mail`'.

Все команды Rmail для отправки сообщения используют тот метод составления, который вы выбрали (см. Раздел 26.6 [Почтовые методы], с. 274).

## 27.11 Резюме

*Резюме* — это буфер, который содержит по одной строке на сообщение и дает вам возможность обзора почты в Rmail-файле. В каждой такой строке показан номер сообщения, его отправитель, метки и тема. Почти все команды Rmail можно использовать также и в буфере резюме; они относятся к сообщению, описанному на текущей строке. Перемещение точки в буфере резюме выбирает другие сообщения, когда вы переходите к их строкам.

Каждый буфер резюме относится только к одному Rmail-файлу; если редактируется множество Rmail-файлов, то они имеют отдельные буферы резюме. Имя буфера резюме создается добавлением '`-summary`' в конец имени буфера Rmail. Как правило, в один и тот же момент показывается только один буфер резюме.

### 27.11.1 Создание резюме

Ниже приводятся команды для создания резюме для текущего Rmail-файла. Если Rmail-файл имеет буфер резюме, изменения в этом файле (такие как удаление или вычеркивание сообщений и получение новой почты) автоматически обновляют резюме.

`h`
`C-M-h` Создает резюме всех сообщений (`rmail-summary`).

`l` *метки* ⟨RET⟩
`C-M-l` *метки* ⟨RET⟩
Создает резюме сообщений, которые имеют одну или несколько указанных меток (`rmail-summary-by-labels`).

`C-M-r` *получатели* ⟨RET⟩
Создает резюме сообщений, отправленных одному или нескольким из указанных получателей (`rmail-summary-by-recipients`).

`C-M-t` *тема* ⟨RET⟩
Создает резюме сообщений, которые имеют совпадение с регулярным выражением *тема* в полях заголовка '`Subject`' (`rmail-summary-by-topic`).

Команда `h` или `C-M-h` (`rmail-summary`) заполняет буфер резюме для текущего Rmail-файла перечнем всех сообщений из этого файла. Потом она показывает и выбирает этот буфер резюме в другом окне.

`C-M-l` *метки* ⟨RET⟩ (`rmail-summary-by-labels`) делает частичное резюме, упоминающее только сообщения, которые имеют одну или несколько *меток*. *Метки* должны содержать имена меток, разделенные запятыми.

`C-M-r` *получатели* ⟨RET⟩ (`rmail-summary-by-recipients`) делает частичное резюме, упоминающее только сообщения, которые имеют одного или нескольких *получателей*. Аргумент *получатели* должен содержать почтовые адреса, разделенные запятыми.

`C-M-t` *тема* ⟨RET⟩ (`rmail-summary-by-topic`) создает частичное резюме, упоминающее только сообщения, чьи поля '`Subject`' имеют совпадения с регулярным выражением *тема*.

Следует отметить, что для любого Rmail-файла существует только один буфер резюме; создание резюме одного вида отменяет любое ранее сделанное резюме.

Переменная `rmail-summary-window-size` говорит, сколько строк должно использовать окно резюме. Переменная `rmail-summary-line-count-flag` указывает, должны ли строки резюме содержать число, показывающее количество строк в сообщении.

### 27.11.2 Редактирование резюме

Вы можете делать в буфере резюме Rmail практически все, что вы можете делать в самом буфере Rmail. На самом деле, если у вас есть буфер резюме, нет необходимости переключаться назад в буфер Rmail.

Вы можете выбирать и просматривать различные сообщения в буфере Rmail из буфера резюме просто перемещая в нем точку к разным строкам. Не играет роли, какую именно команду Emacs вы используете для перемещения точки; на какой строке окажется точка после команды, то сообщение и будет выбрано в буфере Rmail.

Почти все команды Rmail работают и в буфере резюме, также как и в буфере Rmail. А именно, `d` в буфере резюме удаляет текущее сообщение, `u` отменяет удаление, а `x` вычеркивает. `o` и `C-o` выводят текущее сообщение в файл; `r` позволяет написать ответ. Вы можете прокручивать текущее сообщение, оставаясь в буфере резюме, с помощью ⟨SPC⟩ и ⟨DEL⟩.

Команды Rmail для передвижения по сообщениям работают также и в буфере резюме, но с особенностью: они перемещают по тем сообщениям, которые перечислены в резюме. Они также убеждаются, что буфер Rmail появляется на экране (в отличие от команд движения курсора, которые обновляют содержимое буфера Rmail, но не показывают его в окне, если он уже не виден). Вот перечень этих команд:

`n` Перейти к следующей строке, пропуская удаленные, и выбрать ее сообщение.

`p` Перейти к предыдущей строке, пропуская удаленные, и выбрать ее сообщение.

`M-n` Перейти к следующей строке и выбрать ее сообщение.

`M-p` Перейти к предыдущей строке и выбрать ее сообщение.

`>` Перейти к последней строке и выбрать ее сообщение.

`<` Перейти к первой строке и выбрать ее сообщение.

`M-s` *образец* ⟨RET⟩
: Производит поиск *образца* в сообщениях начиная с текущего; выбирает найденное сообщение и перемещает точку в буфере резюме к его строке.

Удаление, отмена удаления, получение новой почты и даже выбор другого сообщения — все обновляют буфер резюме, когда вы делаете эти операции в буфере Rmail. Если переменная `rmail-redisplay-summary` не равна `nil`, эти действия также возвращают буфер резюме на экран.

Когда вы завершили использование резюме, наберите `Q` (`rmail-summary-wipe`), чтобы удалить окно буфера резюме. Вы также можете выйти из Rmail, находясь в буфере резюме: `q` (`rmail-summary-quit`) удаляет окно резюме, а затем выходит из Rmail, записывая Rmail-файл и переключая в другой буфер.

## 27.12 Сортировка Rmail-файла

`M-x rmail-sort-by-date`
: Сортирует сообщения текущего Rmail-файла по дате.

`M-x rmail-sort-by-subject`
: Сортирует сообщения текущего Rmail-файла по теме.

`M-x rmail-sort-by-author`
: Сортирует сообщения текущего Rmail-файла по имени автора.

`M-x rmail-sort-by-recipient`
: Сортирует сообщения текущего Rmail-файла по именам получателей.

`M-x rmail-sort-by-correspondent`
: Сортирует сообщения текущего Rmail-файла по имени другого корреспондента.

`M-x rmail-sort-by-lines`
: Сортирует сообщения текущего Rmail-файла по размеру (числу строк).

`M-x rmail-sort-by-keywords` ⟨RET⟩ *метки* ⟨RET⟩
: Сортирует сообщения текущего Rmail-файла по меткам. Аргумент *метки* должен быть списком меток, разделенных запятыми. Порядок этих меток определяет порядок сообщений; сообщения с первой меткой идут первыми, сообщения со второй меткой идут вторыми, и так далее. Сообщения, не имеющие ни одну из этих меток, идут последними.

Команды Rmail для сортировки производят *устойчивую сортировку*: если нет причины предпочитать одно из двух сообщений, их порядок не изменяется. Вы можете использовать это для сортировки по нескольким критериям. Например, если вы применили `rmail-sort-by-date` и затем `rmail-sort-by-author`, сообщения от одного автора появятся в порядке возрастания даты.

С числовым аргументом, все эти команды изменяют порядок сравнения на противоположный. Это означает, что они сортируют сообщения от новых к старым, от больших в меньшим или в обратном алфавитном порядке.

## 27.13 Отображение сообщений

Rmail переформатирует заголовок каждого сообщения перед его показом. При этом удаляются неинтересные поля заголовков, чтобы уменьшить беспорядок. Чтобы посмотреть заголовок целиком или повторить операцию переформатирования, вы можете использовать команду `t` (`rmail-toggle-headers`).

`t`
: Включает или выключает показ полного заголовка (`rmail-toggle-header`).

При переформатировании удаляются большинство полей заголовка, поскольку они не представляют интереса. Переменная `rmail-ignored-headers` содержит регулярное выражение, указывающее какие поля надо скрывать таким способом — если оно соответствует началу поля, то это поле скрывается.

Rmail сохраняет полные исходные заголовки перед переформатированием; чтобы посмотреть их, используйте команду `t` (`rmail-toggle-headers`). Она сбрасывает переформатированные заголовки текущего сообщения и показывает его с исходными заголовками. Повтор команды `t` снова переформатирует сообщение. Повторный выбор сообщения тоже приводит к переформатированию.

Одно из последствий этого состоит в том, что если вы редактировали переформатированный заголовок (с помощью `e`; см. Раздел 27.14 [Редактирование сообщений], с. 287), последующее использование `t` сбросит ваши изменения. С другой стороны, если вы использовали `e` после `t`, чтобы редактировать первоначальный (не переформатированный) заголовок, эти изменения останутся.

Когда команде `t` дан префиксный аргумент, положительный аргумент обозначает показ переформатированного заголовка, а нулевой или отрицательный — полного.

При использовании в оконной системе, которая поддерживает несколько шрифтов, Rmail подсвечивает некоторые поля заголовка, которые особенно интересны — по умолчанию это

поля ‘From’ и ‘Subject’. Переменная rmail-highlighted-headers хранит регулярное выражение, задающее поля заголовка, которые нужно подсвечивать; если оно соответствует началу поля, все это поле подсвечивается.

Если вы зададите необычные цвета для текста и фона, то цвета, используемые для подсветки, могут с ними плохо смотреться. В таком случае задайте другие цвета для начертания highlight. Это стоит сделать, потому что начертание highlight используется и других видах выделения. См. Раздел 17.13 [Начертания], с. 155, чтобы узнать, как это сделать.

Чтобы полностью выключить подсветку в Rmail, установите rmail-highlighted-headers равной nil.

## 27.14 Редактирование внутри сообщения

Большинство обычных команд Emacs доступны в режиме Rmail, хотя некоторые, такие как C-M-n и C-M-h, переопределяются в Rmail для других целей. Однако обычно буфер Rmail предназначен только для чтения, и большинство букв переопределены как команды Rmail. Если вы хотите отредактировать текст сообщения, вы должны использовать команду Rmail e.

e Редактировать текущее сообщение как обычный текст.

Команда e (rmail-edit-current-message) переключает из режима Rmail в режим Rmail Edit, другой основной режим, практически эквивалентный режиму Text. Это изменение отображается в строке режима.

В режиме Rmail Edit буквы вставляют себя как обычно, а команды Rmail недоступны. Когда вы закончите редактирование сообщения и будете готовы вернуться в Rmail, наберите команду C-c C-c, которая переключит вас назад в режим Rmail. С другой стороны, можно вернуться в режим Rmail, но отменить все сделанные изменения, набрав C-c C-].

Вход в режим Rmail Edit вызывает ловушку text-mode-hook, а затем ловушку rmail-edit-mode-hook (см. Раздел 31.2.3 [Ловушки], с. 349). При этом в сообщению присваивается атрибут ‘edited’. Она также показывает полный заголовок сообщения, так что вы можете редактировать как тело, так и заголовки сообщения, и ваши изменения в заголовках останутся навсегда.

## 27.15 Сообщения-дайджесты

*Сообщение-дайджест* — это сообщение, которое создано для передачи нескольких других сообщений. Дайджесты используются в некоторых списках рассылки с ведущим; все сообщения, которые пришли для рассылки в течение какого-то периода времени, например одного дня, помещаются в единый дайджест, который затем рассылается подписчикам. Пересылка одного такого дайджеста занимает гораздо меньше машинного времени, чем пересылка отдельных сообщений, хоть и при одинаковом общем объеме, поскольку затраты ресурсов на одно сообщение при передаче почты по сети довольно значительны.

При получении сообщения-дайджеста наиболее удобным способом для его прочтения является *разброшюровка*: обратное разделение его на несколько отдельных сообщений. После этого можно читать и удалять отдельные сообщения как вам угодно.

Чтобы сделать это, необходимо выбрать сообщение-дайджест и затем набрать команду M-x undigestify-rmail-message. Она извлекает каждое подсообщение как отдельное сообщение Rmail и вставляет их вслед за дайджестом. Само сообщение-дайджест помечается как удаленное.

## 27.16 Преобразование Rmail-файла в системный формат

Команда `M-x unrmail` преобразует файл в формате Rmail в формат входных почтовых ящиков (также известный как системный почтовый формат), чтобы вы могли использовать его с другими программами для редактирования почты. Вы должны указать два аргумента, имя Rmail-файла и имя для преобразованного файла. `M-x unrmail` не изменяет сам Rmail-файл.

## 27.17 Чтение сообщений в Rot13

Сообщения из списков рассылки, которые могут оскорбить некоторых читателей, иногда кодируются простым методом, называемым *rot13* — потому что он сдвигает алфавит на 13 букв. Этот код служит не для секретности, он не предоставляет ее; скорее, он позволяет тем, кто может быть оскорблен, избежать возможности даже увидеть настоящий текст сообщения.

Чтобы просмотреть буфер, использующий код rot13, запустите команду `M-x rot13-other-window`. Она показывает текущий буфер в другом окне, применяя код при отображении текста.

## 27.18 `movemail` и POP

При получении новой почты Rmail сначала копирует ее из входного файла в Rmail-файл; затем он записывает Rmail-файл; затем усекает входной файл. Таким образом, сбой в системе может вызвать дублирование почты во входном файле и в Rmail-файле, но не может вызвать потери. Если `rmail-preserve-inbox` не равна `nil`, то Rmail будет копировать новую почту из входного файла в Rmail-файл, не усекая входной файл. Вы можете сделать такую установку, например, на портативном компьютере, который вы используете для проверки почты через POP во время путешествий, чтобы ваша почта оставалась на сервере, и вы могли сохранить ее позднее на своей рабочей станции.

В некоторых случаях Rmail копирует новую почту из входного файла не прямым путем. Сначала он запускает программу `movemail`, чтобы переместить почту из входного файла в промежуточный файл, называемый ‘`~/.newmail-`*входной-файл*’. Затем Rmail вносит новую почту из этого файла, сохраняет Rmail-файл и только затем удаляет промежуточный файл. Если в неподходящий момент случится фатальный сбой, этот файл останется на месте, и Rmail будет использовать его в следующий раз при получении новой почты из этого входного файла.

Если Rmail по какой-то причине не может преобразовать данные из файла ‘`~/.newmail-`*входной-файл*’ в формат Babyl, он переименовывает его в ‘`~/RMAILOSE.`*n*’ (*n* — это целое число, выбранное так, чтобы оно было уникальным), так что Rmail не будет иметь проблем с этими данными снова. Вам стоит просмотреть этот файл, найти сообщение, которое сбивает с толку Rmail (вероятно, оно содержит знак control-подчерк, восьмеричный код 037), и удалить его. Тогда вы можете использовать `1 g`, чтобы получить почту из исправленного файла.

Некоторые системы используют для доступа к данным почтовых ящиков пользователей метод, называемый POP, вместо записи этих данных в почтовые файлы. `movemail` может работать с POP, если вы скомпилировали ее с определенным макросом `MAIL_USE_POP`. (Вы можете получить это, задав ключ ‘`-with-pop`’, когда запускаете `configure` во время установки Emacs.) `movemail` работает только с POP3, но не с более ранними версиями POP.

В предположении, что вы правильно скомпилировали и установили `movemail`, вы можете указать входной почтовый ящик POP, написав “имя файла” в форме ‘`po:`*имя-пользователя*’ в списке входных почтовых ящиков Rmail-файла. `movemail` обрабатывает

такие имена путем установки соединения с POP-сервером. Переменная среды `MAILHOST` задает машину, на которой будет производиться поиск сервера.

Для доступа к почте через POP может потребоваться пароль. Если переменная `rmail-pop-password` отлична от `nil`, то она задает пароль для использования с POP. Иначе, если `rmail-pop-password-required` не равна `nil`, то Rmail спрашивает пароль у вас.

Если вам нужно передать `movemail` дополнительные флаги командной строки, установите переменную `rmail-movemail-flags` равной списку желаемых флагов. Не используйте эту переменную для передачи флага '`-p`', который предотвращает затирание содержимого входного файла; пользуйтесь вместо этого переменной `rmail-preserve-inbox`.

Установленная в вашей системе программа `movemail` может поддерживать аутенфикацию Kerberos. Если она поддерживается, то это будет использоваться по умолчанию всегда, когда вы будете пытаться получить почту через POP при неустановленных `rmail-pop-password` и `rmail-pop-password-required`.

Некоторые POP-серверы хранят сообщения в обратном порядке. Если ваш сервер делает так, а вам хотелось бы читать почту в том порядке, в каком она поступала, вы можете велеть `movemail` перевернуть очередность загружаемых сообщений, добавив к `rmail-movemail-flags` флаг '`-r`'.

# 28 Dired, редактор каталогов

Dired создает буфер Emacs, содержащий распечатку каталога и, возможно, некоторых его подкаталогов. Вы можете использовать обычные команды Emacs, чтобы передвигаться по этому буферу, и специальные команды Dired, чтобы производить различные действия над файлами.

## 28.1 Вход в Dired

Чтобы запустить Dired, выполните `C-x d` или `M-x dired`. Эта команда считывает имя каталога или шаблон имени файла как аргумент минибуфера, чтобы определить, какие файлы нужно перечислить. Команда `dired` отличается от `list-directory` тем, что она переводит буфер в режим Dired, так что в нем становятся доступными специальные команды Dired.

Переменная `dired-listing-switches` задает ключи для передачи `ls` при создании распечатки каталога; эта строка *должна* содержать '`-l`'. Если вы используете с командой `dired` числовой аргумент, вы можете указать в минибуфере ключи для `ls` до того, как введете имя каталога.

Чтобы показать буфер Dired в другом окне, а не в выбранном, вместо `C-x d` используется `C-x 4 d` (`dired-other-window`). `C-x 5 d` (`dired-other-frame`) использует для показа буфера Dired отдельный фрейм.

## 28.2 Команды, действующие в буфере Dired

Буфер Dired помечен как "предназначенный только для чтения", и вставлять в него текст бесполезно, так что обычные печатные знаки, такие как `d` и `x`, используются для команд Dired. Одни команды Dired *устанавливают флаг* на *текущем файле* (это файл на текущей строке) или *помечают* его; другие команды выполняют действия над помеченными файлами или файлами с установленным флагом.

Все обычные команды движения курсора в Emacs доступны и в буферах Dired. Также предусмотрены некоторые специальные команды. Ключи `C-n` и `C-p` переопределены так, что они устанавливают курсор в начало имени файла на строке, а не в начало самой строки.

Для большего удобства, ⟨SPC⟩ и `n` в Dired эквивалентны `C-n`. `p` эквивалентен `C-p`. (Движение по строкам делается в Dired настолько часто, что оно заслуживает того, чтобы набор его был облегчен.) ⟨DEL⟩ (передвинуться вверх и убрать флаг) часто бывает полезным просто для движения вверх.

## 28.3 Удаление файлов с помощью Dired

Прежде всего Dired используется, чтобы *установить флаг* для удаления на некоторых файлах, а потом удалить файлы, ранее помеченные флагом.

`d` Установить флаг удаления для этого файла.

`u` Убрать флаг удаления на этой строке.

⟨DEL⟩ Передвинуть точку на строку выше и убрать на этой строке флаг удаления.

`x` Удалить файлы, помеченные флагом удаления.

Вы можете установить на файле флаг, переместившись на строку, описывающую файл, и набрав на ней `d` (`dired-flag-file-deletion`). Флаг удаления выглядит как '`D`' в начале строки. Эта команда передвигает точку в начало следующей строки, таким образом,

повторение команды `d` помечает для удаления последующие файлы. Числовой аргумент служит в качестве счетчика повторов.

Файлы помечаются для удаления, а не удаляются немедленно, чтобы уменьшить опасность случайного удаления файла. До тех пор, пока вы прямо не укажете Dired удалить помеченный файл, вы можете убрать флаги, используя команды `u` или ⟨DEL⟩. `u` (`dired-unmark` работает точно так же, как `d`, но удаляет флаги, а не создает их. ⟨DEL⟩ (`dired-unmark-backward`) двигается вверх, убирая флаги; это подобно `u` с аргументом −1.

Чтобы удалить файлы с установленным флагом, наберите `x` (`dired-expunge`). Эта команда покажет сначала список всех имен файлов, помеченных для удаления, и потребует подтверждения вводом `yes`. Если вы подтверждаете, то все помеченные флагом файлы уничтожаются и их строки удаляются из текста буфера Dired. Сокращенный буфер Dired остается выбранным.

Если при запросе подтверждения вы ответите `no` или выйдите с помощью `C-g`, вы немедленно вернетесь в Dired; все флаги удаления останутся в буфере, и файлы не будут удалены.

## 28.4 Установка флага на несколько файлов одновременно

`#` Помечает флагом удаления все файлы, которые появились при самосохранении (файлы, чьи имена начинаются и кончаются на ‘`#`’) (см. Раздел 14.5 [Самосохранение], с. 114).

`~` Помечает флагом удаления все резервные файлы (файлы, чьи имена кончаются на ‘`~`’) (см. Раздел 14.3.1 [Резервные файлы], с. 110).

`&` Помечает флагом удаления все файлы с определенными типами имен, предполагающими, что вы легко сможете их снова создать.

`.` (Точка) Помечает флагом удаления излишние резервные файлы. Сохраняются только несколько самых старых и самых новых резервных копий; промежуточные помечаются флагом.

`% d` *регулярное-выражение* ⟨RET⟩
Помечает флагом удаления все файлы, чьи имена соответствуют заданному *регулярному-выражению*.

Команды `#`, `~`, `&` и `.` устанавливают флаг для нескольких файлов, основываясь на их именах. Эти команды полезны именно потому, что сами по себе они не удаляют файлы; вы можете убрать флаги удаления с любых помеченных файлов, которые вы в действительности хотите сохранить.

`&` (`dired-flag-garbage-files`) устанавливает флаг удаления для файлов, чьи имена соответствуют регулярному выражению, заданному переменной `dired-garbage-files-regexp`. По умолчанию ей соответствуют определенные файлы, производимые TEX, и файлы ‘`.orig`’ и ‘`.rej`’, производимые программой `patch`.

`#` (`dired-flag-auto-save-files`) устанавливает флаг удаления для всех файлов, чьи имена выглядят как имена самосохраненных файлов (см. Раздел 14.5 [Самосохранение], с. 114) — это файлы с именами, начинающимися и заканчивающимися на ‘`#`’. `~` (`dired-flag-backup-files`) устанавливает флаг удаления для всех файлов, чьи имена говорят, что это резервные копии (см. Раздел 14.3.1 [Резервные файлы], с. 110) — это файлы с именами, заканчивающимися на ‘`~`’.

`.` (точка, `dired-clean-directory`) устанавливает флаг удаления лишь для некоторых резервных копий: для всех, кроме самых старых и самых новых резервных копий одного файла. Обычно `dired-kept-versions` (**не** `kept-new-versions`; эта переменная применяется при сохранении) задает число самых новых сохраняемых версий каждого файла, а `kept-old-versions` задает число самых старых сохраняемых версий.

Точка с положительным числовым аргументом, как в `C-u 3 .`, указывает число оставляемых новых версий, перекрывая значение `dired-kept-versions`. Отрицательный числовой аргумент перекрывает `kept-old-versions`, используя число, противоположное заданному, для указания числа оставляемых старых версий каждого файла.

Команда `% d` устанавливает флаг удаления для всех файлов, чьи имена соответствуют заданному регулярному выражению (`dired-flag-files-regexp`). При поиске используется имя файла с исключенной частью, определяющей каталог. Для привязки регулярного выражения к началу или концу имени вы можете использовать '`^`' и '`$`'. Подкаталоги вы можете исключить, скрыв их (см. Раздел 28.13 [Скрывание подкаталогов], с. 299).

## 28.5 Обращение к файлам в Dired

Есть несколько команд Dired для обращения к файлам, перечисленным в буфере Dired, или просмотра их содержимого. Все они применяются к файлу на текущей строке; если этот файл на самом деле является каталогом, эти команды вызывают в нем Dired (создавая отдельный буфер Dired).

`f` Обращается к файлу, описанному на текущей строке; подобна вводу `C-x C-f` и заданию имени этого файла (`dired-find-file`). См. Раздел 14.2 [Обращение], с. 106.

⟨RET⟩ Эквивалент `f`.

`o` Как `f`, но использует для отображения буфера этого файла отдельное окно (`dired-find-file-other-window`). Буфер Dired остается видимым в первом окне. Это подобно использованию `C-x 4 C-f` для обращения к этому файлу. См. Глава 16 [Окна], с. 141.

`C-o` Обращается к файлу, описанному на текущей строке, и отображает его буфер в отдельном окне, но не выбирает это окно (`dired-display-file`).

`Mouse-2` Обращается к файлу в строке, на которой вы щелкнули (`dired-mouse-find-file-other-window`). Эта команда использует для показа файла другое окно, как команда `o`.

`v` Обращается к файлу, описанному на текущей строке, в режиме просмотра, как `M-x view-file` (`dired-view-file`).

Просмотр файла похож на обращение к нему, но этот режим делает акцент на предоставление большего удобства для перемещения по файлу и не позволяет изменять его. См. Раздел 14.10 [Просмотр файла], с. 132.

## 28.6 Пометки Dired vs. флаги

Вместо установки для файла флага с помощью '`D`' вы можете установить на этом файле *метку* с помощью какого-либо другого знака (обычно '`*`'). Большинство команд Dired для работы с файлами, кроме "вычеркивания" (`x`), действуют на файлы, помеченные звездочкой '`*`'.

Вот некоторые команды для пометки с помощью '`*`', для снятия метки или для произведения каких-либо действий над метками. (См. Раздел 28.3 [Удаление в Dired], с. 291, для получения информации о командах для установки и снятия флагов на файлах.)

`m`
`* m` Помечает текущий файл звездочкой '`*`' (`dired-mark`). С числовым аргументом $n$, помечает следующие $n$ файлов начиная от текущего. (Если $n$ отрицателен, помечает $-n$ предыдущих файлов.)

`* *` Помечает все исполняемые файлы звездочкой ‘`*`’ (`dired-mark-executables`). С числовым аргументом, убирает метки со всех таких файлов.

`* @` Помечает все символьные ссылки звездочкой (`dired-mark-symlinks`). С числовым аргументом, убирает метки со всех таких файлов.

`* /` Помечает звездочкой все файлы, которые являются в действительности каталогами, исключая ‘`.`’ и ‘`..`’ (`dired-mark-directories`). С числовым аргументом, убирает метки со всех этих файлов.

`* s` Помечает все файлы в текущем подкаталоге, кроме ‘`.`’ и ‘`..`’ (`dired-mark-subdir-files`).

`u`
`* u` Убирает любую метку на этой строке (`dired-unmark`).

⟨DEL⟩
`*` ⟨DEL⟩ Перемещает точку на предыдущую строку и убирает любую метку на этой строке (`dired-unmark-backward`).

`* !` Убирает все метки со всех файлов в этом буфере Dired (`dired-unmark-all-files-no-query`).

`* ?` *знак-метки*
Убирает все метки, использующие знак *знак-метки* (`dired-unmark-all-files`). Аргументом должен быть один знак — не используете для его завершения ⟨RET⟩.

С числовым аргументом, эта команда запрашивает подтверждение на снятие метки для каждого помеченного файла. Вы можете ответить `y` для подтверждения, `n` для отказа или `!` для снятия меток со всех остальных файлов без запроса о них.

`* C-n` Перемещает вниз к следующему помеченному файлу (`dired-next-marked-file`). Файл считается “помеченным”, если на нем есть метки любого вида.

`* C-p` Перемещает вверх к предыдущему помеченному файлу (`dired-prev-marked-file`)

`* t` Переключает все метки (`dired-do-toggle`): файлы, помеченные ‘`*`’ становятся непомеченными, а непомеченные файлы метятся знаком ‘`*`’. Файлы, помеченные другим способом, не затрагиваются.

`* c` *старый новый*
Заменяет все метки, использующие знак *старый* на метки со знаком *новый* (`dired-change-marks`). Эта команда дает основной метод создания или использования меток, отличных от ‘`*`’ или ‘`D`’. Аргументами должны быть одиночные знаки — не используйте ⟨RET⟩ для их завершения.

С помощью этой команды вы можете использовать почти любой знак в качестве знака метки для разделения различных классов файлов. Если *старый* знак — это пробел (‘ ’), то эта команда действует на все непомеченные файлы; если *новый* знак является пробелом, эта команда убирает метку с файлов, на которые она действует.

Чтобы продемонстрировать мощь этой команды, мы покажем способ установить метку ‘`D`’ на все файлы, которые не были помечены, в то же время снимая метку со всех файлов, имевших метку ‘`D`’:

```
* c D t  * c SPC D  * c t SPC
```

Здесь предполагается, что ни один из файлов не помечен флагом ‘`t`’.

`% m` *regexp* ⟨RET⟩
`* %` *regexp* ⟨RET⟩
: Помечает (знаком '`*`') все файлы, чьи имена соответствуют регулярному выражению *regexp* (`dired-mark-files-regexp`). Эта команда похожа на `% d`, но она помечает файлы звездочкой '`*`', а не устанавливает флаг '`D`'. См. Раздел 28.4 [Установка флага на несколько файлов], с. 292.

  Для поиска совпадений используется только та часть имени файла, которая не задает каталог. Для привязки регулярного выражения к началу или концу имени вы можете использовать '`^`' и '`$`'. Подкаталоги вы можете исключить, скрыв их (см. Раздел 28.13 [Скрывание подкаталогов], с. 299).

`% g` *regexp* ⟨RET⟩
: Помечает (знаком '`*`') все файлы, чье *содержимое* включает совпадения с регулярным выражением *regexp* (`dired-mark-files-containing-regexp`). Эта команда похожа на `% m`, но она просматривает содержимое файлов, а не их имена.

`C-_`
: Отменяет изменения в буфере Dired, такие как добавление или снятие меток (`dired-undo`).

## 28.7 Действия над файлами

Этот раздел описывает основные команды Dired для произведения различных действий над несколькими файлами. Все эти команды являются заглавными буквами; все они используют минибуфер, либо для считывания аргументов, либо для запрашивания подтверждения перед совершением действий. Все они дают вам некоторые способы указания, какие файлы должны быть обработаны:

- Если вы даете команде числовой аргумент $n$, она действует на $n$ следующих файлов, начиная с текущего. (Если $n$ отрицательно, то эта команда действует на $-n$ файлов, предшествующих текущей строке.)
- Иначе, если какие-то файлы помечены с помощью '`*`', команда действует на все эти файлы.
- Иначе эта команда действует только на текущий файл.

Вот команды для манипуляций над файлами, действующие таким способом. (Некоторые другие команды Dired, такие как `!` и '`%`', также придерживаются этих соглашений для принятия решения о выборе файлов для работы.)

`C` *новый* ⟨RET⟩
: Копирует указанные файлы (`dired-do-copy`). Аргумент *новый* — это каталог, в который нужно копировать, или (при копировании единственного файла) новое имя.

  Если `dired-copy-preserve-time` не равна `nil`, то при копировании с помощью этой команды время изменения нового файла устанавливается таким же, как у старого файла.

`D`
: Удаляет указанные файлы (`dired-do-delete`). Подобно остальным командам в этом разделе, эта команда действует на *помеченные* файлы или на $n$ следующих файлов. Напротив, `x` (`dired-expunge`) удаляет все файлы с *установленным флагом*.

`R` *новый* ⟨RET⟩
: Переименовывает указанные файлы (`dired-do-rename`). Аргумент *новый* — это каталог, в который нужно переименовывать, или (при переименовании единственного файла) новое имя.

Dired автоматически изменяет имена файлов, к которым вы обращаетесь, для связанных с этими переименованным файлами буферов так, чтобы они отражали новые имена.

`H` *новый* ⟨RET⟩
: Создает жесткие ссылки на указанные файлы (`dired-do-hardlink`). Аргумент *новый* — это каталог, в котором нужно создавать ссылки, или (при создании ссылки на единственный файл) имя этой ссылки.

`S` *новый* ⟨RET⟩
: Создает символьные ссылки на указанные файлы (`dired-do-symlink`). Аргумент *новый* — это каталог, в котором нужно создавать ссылки, или (при создании ссылки на единственный файл) имя этой ссылки.

`M` *режим* ⟨RET⟩
: Изменяет режим (также называемый “битами прав доступа”) указанных файлов (`dired-do-chmod`). Эта команда использует программу `chmod`, потому *режим* может быть любым аргументом, который `chmod` способен обработать.

`G` *новая-группа* ⟨RET⟩
: Заменяет группу владельцев указанных файлов на *новую-группу* (`dired-do-chgrp`).

`O` *новый-владелец* ⟨RET⟩
: Заменяет владельца указанных файлов на *нового-владельца* (`dired-do-chown`). (На большинстве систем это может делать только привилегированный пользователь.)

  Переменная `dired-chown-program` задает имя программы, используемой для этих задач (различные системы помещают `chown` в разные места).

`P` *команда* ⟨RET⟩
: Печатает указанные файлы (`dired-do-print`). Вы должны указать команду печати, но в минибуфере сразу появляется подходящая предполагаемая строка, полученная с помощью переменных `lpr-command` и `lpr-switches` (эти же переменные использует `lpr-buffer` ; см. Раздел 30.4 [Распечатка], с. 331).

`Z`
: Сжимает указанные файлы (`dired-do-compress`). Если оказывается, что какой-то файл уже сжат, эта команда наоборот раскрывает его.

`L`
: Загружает указанные файлы Emacs Lisp (`dired-do-load`). См. Раздел 23.7 [Библиотеки Лиспа], с. 253.

`B`
: Байт-компилирует указанные файлы на Emacs Lisp (`dired-do-byte-compile`). См. раздел “Byte Compilation” в *The Emacs Lisp Reference Manual*.

`A` *regexp* ⟨RET⟩
: Производит поиск регулярного выражения *regexp* во всех указанных файлах (`dired-do-search`).

  Эта команда — вариант команды `tags-search`. Поиск останавливается при первом найденном совпадении; чтобы продолжить поиск и найти следующее совпадение, нажмите `M-,`. См. Раздел 22.13.5 [Поиск с помощью тегов], с. 229.

`Q` *старое* ⟨RET⟩ *новое* ⟨RET⟩
: Производит `query-replace-regexp` в каждом из указанных файлов, заменяя совпадения *старого* (регулярного выражения) на строку *новое* (`dired-do-query-replace`).

  Эта команда — вариант `tags-query-replace`. Если вы выйдите из цикла замены с подтверждением, вы можете использовать `M-,` для продолжения поиска и замены дальнейших совпадений. См. Раздел 22.13.5 [Поиск с помощью тегов], с. 229.

Одна особая команда для работы с файлами — это `+` (`dired-create-directory`). Она считывает имя каталога и создает его, если каталог с таким именем еще не существует.

## 28.8 Команды оболочки в Dired

Команда Dired `!` (`dired-do-shell-command`) считывает в минибуфере командную строку оболочки и запускает эту команду оболочки для всех указанных файлов. Вы можете задать обрабатываемые файлы обычными методами, как для команд Dired (см. Раздел 28.7 [Действия над файлами], с. 295). Есть два способа применить команду оболочки к нескольким файлам:

- Если вы используете в команде оболочки '`*`', то эта команда запускается один раз, а '`*`' заменяется списком имен файлов. Имена файлов передаются в том же порядке, в каком они появляются в буфере Dired.

  Таким образом, `! tar cf foo.tar *` ⟨RET⟩ запускает `tar` для всего списка имен файлов, помещая их все в один tar-файл '`foo.tar`'.
- Если командная строка не содержит '`*`', она запускается один раз *для каждого файла* с добавленным в конце именем этого файла.

  Например, `! uudecode` ⟨RET⟩ запускает для каждого файла команду `uudecode`.

Что если вы хотите выполнить команду оболочки один раз для каждого файла, но с именем файла, вставленным в середине? Или если вы хотите использовать имена файлов более сложным образом? Используйте циклы оболочки. Например, такая команда оболочки запустила бы `uuencode` для каждого заданного файла, записывая вывод в соответствующий '`.uu`'-файл:

```
for file in *; do uuencode $file $file >$file.uu; done
```

Рабочим каталогом команды оболочки служит каталог верхнего уровня буфера Dired.

Команда `!` не пытается обновить буфер Dired, чтобы показать новые или измененные файлы, потому что на самом деле она не понимает команд оболочки и не знает, какие имена изменились. Для обновления буфера Dired используйте команду `g` (см. Раздел 28.14 [Обновление в Dired], с. 300).

## 28.9 Преобразование имен файлов в Dired

Вот команды, которые изменяют имена файлов систематическим образом:

`% u` Изменяет имя каждого из выбранных файлов на имя, написанное заглавными буквами (`dired-upcase`). Если старыми именами файлов были '`Foo`' и '`bar`', то новыми будут '`FOO`' и '`BAR`'.

`% l` Изменяет имя каждого из выбранных файлов на имя, написанное строчными буквами (`dired-downcase`). Если старыми именами файлов были '`Foo`' и '`bar`', то новыми будут '`foo`' и '`bar`'.

`% R` *старое* ⟨RET⟩ *новое* ⟨RET⟩
`% C` *старое* ⟨RET⟩ *новое* ⟨RET⟩
`% H` *старое* ⟨RET⟩ *новое* ⟨RET⟩
`% S` *старое* ⟨RET⟩ *новое* ⟨RET⟩
Эти четыре команды переименовывают, копируют, создают жесткие и символьные ссылки, вычисляя в каждом случае новое имя путем подстановки в регулярном выражении, задающем имя старого файла.

Эти четыре команды подстановки в регулярном выражении в действительности производят поиск и замену в именах выбранных файлов в буфере Dired. Они принимают два аргумента: регулярное выражение *старое* и образец подстановки *новое*.

Эти команды сравнивают каждое “старое” имя файла с регулярным выражением *старое* и затем заменяют совпавшую часть на *новое*. Вы можете использовать в строке *новое* обозначения ‘\&’ и ‘*цифра*’, чтобы сослаться на весь или на часть совпавшего образца в старом имени файла, как в `replace-regexp` (см. Раздел 12.7.2 [Замена регулярного выражения], с. 96). Если в имени файла есть более одного совпадения с регулярным выражением, заменяется только первое.

Например, `% R ^.*$ ⟨RET⟩ x-\& ⟨RET⟩` переименовывает каждый выбранный файл, добавляя ‘x-’ в начало имени. Обратная процедура, удаление ‘x-’ из начала каждого имени файла, также возможна: один способ — набрать `% R ^x-\(.*\)$ ⟨RET⟩ \1 ⟨RET⟩`; другой — это `% R ^x- ⟨RET⟩ ⟨RET⟩`. (Используйте ‘^’ и ‘$’ для привязки регулярных выражений к началу или концу имени.)

Обычно при замене не затрагиваются имена каталогов, которым принадлежат файлы; обрабатываются только файлы в этом каталоге. Если вы зададите числовой аргумент, равный нулю, при замене будут обрабатываться полные абсолютные имена файлов, включая имена каталогов.

Часто вы можете захотеть выбрать набор обрабатываемых файлов с помощью того же регулярного выражения *regexp*, что будет использоваться для их обработки. Чтобы сделать так, пометьте эти файлы командой `% m` *regexp* `⟨RET⟩`, а затем примените это же регулярное выражение в команде обработки. Для облегчения этого, команды обработки файлов, начинающиеся на `%`, используют по умолчанию последнее регулярное выражение, заданное любой команде `%`.

## 28.10 Сравнение файлов в Dired

В Dired есть две команды, которые сравнивают заданные файлы с помощью программы `diff`.

`=` Сравнивает текущий файл (файл в позиции точки) с другим файлом (файлом в позиции метки), используя программу `diff` (`dired-diff`). Файл в позиции метки — это первый аргумент `diff`, а файл в позиции точки — второй.

`M-=` Сравнивает текущий файл с его последней резервной копией (`dired-backup-diff`). Если текущий файл сам является резервной копией, сравнивает его с оригиналом; таким образом вы можете сравнить файл с любой его резервной версией по вашему выбору.

Первым аргументом `diff` передается резервная копия.

## 28.11 Подкаталоги в Dired

В обычном случае буфер Dired показывает только один каталог; но вы также можете включить в список и его подкаталоги.

Простейший способ включить несколько подкаталогов в один буфер Dired — задать для запуска `ls` ключи ‘`-lR`’. (Если при вызове Dired вы зададите числовой аргумент, вы сможете написать эти ключи в минибуфере.) Это произведет рекурсивный список каталога, показывающий все подкаталоги всех уровней.

Но обычно всех подкаталогов бывает слишком много; чаще вы предпочли бы включить только конкретные подкаталоги. Вы можете сделать это с помощью команды `i`:

`i` Вставляет содержимое подкаталога ниже в этом буфере.

Применяйте команду `i` (`dired-maybe-insert-subdir`) на строке, описывающей файл, который является каталогом. Она вставляет содержимое этого каталога в этот же буфер

Dired и перемещает к нему. Вставленное содержимое подкаталога следует после каталога верхнего уровня данного буфера Dired, как в выводе ‘`ls -lR`’.

Если содержимое подкаталога уже находится в этом буфере, команда `i` просто перемещает к нему.

В обоих случаях `i` до перемещения устанавливает метку Emacs, так что `C-u C-`⟨SPC⟩ возвращает вас к предыдущей позиции в буфере (к строке, описывающей подкаталог).

Используйте команду `l` (`dired-do-redisplay`) для обновления содержимого подкаталога. Для удаления подкаталога вы можете применить команду `k`. См. Раздел 28.14 [Обновление в Dired], с. 300.

## 28.12 Перемещение по подкаталогам

Когда буфер Dired перечисляет подкаталоги, вы можете использовать команды перемещения по страницам `C-x [` и `C-x ]` для перехода через целые каталоги.

Следующие команды передвигают в пределах одного уровня, вверх или вниз по дереву каталогов в одном буфере Dired. Они перемещают к *строкам заголовков каталогов*; это строки, сообщающие имя каталога, они выводятся перед его содержимым.

`C-M-n` Переходит к строке заголовка следующего подкаталога, независимо от его уровня (`dired-next-subdir`).

`C-M-p` Переходит к строке заголовка предыдущего подкаталога, независимо от его уровня (`dired-prev-subdir`).

`C-M-u` Переходит к строке заголовка родительского подкаталога (`dired-tree-up`).

`C-M-d` Переходит вниз по дереву каталогов, к строке заголовка первого подкаталога (`dired-tree-down`).

`<` Перемещает вверх к предыдущей строке файла-каталога (`dired-prev-dirline`). Это строки, описывающие каталог как файл в его родительском каталоге.

`>` Перемещает к следующей строке файла-каталога (`dired-prev-dirline`).

## 28.13 Скрывание подкаталогов

*Скрыть* подкаталог — значит сделать невидимым его содержимое, за исключением строки заголовка, средствами выборочного показа (см. Раздел 11.4 [Выборочный показ], с. 83).

`$` Скрывает или открывает подкаталог, на котором находится точка, и перемещает точку к следующему подкаталогу (`dired-hide-subdir`). Числовой аргумент служит в качестве счетчика повторов.

`M-$` Скрывает все подкаталоги в этом буфере Dired, оставляя только их строки заголовков (`dired-hide-all`). Или, если какой-нибудь подкаталог уже скрыт, делает все подкаталоги снова видимыми. Вы можете использовать эту команду, чтобы получить обзор очень глубоких деревьев каталогов или чтобы быстро переместиться к далеким подкаталогам.

Обычные команды Dired никогда не затрагивают файлы в скрытых подкаталогах. Например, команды, работающие с помеченными файлами, игнорируют файлы в скрытых каталогах, даже если они помечены. Следовательно, вы можете использовать скрывание, чтобы временно исключить подкаталоги из области действия различных операций, не убирая при этом метки.

Команды скрывания подкаталогов переключают; это значит, что они скрывают то, что было видимо, и показывают то, что было скрыто.

## 28.14 Обновление буфера Dired

Этот раздел описывает команды для обновления буфера Dired, чтобы он отражал внешние (сделанные не в Dired) изменения в буферах и файлах, и для удаления части буфера Dired.

`g` Обновляет все содержимое буфера Dired (`revert-buffer`).

`l` Обновляет указанные файлы (`dired-do-redisplay`).

`k` Удаляет заданные *строки файлов* — не сами файлы, а только строки в буфере (`dired-do-kill-lines`).

`s` Переключает между сортировкой в алфавитном порядке и по дате/времени (`dired-sort-toggle-or-edit`).

`C-u s` *переключатели* ⟨RET⟩
Обновляет буфер Dired, используя *переключатели* в качестве `dired-listing-switches`.

Наберите `g` (`revert-buffer`), чтобы содержимое буфера Dired обновилось, основываясь на сделанных в перечисленных файлах и каталогах изменениях. Эта команда сохраняет все метки, кроме меток, стоявших на удаленных файлах. Скрытые подкаталоги обновляются, но остаются скрытыми.

Чтобы обновить только некоторые файлы, наберите `l` (`dired-do-redisplay`). Эта команда применяется к следующим *n* файлам, или к помеченным файлам, если такие есть, или к текущему файлу. Обновление их означает считывание нового статуса из файловой системы и обновление буфера для правильного отображения состояния этих файлов.

Если вы примените `l` на строке заголовка подкаталога, она обновит содержимое этого подкаталога.

Чтобы удалить заданные *строки файлов* — не сами файлы, только их строки — напечатайте `k` (`dired-do-kill-lines`). Запущенная с числовым аргументом *n*, эта команда применяется к следующим *n* файлам; иначе она применяется к помеченным файлам.

Если вы уничтожите строку для файла, являющегося каталогом, содержимое этого каталога также будет удалено из буфера. Другой способ удалить подкаталог из буфера Dired — набрать `C-u k` на строке заголовка этого подкаталога.

Команда `g` возвращает все строки, уничтоженные таким методом, но не возвращает подкаталоги — вы должны использовать `i`, чтобы снова вставить каждый подкаталог.

Файлы в буферах Dired обычно перечисляются в алфавитном порядке по именам. Или Dired может отсортировать их по дате и времени. Команда Dired `s` (`dired-sort-toggle-or-edit`) переключает между этими двумя режимами сортировки. Строка режима в буфере Dired показывает, по какому признаку в данный момент отсортированы файлы: по имени или по дате.

`C-u s` *переключатели* ⟨RET⟩ позволяет вам задать новое значение для `dired-listing-switches`.

## 28.15 Dired и `find`

Вы можете задать набор файлов для показа в буфере Dired более гибким способом, используя для выбора файлов утилиту `find`.

Чтобы найти все файлы, чьи имена соответствуют заданному шаблону, запустите `M-x find-name-dired`. Эта команда считывает аргументы *каталог* и *образец* и выбирает все файлы в *каталоге* или его подкаталогах, чьи имена соответствуют *образцу*.

Выбранные таким способом файлы отображаются в буфере Dired, в котором доступны обычные команды Dired.

Если вы хотите проверять содержимое файлов, а не их имена, используйте `M-x find-grep-dired`. Эта команда считывает в минибуфере два аргумента, *каталог* и *regexp*; она выбирает все файлы в *каталоге* или его подкаталогах, которые содержат совпадения с регулярным выражением *regexp*. Для этого она запускает программы `find` и `grep`. Смотрите также `M-x grep-find`, раздел Раздел 23.1 [Компиляция], с. 247. Помните, что регулярное выражение задается для `grep`, а не для Emacs.

Наиболее общая команда в этой серии — команда `M-x find-dired`, которая позволяет вам указать любое условие, которое может проверить `find`. Эта команда принимает два аргумента минибуфера, *каталог* и *аргументы-find*; она запускает `find` в *каталоге*, передавая *аргументы-find*, чтобы сообщить, какие условия должна проверить `find`. Чтобы использовать эту команду, вы должны уметь пользоваться программой `find`.

Формат распечатки, производимой этими командами, управляется переменной `find-ls-option`; ее значение по умолчанию велит использовать для `ls` ключи ‘`-ld`’. Если ваши распечатки повреждены, вам может понадобиться изменить значение этой переменной.

# 29 Календарь и дневник

Emacs может работать как настольный календарь с ежедневником для планируемых и прошедших событий. Чтобы войти в календарь, наберите `M-x calendar`; это покажет календарь на три месяца, отцентрированный на текущем месяце, а точка будет находиться на текущей дате. С числовым аргументом, как в `C-u M-x calendar`, эта команда запрашивает у вас месяц и год, которые должны оказаться в центре трехмесячного календаря. Календарь использует свой собственный буфер с основным режимом Calendar.

`Mouse-2` в календаре выводит меню операций для даты; `C-Mouse-3` выводит меню часто используемых средств календаря, которые не зависят от конкретной даты. Чтобы выйти из календаря, введите `q`. См. раздел “Calendar” в *The Emacs Lisp Reference Manual*, для получения информации о настройке календаря и дневника.

## 29.1 Перемещение по календарю

Режим Calendar позволяет вам перемещаться по календарю через логические промежутки времени, такие как дни, недели, месяцы и годы. Если вы переместитесь за те три месяца, которые показываются изначально, календарь автоматически “прокручивается” на столько, сколько нужно, чтобы выбранная дата стала видимой. Перемещение к дате позволяет вам просмотреть приходящиеся на нее праздники или записи в дневнике или преобразовать ее в другие календарные системы; перемещение на большие интервалы времени также полезно просто для прокрутки календаря.

### 29.1.1 Перемещение по обычным календарным единицам

Команды для перемещения в буфере календаря аналогичны командам для перемещения в тексте. Вы можете переходить вперед и назад по дням, неделям, месяцам и годам.

`C-f` Перемещает на день вперед (`calendar-forward-day`).

`C-b` Перемещает на день назад (`calendar-backward-day`).

`C-n` Перемещает на неделю вперед (`calendar-forward-week`).

`C-p` Перемещает на неделю назад (`calendar-backward-week`).

`M-}` Перемещает на месяц вперед (`calendar-forward-month`).

`M-{` Перемещает на месяц назад (`calendar-backward-month`).

`C-x ]` Перемещает на год вперед (`calendar-forward-year`).

`C-x [` Перемещает на год назад (`calendar-backward-year`).

Команды для перемещения по дням и неделям — это естественные аналоги обычных команд Emacs для перемещения по знакам или строкам. Так же, как `C-n` обычно перемещает к тому же столбцу в следующей строке, в режиме Calendar она перемещает к тому же дню следующей недели. А `C-p` перемещает к тому же дню предыдущей недели.

Курсорные стрелки эквивалентны `C-f`, `C-b`, `C-n` и `C-p`, как и в других режимах.

Команды для перемещения по месяцам и годам работают так же, как команды для недель, но переносят на большие расстояния. Команды для месяцев `M-}` и `M-{` перемещают вперед или назад на целый месяц. Команды для лет `C-x ]` и `C-x [` перемещают вперед или назад на целый год.

Простейший способ запомнить эти команды — рассматривать месяцы и годы как аналоги абзацев и страниц, соответственно. Но сами команды не вполне аналогичны. Обычные

команды Emacs для абзацев передвигают к началу или концу абзаца, тогда как эти команды для месяцев и лет перемещают по целому месяцу или году, что обычно подразумевает переход за конец месяца или года.

Все эти команды принимают числовой аргумент в качестве счетчика повторов. Для удобства, в режиме Calendar цифры и знак минус задают числовой аргумент даже без модификатора Meta. Например, `100 C-f` перемещает точку на 100 дней вперед от ее текущей позиции.

### 29.1.2 Начало или конец недели, месяца или года

Неделя (или месяц, или год) — это не просто некоторое количество дней; мы полагаем, что недели (месяцы, года) начинаются с определенных дат. Поэтому режим Calendar предоставляет команды для перехода к началу или концу недели, месяца или года:

`C-a` Переход к началу недели (`calendar-beginning-of-week`).

`C-e` Переход к концу недели (`calendar-end-of-week`).

`M-a` Переход к началу месяца (`calendar-beginning-of-month`).

`M-e` Переход к концу месяца (`calendar-end-of-month`).

`M-<` Переход к началу года (`calendar-beginning-of-year`).

`M->` Переход к концу года (`calendar-end-of-year`).

Эти команды также принимают числовые аргументы в качестве счетчиков повторений, который указывает, на сколько недель, месяцев или лет нужно переместиться вперед или назад.

По умолчанию недели начинаются с воскресенья. Чтобы сделать так, чтобы они начинались с понедельника, установите переменную `calendar-week-start-day` в значение 1.

### 29.1.3 Указанные даты

Режим Calendar предоставляет команды для перехода к конкретной дате, которую можно задать многими способами.

`g d` Перемещает точку к указанной дате (`calendar-goto-date`).

`o` Центрирует календарь вокруг указанного месяца (`calendar-other-month`).

`.` Перемещает к сегодняшней дате (`calendar-goto-today`).

`g d` (`calendar-goto-date`) запрашивает год, месяц и день месяца, а затем перемещает к этой дате. Поскольку календарь включает все даты от начала нашей эры, вы должны вводить год полностью; то есть пишите ‘`1990`’, а не ‘`90`’.

`o` (`calendar-other-month`) запрашивает месяц и год, а затем центрирует трехмесячный календарь вокруг этого месяца.

Вы можете вернуться к сегодняшней дате с помощью команды `.` (`calendar-goto-today`).

## 29.2 Прокрутка календаря

Изображение календаря автоматически прокручивается, когда вы сдвигаетесь за пределы видимой части. Вы также можете прокручивать его вручную. Представьте, что окно календаря содержит длинную ленту бумаги с месяцами. Прокрутка ее означает, что в этом окне станут видимыми новые месяцы.

`C-x <` Прокручивает календарь на один месяц вперед (`scroll-calendar-left`).

`C-x >` Прокручивает календарь на один месяц назад (`scroll-calendar-right`).

`C-v` ⟨NEXT⟩ Прокручивает календарь на три месяца вперед (`scroll-calendar-left-three-months`).

`M-v` ⟨PRIOR⟩ Прокручивает календарь на три месяца назад (`scroll-calendar-right-three-months`).

Самые основные команды прокрутки календаря прокручивают по одному месяцу за раз. Это означает, что между отображением до команды и отображением после делается два месяца перекрытия. `C-x <` прокручивает содержимое календаря на месяц влево; то есть она перемещает вид вперед по времени. `C-x >` прокручивает содержимое календаря вправо, что перемещает назад во времени.

Команды `C-v` и `M-v` прокручивают календарь по целым экранам — по три месяца — по аналогии с обычным значением этих команд. `C-v` делает видимыми более поздние даты, а `M-v` — более ранние. Эти команды принимают числовой аргумент в качестве счетчика повторов; в частности, так как `C-u` умножает следующую команду на четыре, набор `C-u C-v` прокручивает календарь на год вперед, а `C-u M-v` прокручивает на год назад.

Функциональные клавиши ⟨NEXT⟩ и ⟨PRIOR⟩ эквивалентны `C-v` и `M-v`, как и в других режимах.

## 29.3 Посчет дней

`M-=` Выводит число дней в текущей области (`calendar-count-days-region`).

Чтобы определить число дней в области, наберите `M-=` (`calendar-count-days-region`). Количество дней выводится *включительно*; то есть дни, указываемые точкой и меткой, учитываются.

## 29.4 Другие команды календаря

`p d` Выводит номер дня в году (`calendar-print-day-of-year`).

`C-c C-l` Заново генерирует окно календаря (`redraw-calendar`).

`SPC` Прокручивает следующее окно (`scroll-other-window`).

`q` Выходит из календаря (`exit-calendar`).

Чтобы напечатать число дней, прошедших с начала года или остающихся до конца года, наберите команду `p d` (`calendar-print-day-of-year`). Она покажет оба этих числа в эхо-области. Количество прошедших дней включает выбранную дату. Число оставшихся дней не включает ее.

Если окно календаря окажется испорчено, наберите `C-c C-l` (`redraw-calendar`) чтобы его перерисовать. (Такое может случиться, только если вы используете команды редактирования не из режима Calendar.)

В режиме Calendar вы можете использовать SPC (scroll-other-window) для прокрутки другого окна. Это удобно, если вы смотрите в другом окне список праздников или записи ежедневника.

Чтобы выйти из календаря, наберите q (exit-calendar). Эта команда прячет все буферы, связанные с календарем, и выбирает другие буферы. (Если фрейм содержит специально назначенное окно календаря, выход из календаря минимизирует этот фрейм.)

## 29.5 LaTeX и календарь

Команды календаря для работы с LaTeX генерируют буфер с LaTeX-кодом, который печатает календарь. В зависимости от использованой команды напечатанный календарь покрывает день, неделю, месяц или год, в котором находится точка.

t m — Генерирует календарь на один месяц (cal-tex-cursor-month).

t M — Генерирует горизонтальный календарь на один месяц (cal-tex-cursor-month-landscape).

t d — Генерирует календарь на один день (cal-tex-cursor-day).

t w 1 — Генерирует одностраничный календарь на одну неделю (cal-tex-cursor-week).

t w 2 — Генерирует двухстраничный календарь на одну неделю (cal-tex-cursor-week2).

t w 3 — Генерирует календарь в стиле ISO на одну неделю (cal-tex-cursor-week-iso).

t w 4 — Генерирует календарь на одну неделю, начинающуюся с понедельника (cal-tex-cursor-week-monday).

t f w — Генерирует двухнедельный календарь на одной странице в стиле Filofax (cal-tex-cursor-filofax-2week).

t f W — Генерирует недельный календарь на одной странице в стиле Filofax (cal-tex-cursor-filofax-week).

t y — Генерирует календарь на один год (cal-tex-cursor-year).

t Y — Генерирует горизонтальный календарь на один год (cal-tex-cursor-year-landscape).

t f y — Генерирует календарь на один год в стиле Filofax (cal-tex-cursor-filofax-year).

Некоторые из этих команд печатают календарь горизонально (в “ландшафтном режиме”), поэтому его ширина может быть больше длины. Некоторые из них используют размер бумаги Filofax (3.75 на 6.75 дюймов). Все эти команды принимают префиксный аргумент, задающий число печатаемых дней, недель, месяцев или лет (всегда начиная от текущего).

Если переменная cal-tex-holidays не равна nil (это значение по умолчанию), то печатаемые календари показывают праздники из calendar-holidays. Если переменная cal-tex-diary отлична от nil (по умолчанию это nil), также включаются записи из дневника (только в недельные и месячные календари). Если отлична от nil переменная cal-tex-rules (по умолчанию nil), то в тех стилях распечатки календаря, где достаточно места, страницы разлинованы.

## 29.6 Праздники

Календарь Emacs знает обо всех больших и о многих малых праздниках и может их отображать.

`h` Показать праздники, приходящиеся на выбранную дату (`calendar-cursor-holidays`).

`Mouse-2 Holidays`
Показать все праздники для даты, на которой вы щелкнули.

`x` Пометить праздники (`mark-calendar-holidays`).

`u` Снять метки в окне календаря (`calendar-unmark`).

`a` Перечислить в другом окне все праздники для показанных трех месяцев (`list-calendar-holidays`).

`M-x holidays`
Перечислить в другом окне все праздники для трех месяцев вокруг текущей даты.

`M-x list-holidays`
Перечислить в другом окне праздники для заданного промежутка лет.

Чтобы узнать, не приходится ли на данную дату праздников, расположите точку на этой дате в окне календаря и используйте команду `h`. Или щелкните на этой дате `Mouse-2` и затем выберите из появившегося меню пункт `Holidays`. В обоих случаях выводятся праздники для заданной даты, в эхо-области, если они там умещаются, или в отдельном окне.

Чтобы просмотреть распределение праздников для всех показанных в календаре дат, используйте команду `x`. Она отображает праздничные дни другим начертанием (или помещает после этих дат значок '`*`', если множественные начертания недоступны). Эта команда применяется и к месяцам, видимым в данный момент, и к тем, что станут впоследствии видимыми при прокрутке. Чтобы выключить разметку и стереть текущие метки, наберите `u`, что также стирает метки дневника (см. Раздел 29.10 [Дневник], с. 313).

Чтобы получить еще более подробную информацию, используйте команду `a`, которая показывает отдельный буфер, содержащий список всех праздников в текущем трехмесячном диапазоне. Вы можете использовать ⟨SPC⟩ в окне календаря для прокрутки этого списка.

Команда `M-x holidays` выводит список праздников для текущего, предыдущего и следующего месяца; она работает, даже если у вас нет окна календаря. Если вы хотите получить перечень праздников, центрированный вокруг другого месяца, используйте `C-u M-x holidays`, которая спрашивает месяц и год.

Известные Emacs праздники включают праздники США и основные христианские, иудейские и исламские праздники; также он знает о солнцестояниях и равноденствиях.

Команда `M-x list-holidays` выводит список праздников для нескольких лет. Эта функция запрашивает у вас начальный и конечный год и позволяет выбрать все праздники или одну из их категорий. Вы можете использовать эту команду, даже если у вас нет окна календаря.

Emacs использует для праздников даты, основанные на *текущей практике*, а не на исторических фактах. К примеру, исторически момент сезонного перевода времени и даже само его существование изменялось от года к году, но текущий закон США говорит, что время переводится в первое воскресенье апреля. Когда правила сезонного перевода времени установлены по американским нормам, Emacs всегда использует текущее определение, даже если оно было неправильно в некоторые прошлые годы.

## 29.7 Время восхода и заката Солнца

Особые команды календаря могут сообщить вам с точностью до минуты-двух времена восхода и заката Солнца для любой даты.

`S` Показывает времена восхода и заката Солнца для выбранной даты (`calendar-sunrise-sunset`).

`Mouse-2 Sunrise/Sunset`
Показывает времена восхода и заката Солнца для даты, на которой вы щелкнули.

`M-x sunrise-sunset`
Показывает времена восхода и заката Солнца для сегодняшней даты.

`C-u M-x sunrise-sunset`
Показывает времена восхода и заката Солнца для заданной даты.

Чтобы просмотреть в эхо-области *местное время* восхода и заката Солнца, находясь в календаре, переместите точку к желаемой дате и нажмите `S`. Или щелкните на этой дате `Mouse-2` и выберите из появившегося меню пункт `Sunrise/Sunset`. Команда `M-x sunrise-sunset`, доступная извне календаря, показывает эту информацию для текущей или указанной даты. Чтобы задать отличную от сегодняшней дату, используйте команду `C-u M-x sunrise-sunset`, которая спросит год, месяц и день.

Вы можете просмотреть время восхода и заката Солнца для любой местности и даты с помощью `C-u C-u M-x sunrise-sunset`. Эта команда запрашивает вас широту, долготу, разницу от универсального координированного времени и дату, а затем сообщает вам времена восхода и заката для этого места и этой даты.

Поскольку время восхода и заката Солнца зависит от положения на Земле, перед использованием этих команд вы должны сказать Emacs вашу широту, долготу и название местности. Вот пример задаваемых значений:

```
(setq calendar-latitude 40.1)
(setq calendar-longitude -88.2)
(setq calendar-location-name "Urbana, IL")
```

Задавайте значения `calendar-latitude` и `calendar-longitude` с точностью до одной десятой.

Ваш часовой пояс также влияет на время восхода и заката. Обычно Emacs получает информацию о часовом поясе от операционной системы, но если это не те значения, которые вы хотите (или если операционная система их не предоставляет), вы должны установить их сами. Вот пример:

```
(setq calendar-time-zone -360)
(setq calendar-standard-time-zone-name "CST")
(setq calendar-daylight-time-zone-name "CDT")
```

Значение `calendar-time-zone` — это разница между вашим местным стандартным и универсальным координированным (гринвичским) временем, выраженная в минутах. Значения `calendar-standard-time-zone-name` и `calendar-daylight-time-zone-name` — это сокращения, принятые для обозначения вашего часового пояса. Emacs показывает время восхода и заката в *с учетом сезонного перевода часов*. См. Раздел 29.12 [Сезонный перевод часов], с. 319, о том, как определяется сезонная поправка.

Как пользователь, вы можете счесть удобным установить переменные календаря для задания вашего обычного физического местонахождения в вашем файле ‘`.emacs`’. А когда вы устанавливаете Emacs на машине, вы можете создать файл ‘`default.el`’, который будет правильно их устанавливать для типичного местонахождения большинства пользователей этой машины. См. Раздел 31.7 [Файл инициализации], с. 366.

## 29.8 Фазы Луны

Эти команды календаря показывают даты и времена лунных фаз (новолуние, первая четверть, полнолуние, последняя четверть). Эта возможность полезна при отладке проблем, которые “зависят от фазы Луны”.

`M` Выводит даты и времена всех четвертей Луны для показанного трехмесячного периода (`calendar-phases-of-moon`).

`M-x phases-of-moon`
Выводит даты и времена четвертей Луны для трех месяцев вокруг сегодняшней даты.

Внутри календаря, для показа в отдельном буфере фаз Луны для текущего трехмесячного периода используется команда `M`. Перечислямые даты и времена точны до нескольких минут.

Вне календаря, для показа фаз Луны для текущего, предыдущего и следующего месяца используйте команду `M-x phases-of-moon`. Чтобы получить сведения для какого-то другого месяца, используйте `C-u M-x phases-of-moon`, она спросит вас о месяце и годе.

Даты и времена лунных фаз даются в местном времени (с учетом сезонных поправок, если это необходимо); но если переменная `calendar-time-zone` не определена, используется универсальное координированное время (гринвичский часовой пояс). См. Раздел 29.12 [Сезонный перевод часов], с. 319.

## 29.9 Преобразование в другие календарные системы и из них

В Emacs *всегда* показывается григорианский календарь, иногда называемый календарем “нового стиля”, который в наши дни применяется почти везде. Однако, этот календарь не существовал до шестнадцатого столетия и не использовался широко до восемнадцатого; только в начале девятнадцатого века он полностью вытеснил юлианский календарь и получил всемирное признание. Календарь Emacs может отобразить любой месяц начиная от января первого года нашей эры, но всегда показывает григорианский календарь, даже для дат, когда его еще не существовало.

Хотя Emacs не может показывать другие календари, он умеет преобразовавать даты из нескольких других календарных систем.

### 29.9.1 Поддерживаемые календарные системы

Коммерческий календарь ISO используется в основном в Европе.

Юлианский календарь, названный в честь Юлия Цезаря, использовался в Европе в средние века и во многих других странах вплоть до девятнадцатого века.

Астрономы используют простой подсчет дней, истекших от полудня понедельника, первого января, 4713-го года до нашей эры по юлианскому календарю. Число прошедших дней называется *юлианским* или *астрономическим номером дня*.

Иудейский календарь по традиции используется в еврейской религии. Календарная программа Emacs использует иудейский календарь для определения дат еврейских праздников. Даты иудейского календаря начинаются и завершаются при заходе Солнца.

Исламский календарь используется во многих мусульманских странах. Emacs использует его для определения дат исламских праздников. В исламском мире нет всеобщей договоренности о календаре; в Emacs применяется широко распространенная версия, но точные даты исламского календаря часто основываются на высказываниях религиозных авторитетов, а не на вычислениях. Как следствие, действительные даты наблюдения могут слегка отличаться от вычисленных в Emacs. Даты исламского календаря начинаются и завершаются при заходе Солнца.

Календарь французской революции был создан якобинцами после революции 1789-го года для введения более светского и основанного на представлениях о природе взгляда на годовой цикл и для установления десятидневной недели, более рационалистической, как метрическая система. Французское правительство официально отменило этот календарь в конце 1805-го года.

Центральноамериканские Майя использовали три разные, перекрывающиеся календарные системы, *долгий счет*, *тцолкин* и *хааб*. Emacs знает обо всех трех календарях. Эксперты обсуждают точное соответствие между календарем Майя и нашим календарем; Emacs использует для вычислений корреляцию Гудмана-Мартинеса-Томсона.

Копты используют календарь, основанный на древнеегипетском солнечном календаре. Он состоит из двенадцати тридцатидневных месяцев, за которым идет дополнительный пятидневный период. Раз в четыре года к этому периоду добавляется шестой, високосный день. Эфиопский календарь обладает такой же структурой, но в нем другие номера лет и названия месяцев.

Персы используют солнечный календарь, основанный на разработках Омара Хайама. Их календарь состоит из двенадцати месяцев, первые шесть из которых содержат 31 дней, следующие пять — 30 дней, а в последнем 29 дней в обычные года и 30 дней в високосные. Високосные года случаются по сложному образцу каждые четыре или пять лет.

Китайский календарь — это сложная система лунных месяцев, скомбинированных с солнечными годами. Годы проходят по циклам из шестидесяти лет, обычный год содержит 12 месяцев, високосный — 13; в каждом месяце либо 29, либо 30 дней. Каждый год, обычный месяц и день именуется комбинацией одного из десяти “небесных стеблей” и одной из двенадцати “земных ветвей”, в общей сложности есть шестьдесят имен, которые проходят по кругу.

### 29.9.2 Преобразование в другой календарь

Следующие команды описывают выбранную дату (дату, в которой находится точка) в различных календарных системах:

`Mouse-2 Other Calendars`
: Выводит дату, на которой вы щелкнули, выраженную через различные другие календари.

`p c`
: Выводит эквивалент выбранного дня в коммерческом календаре ISO (`calendar-print-iso-date`).

`p j`
: Выводит юлианскую дату для выбранного дня (`calendar-print-julian-date`).

`p a`
: Выводит астрономический (юлианский) номер дня для выбранной даты (`calendar-print-astro-day-number`).

`p h`
: Показывает иудейскую дату для выбранного дня (`calendar-print-hebrew-date`).

`p i`
: Показывает исламскую дату для выбранного дня (`calendar-print-islamic-date`).

`p f`
: Показывает французскую революционную дату для выбранного дня (`calendar-print-french-date`).

`p C`
: Показывает китайскую дату для выбранного дня (`calendar-print-chinese-date`).

`p k`
: Показывает коптскую дату для выбранного дня (`calendar-print-coptic-date`).

`p e` Показывает эфиопскую дату для выбранного дня (`calendar-print-ethiopic-date`).

`p p` Показывает дату персидского календаря для выбранного дня (`calendar-print-persian-date`).

`p m` Показывает дату календаря Майя для выбранного дня (`calendar-print-mayan-date`).

Если вы используете X, простейший способ перевести дату в другие календари — щелкнуть на ней `Mouse-2` и выбрать из появившегося меню пункт `Other Calendars`. Это покажет в виде меню эквивалентные формы для данной даты во всех календарях, которые понимает Emacs. (Выбор альтернатив из этого меню не делает ничего — оно используется только для показа.)

Переместите точку к желаемой дате в григорианском календаре и нажмите подходящие клавиши. `p` — это мнемоника для “print”, поскольку Emacs “печатает” эквивалентную дату в эхо-области.

### 29.9.3 Преобразование из другого календаря

Вы можете использовать поддерживаемые календари для задания даты, к которой следует переместиться. Этот раздел описывает команды для проделывания этого во всех календарях, кроме майянского; чтобы узнать, как это сделать для календаря Майя, смотрите следующий раздел.

`g c` Переход к дате, заданной в коммерческом календаре ISO (`calendar-goto-iso-date`).

`g j` Переход к дате, заданной в юлианском календаре (`calendar-goto-julian-date`).

`g a` Переход к дате, заданной астрономическим (юлианским) номером дня (`calendar-goto-astro-day-number`).

`g h` Переход к дате, заданной в иудейском календаре (`calendar-goto-hebrew-date`).

`g i` Переход к дате, заданной в исламском календаре (`calendar-goto-islamic-date`).

`g f` Переход к дате, заданной во французском революционном календаре (`calendar-goto-french-date`).

`g C` Переход к дате, заданной в китайском календаре (`calendar-goto-chinese-date`).

`g p` Переход к дате, заданной в персидском календаре (`calendar-goto-persian-date`).

`g k` Переход к дате, заданной в коптском календаре (`calendar-goto-coptic-date`).

`g e` Переход к дате, заданной в эфиопском календаре (`calendar-goto-ethiopic-date`).

Эти команды запрашивают у вас дату из другого календаря, перемещают точку к эквивалентной дате григорианского календаря и показывают дату другого календаря в эхо-области. При считывании названия месяца Emacs использует строгое завершение (см. Раздел 5.3 [Завершение], с. 47), поэтому вам не нужно беспокоиться о правильности написания иудейских, исламских или французских названий.

Один из вопросов, часто возникающих в связи с иудейским календарем, — вычисление годовщины смерти, называемой “yahrzeit”. Календарь Emacs включает средства для таких

вычислений. Если вы находитесь в календаре, команда `M-x list-yahrzeit-dates` спрашивает вас о промежутке лет и затем показывает для этого промежутка список yahrzeit для даты в точке. Если вы не в календаре, эта команда сначала спросит о дате смерти и промежутке лет, а затем покажет список дат yahrzeit.

### 29.9.4 Преобразование из календаря Майя

Это команды для выбора дат, оснаванных на календаре Майя:

`g m l` Переходит к дате, заданной календарем долгого счета (`calendar-goto-mayan-long-count-date`).

`g m n t` Переходит к следующему появлению некоторого места в календаре тцолкин (`calendar-next-tzolkin-date`).

`g m p t` Переходит к предыдущему появлению некоторого места в календаре тцолкин (`calendar-previous-tzolkin-date`).

`g m n h` Переходит к следующему появлению некоторого места в календаре хааб (`calendar-next-haab-date`).

`g m p h` Переходит к предыдущему появлению некоторого места в календаре хааб (`calendar-previous-haab-date`).

`g m n c` Переходит к следующему появлению некоторого места в календарном круге (`calendar-next-calendar-round-date`).

`g m p c` Переходит к предыдущему появлению некоторого места в календарном круге (`calendar-previous-calendar-round-date`).

Чтобы понять эти команды, вы должны понимать календари Майя. *Долгий счет* — это подсчет дней в следующих единицах:

1 кин = 1 день  1 уинал = 20 кин  1 тун = 18 уинал
1 катун = 20 тун  1 бактун = 20 катун

Таким образом, дата 12.16.11.16.6 в долгом счете означает 12 бактун, 16 катун, 11 тун, 16 уинал и 6 кин. Календарь Emacs может обрабатывать майянские даты долгого счета от 7.17.18.13.1, но не более ранние. Когда вы используете команду `g m l`, набирайте майянскую дату долгого счета, разделяя бактун, катун, тун, уинал и кин точками.

Майянский календарь тцолкин — это цикл из 260 дней, формируемый парой независимых циклов из 13 и 20 дней. Поскольку этот цикл повторяется бесконечно, Emacs предоставляет команды для перемещения назад и вперед к предыдущей или следующей точке цикла. Наберите `g m p t`, чтобы перейти к предыдущей дате в тцолкин; Emacs спросит у вас дату в тцолкин и переместит точку к предыдущему появлению этой даты. Аналогично, наберите `g m n t`, чтобы перейти к следующему появлению даты в тцолкин.

Майянский календарь хааб — это цикл из 365 дней, собранный из 18 месяцев по 20 в каждом, за которыми следует пятидневный период без месяца. Подобно циклу тцолкин, этот цикл повторяется бесконечно, и есть команды для перехода назад и вперед к предыдущей или следующей точке этого цикла. Наберите `g m p h`, чтобы перейти к предыдущей дате хааб; Emacs спросит у вас дату в хааб и переместит точку к предыдущему появлению этой даты. Аналогично, наберите `g m n h`, чтобы перейти к следующему появлению даты в хааб.

Майя также использовали комбинацию дат тцолкин и хааб. Эта комбинация — цикл примерно в 52 года, называемый *календарным кругом*. Если вы наберете `g m p c`, Emacs спросит у вас даты хааб и тцолкин и затем переместит точку к предыдущему появлению этой комбинации. Используйте `g m n c` для перемещения точки к следующему появлению комбинации. Эти команды сообщают об ошибке, если набранная вами комбинация дат хааб/тцолкин невозможна.

При запросе майянской даты Emacs использует строгое завершение (см. Раздел 5.3.3 [Строгое завершение], с. 48), поэтому вам не нужно беспокоиться о правильности написания.

## 29.10 Дневник

С помощью дневника Emacs можно прослеживать назначенные встречи и другие события на ежедневной основе и в совокупности с календарем. Чтобы воспользоваться этими возможностями, вы должны сначала создать *файл дневника*, содержащий перечень событий и их дат. Тогда Emacs сможет автоматически подобрать и показать вам события на сегодняшний день, на ближайшее будущее или для указанной даты.

По умолчанию в качестве файла дневника Emacs использует файл '`~/diary`'. Это тот же самый файл, что применяеися в утилите `calendar`. Вот пример файла '`~/diary`':

```
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.
```

В данном примере для выравнивания описаний в большинстве вхождений событий использованы пробелы. Такое форматирование полностью является делом вкуса.

Хотя вероятнее всего вы начнете составлять ежедневник вручную, Emacs предоставляет вам несколько команд для просмотра, добавления и изменения записей дневника.

### 29.10.1 Команды для просмотра записей в дневнике

После того как вы создали файл '`~/diary`', вы можете просматривать его из календаря. Вы также можете просмотреть список сегодняшних событий извне режима Calendar.

`d` Показывает все записи дневника для выбранной даты (`view-diary-entries`).

`Mouse-2 Diary`
: Показывает все записи дневника для даты, на которой вы щелкнули.

`s` Показывает весь файл дневника (`show-all-diary-entries`).

`m` Помечает все видимые даты, для которых есть записи в дневнике (`mark-diary-entries`).

`u` Убирает метки из окна календаря (`calendar-unmark`).

`M-x print-diary-entries`
: Печатает твердую копию отображаения дневника в том виде, в каком он есть на экране.

`M-x diary` Показывает все вхождения дневника для сегодняшней даты.

`M-x diary-mail-entries`
: Отправляет вам по почте напоминание о предстоящих событиях.

Отображение записей дневника с помощью `d` показывает в отдельном окне вхождения дневника для выбранной в календаре даты. Строка режима нового окна показывает дату для этих записей и все выпадающие на нее праздники. Если вы зададите для `d` числовой аргумент, она покажет все записи дневника на указанное число дней. То есть, `2 d` покажет все записи для выбранной даты и для следующего дня.

Другой способ просмотреть записи дневника для какой-либо даты — щелкнуть на этой дате `Mouse-2`, а затем выбрать из появившегося меню пункт `Diary`.

Чтобы получить более широкий обзор упомянутых в календаре дней, используйте команду `m`. Она показывает даты, для которых в дневнике есть записи, другим начертанием (или помещает после них знак '`+`', если дисплей не может отобразить несколько начертаний). Эта команда применяется как к видимым в данный момент месяцам, так и к остальным, которые становятся видимыми при последующей прокрутке. Чтобы выключить разметку и стереть текущие метки, нажмите `u`, что выключает также и метки праздников (см. Раздел 29.6 [Праздники], с. 307).

Чтобы увидеть файл дневника полностью, а не только некоторые его вхождения, используйте команду `s`.

При выводе избранных записей дневника применяется средство выборочного показа, чтобы скрыть ненужные записи.

Буфер дневника в той форме, как вы его видите — это иллюзия, поэтому простая его печать не даст то, что вы видите на экране. Для получения твердой копии буфера дневника *таким, каким он виден*, есть особая команда; это команда `M-x print-diary-entries`. Она посылает данные непосредственно на принтер. Вы можете настраивать ее, как `lpr-region` (см. Раздел 30.4 [Распечатка], с. 331).

Команда `M-x diary` показывает записи дневника для текущей даты, независимо от того, виден ли календарь, и кроме того, возможно, для нескольких следующих дней; переменная `number-of-diary-entries` задает число включаемых дней. См. раздел "Calendar" в *The Emacs Lisp Reference Manual*.

Если вы поместите в ваш файл '`.emacs`' строку `(diary)`, то при входе в Emacs вам будет автоматически показываться окно с записями дневника на сегодняшний день. Строка режима этого окна показывает дату и выпадающие на нее праздники.

Многим пользователям нравится получать напоминания о событиях из ежедневника по почте. Чтобы послать себе такие сообщения, используйте команду `M-x diary-mail-entries`. Префиксный аргумент указывает, сколько дней (начиная с сегодняшнего) следует проверять; иначе число дней определяется по переменной `diary-mail-days`.

### 29.10.2 Файл дневника

Ваш *файл дневника* — это файл, в котором записаны события, связанные с определенными датами. Имя этого файла задается переменной `diary-file`; по умолчанию это '`~/diary`'. Программа `calendar` поддерживает подмножество формата, допускаемого средствами дневника в Emacs, так что вы можете использовать эту утилиту для просмотра файла дневника с разумными результатами, если не считать вхождений, которые она не понимает.

Каждая запись в файле дневника описывает одно событие и состоит из одной или нескольких строк. Запись всегда начинается со спецификации даты с левого края. Остальная часть записи — это просто текст, описывающий событие. Если запись содержит более одной строки, то строки после первой должны начинаться с отступа, как указание на то, что они продолжают предыдущее вхождение. Строки, которые не начинаются с правильной даты и не продолжают предыдущее вхождение, игнорируются.

Вы можете подавить разметку определенных записей из дневника в окне календаря; чтобы сделать так, вставьте в начале записи перед датой знак '`&`'. Это не влияет на

отображение записи в окне дневника; это затрагивает только метки дат в окне календаря. Запрет пометки записи особенно полезен для обобщенных вхождений, которые иначе пометили бы слишком много разных дней.

Если первая строка записи состоит только из даты или названия дня недели, и после нее нет пропусков или пунктуации, то такая строка не выводится в окне дневника; там появляются только строки продолжения. Например, такая запись:

```
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
```

появляется в окне дневника без строки с датой в начале. Такой стиль записей выглядит красивее, если вы просматриваете вхождения для одного дня, но может запутать, если вы попросите показать записи для нескольких дней.

Вы можете редактировать записи дневника в том виде, как они появляются в окне, но важно помнить, что этот показанный буфер содержит *полный* файл дневника, но некоторые его части скрыты из вида. Это означает в частности, что команда `C-f` (`forward-char`) может поместить точку в таком месте, которое кажется концом строки, но на самом деле это середина какой-то скрытой строки.

**Будьте внимательны при редактировании записей дневника!** Вставка новых строк или добавление/удаление знаков в середине видимой строки не могут причинить проблем, но редактирование в конце строки может делать не то, что вы ожидаете. Удаление строки может удалить другие невидимые записи, которые следуют за ней. Перед редактированием дневника лучше всего отобразить файл полностью с помощью команды `s` (`show-all-diary-entries`).

### 29.10.3 Формат дат

Вот несколько примерных записей дневника, иллюстрирующих различные способы форматирования даты. Все эти примеры показывают даты, записанные в американском порядке (месяц, день, год), но режим Calendar поддерживает европейский порядок (день, месяц, год) как альтернативу.

```
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
```

Первая запись появляется только один раз, 20 апреля 1993 года. Вторая и третья появляются каждый год в заданные дни, а четвертая использует для месяца символ подстановки (звездочку), поэтому она появляется 25-го числа каждого месяца. Последняя запись появляется каждую неделю в пятницу.

Для выражения даты вы можете использовать только числа, как '*месяц*`/`*день*' или '*месяц*`/`*день*`/`*год*'. После этого должна идти не-цифра. В самой дате, *месяц* и *день* должны быть однозначными или двузначными числами. Необязательный *год* — это тоже число, его можно сокращать до последних двух цифр; то есть вы можете писать как '`11/12/1989`', так и '`11/12/89`'.

Даты также можно записывать в формате '*название-месяца день*' или '*название-месяца день*`,` *год*', где название месяца можно писать полностью или сокращать до трех букв (с точкой или без). Регистр значения не имеет.

Дата может быть *обобщенной*, то есть частично недоопределенной. Тогда она применяется ко всем датам, соответствующим спецификации. Если дата не содержит год, то она обобщенная и относится ко всем годам. Или же *месяц*, *день* или *год* могут быть знаком ‘*’; это соответствует любому месяцу, дню или году. Таким образом, вхождение дневника ‘3/*/*’ соответствует любому дню марта в каждом году; то же и для ‘march *’.

Если вы предпочитаете европейский стиль написания дат — когда день идет перед месяцем — наберите в календаре M-x european-calendar или установите переменную european-calendar-style в значение t *перед* использованием любой команды календаря или дневника. Этот режим интерпретирует все даты в дневнике в европейском стиле, а также применяет европейский стиль при отображении дат дневника. (Заметьте, что после *названия-месяца* в европейском стиле нет запятой.) Чтобы вернуться к американскому (принимаемому по умолчанию) стилю написания дат, наберите M-x american-calendar.

Вы можете использовать название дня недели в качестве обобщенной даты, которая относится к любой дате, приходящейся на этот день недели. Названия дней недели можно сокращать до трех букв (с точкой или без) или писать полностью; регистр значения не имеет.

### 29.10.4 Команды для добавления к дневнику

Когда вы находитесь в календаре, вам доступны несколько команд для создания записей в дневнике:

i d — Добавляет в дневник запись для выбранной даты (insert-diary-entry).

i w — Добавляет в дневник запись для выбранного дня недели (insert-weekly-diary-entry).

i m — Добавляет в дневник запись для выбранного дня месяца (insert-monthly-diary-entry).

i y — Добавляет в дневник запись для выбранного дня года (insert-yearly-diary-entry).

Вы можете создать в дневнике запись для конкретной даты, выбрав эту дату в окне календаря и выполнив команду i d. Эта команда показывает конец вашего файла дневника в другом окне и вставляет дату; затем вы можете напечатать остальную часть записи.

Если вы хотите сделать в дневнике запись, которая относится к какому-то дню недели, выберите этот день (можно любое его появление) и наберите i w. Это вставит название дня недели как обобщенную дату; потом вы можете ввести остальную часть записи. Таким же способом вы можете создать ежемесячную запись. Выберите день месяца, примените команду i m и введите остаток записи. Аналогично вы можете вставить ежегодную запись при помощи команды i y.

Все описанные выше команды по умолчанию создают помечаемые записи. Чтобы сделать в дневнике непомечаемую запись, дайте команде числовой аргумент. Например, C-u i w создает в дневнике непомечаемую еженедельную запись.

Когда вы измените файл дневника, убедитесь, что вы сохранили его перед выходом из Emacs.

### 29.10.5 Особые записи дневника

Помимо вхождений, основанных на календарных датах, файл дневника может содержать *sexp-вхождения* для регулярных событий, таких как годовщины. Такие записи основаны на лисповских выражениях (s-выражениях), которые Emacs выполняет по мере сканирования файла дневника. Вместо даты sexp-вхождение содержит строку ‘%%’, за которой

идет лисповское выражение, заключенное в круглые скобки. Это выражение определяет, к каким датам относится данная запись.

Режим Calendar предоставляет команды для вставки некоторых часто используемых sexp-вхождений:

`i a` Добавляет в дневник запись о годовщине события для текущей даты (`insert-anniversary-diary-entry`).

`i b` Добавляет блочную запись дневника для текущей области (`insert-block-diary-entry`).

`i c` Добавляет циклическую запись дневника, начинающуюся от текущей даты (`insert-cyclic-diary-entry`).

Если вы хотите создать в дневнике запись, которая относится к годовщине какой-то даты, переместите точку к этой дате и используйте команду `i a`. Это покажет конец вашего файла дневника в другом окне и вставит описание годовщины; затем вы можете набрать остальную часть записи. Запись выглядит так:

```
%%(diary-anniversary 10 31 1948) Arthur's birthday
```

Эта запись относится к 31 октября каждого года после 1948; '`10 31 1948`' задает дату. (Если вы используете европейский стиль календаря, месяц и день меняются местами.) Причина того, что это выражение требует указания начального года, состоит в том, что продвинутые функции календаря могут использовать его для подсчета истекших лет.

*Блочная* запись дневника применяется к заданному промежутку последовательных дат. Вот блочная запись, которая относится ко всем датам от 24 июня 1990 до 10 июля 1990 года.

```
%%(diary-block 6 24 1990 7 10 1990) Vacation
```

'`6 24 1990`' обозначает начальную дату, а '`7 10 1990`' — конечную. (Опять же, если вы используете европейский стиль календаря, месяц и день меняются местами.)

Чтобы вставить блочную запись, поместите точку и метку на двух датах, которые начинают и завершают промежуток, и наберите `i b`. Эта команда показывает конец вашего файла дневника в другом окне и вставляет описание блока; потом вы можете напечатать остальную часть записи.

*Циклические* записи дневника повторяются после фиксированного интервала дней. Чтобы создать такую запись, выберите начальную дату и используйте команду `i c`. Эта команда спросит у вас длину интервала, а затем вставит запись, выглядящую следующим образом:

```
%%(diary-cyclic 50 3 1 1990) Renew medication
```

Эта запись относится к 1 марта 1990 года и к каждому 50-ому последующему дню; '`3 1 1990`' задает начальную дату. (Если вы используете европейский стиль календаря, месяц и день меняются местами.)

Все три эти команды создают помечаемые записи. Чтобы вставить непомечаемую запись, дайте команде числовой аргумент. Например, `C-u i a` создает непомечаемую запись для годовщины некоторого события.

Пометка sexp-вхождений дневника в календаре *чрезвычайно* требовательна ко времени, поскольку должна быть отдельно проверена каждая видимая в окне календаря дата. Поэтому лучше делать sexp-вхождения дневника непомечаемыми (с '`&`'), если возможно.

Другой, усложненный вид sexp-вхождения, *плавающая* запись, задает регулярно случающееся событие по сдвигу, выраженному в днях, неделях и месяцах. Ее можно сравнить с записями crontab, интерпретируемыми утилитой `cron`. Вот непомечаемая плавающая запись, которая относится к последнему четвергу ноября:

```
&%%(diary-float 11 4 -1) American Thanksgiving
```

Число 11 обозначает ноябрь (одиннадцатый месяц), 4 обозначает четверг (четвертый день недели, где воскресенье считается нулем), а −1 обозначает “последний” (1 обозначало бы “первый”, 2 обозначало бы “второй”, −2 — “предпоследний” и так далее). Месяц может быть единственным или списком месяцев. Таким образом, вы могли бы изменить 11 выше на ‘`’(1 2 3)`’ и получить запись, относящуюся к последнему четвергу января, февраля и марта. Если месяц задан как `t`, запись относится ко всем месяцам года.

В самом общем виде, sexp-вхождения дневника могут производить для определения своей применимости любые вычисления. См. раздел “Sexp Diary Entries” в *The Emacs Lisp Reference Manual*.

## 29.11 Напоминания о назначенных событиях

Если у вас в дневнике есть запись о назначенном событии, и эта запись начинается с распознаваемого описания времени, Emacs может предупредить вас за несколько минут о приближении этого времени. Emacs предупреждает вас о назначении, показывая сообщение в строке режима.

Чтобы включить напоминание о назначенных событиях, вы должны задействовать средство Emacs для показа времени, `M-x display-time` (см. Раздел 1.3 [Строка режима], с. 25). Вы также должны добавить к ловушке `diary-hook` функцию `appt-make-list` следующим образом:

```
(add-hook 'diary-hook 'appt-make-list)
```

Помещение такого текста в ваш файл ‘`.emacs`’ делает все сразу:

```
(display-time)
(add-hook 'diary-hook 'appt-make-list)
(diary 0)
```

Если сделана такая подготовка, то когда вы отображаете дневник (с помощью команды `d` в окне календаря или через `M-x diary`), для всех записей дневника, в которых есть распознаваемые описания времени, настраивается список назначенных событий, и перед каждым из них вам делается напоминание.

Предположим например, что файл дневника содержит такие строки:

```
Monday
  9:30am Coffee break
 12:00pm Lunch
```

Тогда по понедельникам, после того как вы отобразите дневник, вам будут напоминать в 9:20 о перерыве для кофе и в 11:50 об обеде.

Вы можете записывать время в стиле am/pm (где ‘`12:00am`’ обозначает полночь, а ‘`12:00pm`’ — полдень) или в двадцатичетырехчасовом европейском/военном стиле. Необязательно быть постоянным; файл дневника может содержать смесь этих двух стилей.

Emacs автоматически обновляет список напоминаний сразу после полуночи. Это также отображает в буфере дневника записи для следующего дня, если только вы не установили `appt-display-diary` равной `nil`.

Вы также можете использовать средства напоминания о назначенных событиях в качестве будильника. Команда `M-x appt-add` добавляет записи в список напоминаний, не затрагивая ваш файл дневника. Для удаления записей из списка напоминаний применяется команда `M-x appt-delete`.

Вы можете в любое время выключить средство напоминания о назначенных событиях, установив `appt-issue-message` в значение `nil`.

## 29.12 Сезонный перевод времени

Emacs понимает разницу между стандартным и сезонным временем — времена, выдаваемые для восходов и закатов Солнца, солнцестояний, равноденствий и фаз Луны, учитывают его. Правила сезонного перевода времени меняются от места к месту и кроме того колебались исторически от года к году. Чтобы делать свою работу правильно, Emacs должен знать, какое правило использовать.

Некоторые операционные системы отслеживают правила, применяющиеся в той местности, где вы находитесь; на таких системах Emacs получает необходимую информацию от системы автоматически. Если часть этой информации или вся она неизвестна, Emacs заполняет пробелы правилами, используемыми в данное время в Кембридже, Массачусетс. Если получившиеся правила — это не то, что вы хотите, вы можете сообщить Emacs нужные правила, устанавливая определенные переменные: `calendar-daylight-savings-starts` и `calendar-daylight-savings-ends`.

Значениями этих переменных должны быть лисповские выражения, ссылающиеся на переменную `year` и вычисляющие григорианскую дату перехода на зимнее и летнее время в форме списка (*месяц день год*). Эти значения должны быть равны `nil`, если в вашей местности не бывает сезонного перевода времени.

Emacs использует эти выражения для определения даты перевода времени при создании списка праздников и для корректровки времени при вычислениях, связанных с Луной и Солнцем.

Для массачусетского Кембриджа эти значения таковы:

```
(calendar-nth-named-day 1 0 4 year)
(calendar-nth-named-day -1 0 10 year)
```

Это обозначает первый нулевой день недели (воскресенье) четвертого месяца (апреля) в году, задаваемом переменной `year`, и последнее воскресенье десятого месяца (октября) в этом же году. Если момент перевода времени сменился бы на 1 октября, вы установили бы переменную `calendar-daylight-savings-starts` в такое значение:

```
(list 10 1 year)
```

Если в вашей местности время не переводится, или если вы хотите получать всегда стандартное время, установите `calendar-daylight-savings-starts` и `calendar-daylight-savings-ends` равными `nil`.

Переменная `calendar-daylight-time-offset` задает разницу во времени после перевода, измеряемую в минутах. Для массачусетского Кембриджа это 60.

Переменные `calendar-daylight-savings-starts-time` и `calendar-daylight-savings-ends-time` задают число минут после полуночи по местному времени, когда происходит перевод часов. Для массачусетского Кембриджа значения обоих этих переменных равны 120.

# 30 Разнообразные команды

Эта глава содержит несколько небольших тем, которые не вписываются в другие главы: чтение сетевых новостей, запуск команд оболочки и подпроцессов оболочки, использование одного разделяемого Emacs для утилит, которые предполагают запуск редактора как подпроцесса, печать твердой копии, сортировка текста, сужение отображения до части буфера, редактирование двухколоночных и двоичных файлов, сохранение сеансов Emacs для последующего продолжения, эмуляция других редакторов и различные развлечения.

## 30.1 Gnus

Gnus — это пакет Emacs, разработанный в первую очередь для чтения и отправки новостей Usenet. Его также можно использовать для чтения и написания ответов на сообщения из многих других источников — почты, удаленных каталогов, дайджестов и других.

Здесь мы даем введение в Gnus и описываем некоторые основные возможности. Для получения подробной информации о Gnus наберите `M-x info` и выберите затем руководство по Gnus.

Чтобы запустить Gnus, напечатайте `M-x gnus` ⟨RET⟩.

### 30.1.1 Буферы Gnus

В противоположность большинству обычных пакетов Emacs, Gnus использует для показа информации и получения команд несколько разных буферов. Большую часть времени пользователи проводят в трех буферах: *буфере групп*, *буфере резюме* и *буфере статьи*.

*Буфер групп* содержит перечень групп. Это первый буфер, который Gnus показывает после запуска. Обычно в нем показаны только те группы, на которые вы подписаны, и в которых есть непрочтенные статьи. Используйте этот буфер для выбора конкретной группы.

*Буфер резюме* построчно перечисляет статьи одной группы. По умолчанию для каждой статьи показываются автор, заголовок и число строк, но это можно настроить по своему вкусу, как и большую часть того, что отображает Gnus. Буфер резюме создается, когда вы выбираете группу в буфере групп, и уничтожается, когда вы покидаете эту группу. Используйте этот буфер для выбора статьи.

*Буфер статьи* показывает саму статью. При обычном использовании Gnus вы не выбираете этот буфер — все полезные команды, предназначенные для действий над статьей, работают из буфера резюме. Но вы можете выбрать буфер статьи и выполнять все команды Gnus из него, если хотите.

### 30.1.2 Когда Gnus запускается

При запуске Gnus считывает ваш файл инициализации новостей '`.newsrc`' и пытается установить связь с локальным сервером новостей, который служит хранилищем статей. Сервер новостей не обязан быть тем же компьютером, на который вы вошли.

Если вы запустили Gnus и соединились с сервером, но не видите в буфере групп ни одной группы, наберите `L` или `A k`, чтобы получить перечень всех групп. Затем нажимайте `u`, чтобы переключать подписку на группы.

Когда вы запускаете Gnus первый раз, он подписывает вас на несколько избранных групп. Все остальные группы сначала *уничтожены* с вашей точки зрения; вы можете получить их перечень с помощью `A k`. Все новые группы, появляющиеся в дальнейшем на сервере, становятся для вас *зомбированными*; наберите `A z`, чтобы получить их перечень. Вы можете подписаться на группы, показанные в этих списках, используя команду `u`.

Когда вы покидаете Gnus при помощи `q`, он автоматически записывает в ваших файлах инициализации '`.newsrc`' и '`.newsrc.eld`' статус подписки всех групп. Обычно вам не стоит редактировать эти файлы вручную, но вы можете это делать, если знаете как.

### 30.1.3 Обзор команд Gnus

Чтение новостей — это двухшаговый процесс:

1. Выберите группу в буфере групп.
2. Выбирайте статьи в буфере резюме. Каждая выбранная статья показывается в буфере статьи в большом окне под буфером резюме в маленьком окне.

Каждый буфер Gnus имеет свои особые команды; однако, смысл любого данного ключа в различных буферах Gnus обычно аналогичен, даже если и различается. Вот команды буферов групп и резюме:

`q` В буфере групп, обновляет файл инициализации '`.newsrc`' и покидает Gnus.

В буфере резюме, покидает текущую группу и возвращает в буфер групп. Таким образом, дважды нажав `q`, вы выйдете из Gnus.

`L` В буфере групп, перечисляет все доступные группы на вашем сервере новостей (кроме тех, что вы уничтожили). Это может быть длинный список!

`l` В буфере групп, перечисляет только те группы, на которые вы подписаны, и которые содержат непрочтенные статьи.

`u` В буфере групп, отменяет подписку (или устанавливает ее) на группу, перечисленную в строке, в которой находится точка. Когда вы выходите из Gnus, нажав `q`, Gnus перечисляет в вашем файле '`.newsrc`' те группы, на которые вы подписаны. При следующем запуске Gnus вы не увидите эту группу, потому что обычно Gnus показывает только группы, на которые вы подписаны.

`C-k` В буфере групп, "уничтожает" группу на текущей строке — даже не перечисляет ее отныне в '`.newsrc`'. Это затрагивает как текущий сеанс Gnus, так и последующие.

Когда вы покидаете Gnus при помощи `q`, Gnus записывает информацию в файле '`.newsrc`', описывая все группы, кроме тех, что вы "уничтожили".

⟨SPC⟩ В буфере групп, выбирает группу на строке под курсором и показывает первую непрочтенную статью в этой группе.

В буфере резюме,

- Выбирает статью под курсором, если ни одна еще не выбрана.
- Прокручивает текст текущей статьи (если такая есть).
- Выбирает следующую непрочтенную статью, если текущая статья кончилась.

Таким образом, вы можете пройти по всем статьям, последовательно нажимая ⟨SPC⟩.

⟨DEL⟩ В буфере групп, перемещает точку к предыдущей группе, содержащей непрочтенные статьи.

В буфере резюме, прокручивает текст статьи назад.

`n` Перемещает точку к следующей непрочитанной группе или выбирает следующую непрочитанную статью.

`p` Перемещает точку к предыдущей непрочитанной группе или выбирает предыдущую непрочитанную статью.

`C-n`
`C-p` Перемещает точку к следующему или предыдущему пункту, даже если он помечен как прочтенный. Это не выбирает группу или статью на той строке.

`s` В буфере резюме, начинает наращиваемый поиск в тексте текущего буфера статьи, точно так же, как если бы вы переключились в буфер статьи и набрали `C-s`.

`M-s` *regexp* ⟨RET⟩
В буфере резюме, производит поиск статей, содержащих совпадение с *regexp*.

## 30.2 Запуск команд оболочки из Emacs

В Emacs есть команды для передачи одиночных командных строк подчиненным процессам оболочки. Существует возможность интерактивного запуска оболочки с вводом и выводом в буфер Emacs с именем '`*shell*`'.

`M-!` *кмд* ⟨RET⟩
Запустить командную строку оболочки *кмд* и показать ее вывод (`shell-command`).

`M-|` *кмд* ⟨RET⟩
Запустить командную строку оболочки *кмд* с содержимым области в качестве ввода; возможна замена содержимого области выводом команды (`shell-command-on-region`).

`M-x shell` Запустить подоболочку с вводом и выводом через буфер Emacs. Затем вы можете задавать команды интерактивно.

### 30.2.1 Отдельные команды оболочки

`M-!` (`shell-command`) считывает в минибуфере строку текста и выполняет ее как команду оболочки в подоболочке, созданной только для этой команды. Стандартный ввод команде поступает из нулевого устройства. Если команда оболочки производит какой-либо вывод, то он поступает в буфер Emacs с именем '`*Shell Command Output*`', который отражается не в выбранном, а в другом окне. Числовой аргумент, как в `M-1 M-!`, велит команде вставить весь вывод в текущий буфер. В этом случае точка остается перед выводом, а метка устанавливается за ним.

Если командная строка оболочки завершается на '`&`', она выполняется асинхронно. Для синхронной команды оболочки `shell-command` возвращает выходное значение этой команды (0 обозначает успех), когда она вызывается из Лисп-программы.

`M-|` (`shell-command-on-region`) похожа на `M-!`, но команде оболочки передается в качестве стандартного ввода содержимое области, а не пустота. Если используется числовой аргумент, означающий вставку вывода в текущий буфер, то старая область сначала удаляется, а потом заменяется выводом. Она возвращает выходное значение команды, когда запускается из лисповской программы.

Обе команды `M-!` и `M-|` используют оболочку, указанную переменной `shell-file-name`. При запуске Emacs эта переменная инициализируется на основании вашей переменной среды `SHELL`. Если в имени этого файла не указывается каталог, то просматриваются каталоги в списке `exec-path`; этот список инициализируется при запуске Emacs по переменной среды `PATH`. Ваш файл '`.emacs`' может отменять либо одну, либо обе эти инициализации по умолчанию.

И `M-!` и `M-|` ожидают завершения команды оболочки. Чтобы остановить ожидание, используйте команду `C-g`; она завершает команду оболочки сигналом `SIGINT` — тем же сигналом, который обычно генерируется оболочкой при вводе `C-c`. Emacs ждет, пока эта

команда на самом деле завершится. Если команда оболочки не остановилась (потому что она игнорирует сигнал SIGINT), наберите C-g снова; это пошлет сигнал SIGKILL, который невозможно проигнорировать.

Чтобы указать систему кодирования для M-! или M-|, используйте команду C-x ⟨RET⟩ c непосредственно перед ними. См. Раздел 18.9 [Задание кодирования], с. 168.

Сообщения команды об ошибках обычно перемежаются с обычным выводом. Если вы установите переменную shell-command-default-error-buffer равной строке, являющейся именем буфера, протокол ошибок будет вставляться перед точкой в буфере с этим именем.

### 30.2.2 Интерактивная подчиненная оболочка

Для запуска интерактивной подоболочки с сохранением протокола в буфере Emacs применяется M-x shell. Эта команда создает (или вновь использует) буфер с именем '*shell*' и запускает подоболочку с вводом, приходящим из этого буфера, и выводом, идущим в него. То есть, любой "терминальный ввод" для подоболочки приходит из текста в буфере, а любой "терминальный вывод" из подоболочки поступает в буфер, продвигая точку вперед. Для передачи ввода в подоболочку необходимо отправиться в конец буфера, набрать нужное и завершить набором ⟨RET⟩.

Emacs не ждет, пока подоболочка что-либо сделает. Можно переключать окна или буферы и редактировать их, пока оболочка ожидает, или пока она выполняет команду. Вывод из подоболочки ждет до тех пор, пока у Emacs не появится время на его обработку; прием происходит всякий раз, когда Emacs ожидает ввода с клавиатуры, а также когда есть свободное время.

В качестве имени файла для загрузки подоболочки используется значение переменной explicit-shell-file-name, если оно не nil. В противном случае, используется переменная среды ESHELL если она установлена, или SHELL. Если указанное имя файла является относительным, просматриваются каталоги в списке exec-path; этот инициализируется по переменной среды PATH во время запуска Emacs. Ваш файл '.emacs' может перекрыть одну или обе из этих инициализаций.

Чтобы указать для оболочки систему кодирования, вы можете использовать команду C-x ⟨RET⟩ c непосредственно перед M-x shell. Вы также можете задать систему кодирования после запуска оболочки с помощью команды C-x ⟨RET⟩ p в ее буфере. См. Раздел 18.9 [Задание кодирования], с. 168.

Как только оболочка запущена, на вход ей подается содержимое файла '~/.emacs_*имя-оболочки*', если этот файл существует, где *имя-оболочки* является именем файла, из которого загружается оболочка. Например, если вы используете bash, то ей посылается файл '~/.emacs_bash'.

Команды cd, pushd и popd, передаваемые подчиненной оболочке, отслеживаются в Emacs так, чтобы каталог по умолчанию буфера '*shell*' всегда совпадал с рабочим каталогом оболочки. Эти команды распознаются синтаксически проверкой посылаемых строк ввода. Если вы используете для этих команд псевдонимы, то вы можете указать Emacs, что их тоже следует распознавать. Например, если значение переменной shell-pushd-regexp соответствует началу командной строки оболочки, то эта строка воспринимается как команда pushd. Если для 'pushd' используются псевдонимы, то необходимо изменить эту переменную. Аналогично, shell-popd-regexp и shell-cd-regexp используются для распознавания команд, обозначающих 'popd' и 'cd'. Причем эти команды распознаются только в начале командной строки оболочки.

Если Emacs получает ошибку при попытке обработать то, что он считает командами 'cd', 'pushd' или 'popd', он запускает ловушку shell-set-directory-error-hook (см. Раздел 31.2.3 [Ловушки], с. 349).

Если Emacs не отслеживает правильно изменения текущего каталога подоболочки, используйте команду M-x dirs, чтобы спросить оболочку, какой у нее текущий каталог. Эта

команда работает для оболочек, поддерживающих наиболее распространенный синтаксис команд; она не работает с необычными оболочками.

Вы также можете использовать `M-x dirtrack-mode`, чтобы включить (или выключить) альтернативный и более агрессивный метод отслеживания изменений текущего каталога.

Emacs определяет в подоболочке переменную среды `EMACS` со значением `t`. Сценарий оболочки может проверить эту переменную, чтобы определить, запущен ли он из подоболочки Emacs.

### 30.2.3 Режим Shell

Буферы оболочки использует режим Shell, в котором определено несколько специальных ключей, привязанных к префиксу `C-c`. Они выбраны так, что имитируют обычные клавиши редактирования и управления заданиями, присутствующие в оболочках вне Emacs, с той лишь разницей, что сначала вы должны набрать `C-c`. Ниже приведен полный список таких ключей режима Shell.

⟨RET⟩ В конце буфера, посылает строку в качестве ввода; в противном случае копирует текущую строку в конец буфера и посылает ее (`send-shell-input`). При копировании строки любой текст в ее начале, соответствующий переменной `shell-prompt-pattern`, пропускается; значение этой переменной должно быть регулярным выражением, которое соответствует подсказкам, используемым в вашей оболочке.

⟨TAB⟩ Завершает имя команды или файла перед точкой в буфере оболочки (`comint-dynamic-complete`). ⟨TAB⟩ также завершает ссылки на историю (см. Раздел 30.2.4.3 [Ссылки на историю], с. 328) и имена переменных среды.

Переменная `shell-completion-fignore` задает список расширений файлов, которые должны игнорироваться при завершении в режиме Shell. Установка по умолчанию игнорирует имена файлов, заканчивающиеся на '`~`', '`#`' или '`%`'. Другие родственные с Comint режимы используют переменную `comint-completion-fignore`.

`M-?` Временно показывает список возможных завершений имени файла перед точкой в буфере оболочки (`comint-dynamic-list-filename-completions`).

`C-d` Либо удаляет один знак, либо посылает EOF (`comint-delchar-or-maybe-eof`). Набранный в конце буфера оболочки, `C-d` посылает подоболочке EOF. Набранный в любой другой позиции в этом буфере, `C-d` удаляет один знак, как обычно.

`C-c C-a` Перемещает в начало строки, но после подсказки, если она есть (`comint-bol`). Если вы наберете эту команду два раза подряд, на второй раз она возвращает к метке процесса, то есть к началу ввода, который вы еще не послали подоболочке. (Обычно это одно и то же место — конец подсказки в текущей строке — но после `C-c` ⟨SPC⟩ метка процесса может оказаться на предыдущей строке.)

`C-c` ⟨SPC⟩ Накапливает несколько строк ввода, и потом посылает их вместе. Эта команда вставляет перед точкой перевод строки, но не посылает предшествующий текст на вход подоболочки — по крайней мере не сейчас. Обе строки, и та, что перед переводом строки, и та, что после, будут посланы вместе (и с разделяющим их переводом строки), когда вы нажмете ⟨RET⟩.

`C-c C-u` Уничтожает весь текст, который еще не был послан в качестве ввода (`comint-kill-input`).

`C-c C-w` Уничтожает слово перед точкой (`backward-kill-word`).

`C-c C-c` Прерывает оболочку или ее текущее подзадание, если оно существует (`comint-interrupt-subjob`). Эта команда уничтожает также весь предназначенный для ввода текст, который еще не был послан.

`C-c C-z` Останавливает оболочку или ее текущее подзадание, если оно существует (`comint-stop-subjob`). Эта команда уничтожает также весь текст, который еще не был послан в качестве ввода.

`C-c C-\` Посылает сигнал выхода оболочке или ее текущему подзаданию, если оно существует (`comint-quit-subjob`). Эта команда уничтожает также весь предназначенный для ввода текст, который еще не был послан.

`C-c C-o` Уничтожает последнюю порцию вывода от команды оболочки (`comint-kill-output`). Это полезно, если команда оболочки извергает много текста, который только мешает.

`C-c C-r`
`C-M-l` Прокручивает окно так, чтобы начало последней порции вывода оказалось вверху; также перемещает туда курсор (`comint-show-output`).

`C-c C-e` Прокручивает окно, чтобы начало последней порции вывода оказалось внизу (`comint-show-maximum-output`).

`C-c C-f` Перемещает вперед на одну команду оболочки, но в пределах текущей строки (`shell-forward-command`). Переменная `shell-command-regexp` указывает, как распознать конец команды.

`C-c C-b` Перемещает назад на одну команду оболочки, но в пределах текущей строки (`shell-backward-command`).

`C-c C-l` Показывает историю команд оболочки этого буфера в другом окне (`comint-dynamic-list-input-ring`).

`M-x dirs` Спрашивает оболочку о текущем каталоге, чтобы Emacs скоординировался с оболочкой.

`M-x send-invisible` ⟨RET⟩ *текст* ⟨RET⟩
Считывает *текст* и посылает его на вход оболочки без эхо. Это полезно, когда команда оболочки запускает программу, спрашивающую пароль.

Или вы можете сообщить Emacs, как распознавать запросы пароля и выключать для них эхо:

```
(add-hook 'comint-output-filter-functions
          'comint-watch-for-password-prompt)
```

`M-x comint-continue-subjob`
Возобновляет процесс оболочки. Это полезно, если вы нечаянно приостановили процесс оболочки.[1]

`M-x comint-strip-ctrl-m`
Убирает все знаки control-M из текущей порции вывода оболочки. Наиболее удобный способ использовать эту команду — запускать ее автоматически, когда вы получаете вывод от подоболочки. Чтобы сделать так, вычислите следующее лисповское выражение:

```
(add-hook 'comint-output-filter-functions
          'comint-strip-ctrl-m)
```

`M-x comint-truncate-buffer`
Эта команда усекает буфер оболочки до определенного максимального числа строк, задаваемого переменной `comint-buffer-maximum-size`. Это можно сделать автоматически каждый раз при получении вывода от подоболочки таким способом:

[1] Вы не должны приостанавливать процесс оболочки. Приостановка подзадания в оболочке — это совсем другое дело, это обычная практика, но для продолжения подзадания вы используете саму оболочку; данная команда этого не сделает.

```
(add-hook 'comint-output-filter-functions
          'comint-truncate-buffer)
```

Режим Shell также настраивает команды работы с абзацами таким образом, что только подсказки оболочки начинают абзацы. Таким образом, абзац состоит из введенной команды плюс из следующего за ней в буфере вывода.

Режим Shell происходит от режима Comint, режима общего назначения для общения с интерактивными подпроцессами. Большинство возможностей режима Shell в действительности дает режим Comint, как вы можете понять из имен перечисленных выше команд. Особые средства режима Shell включают выбор регулярного выражения для распознавания подсказки, средство отслеживания каталогов и несколько пользовательских команд.

Другие средства Emacs, использующие варианты режима Comint, включают GUD (см. Раздел 23.5 [Отладчики], с. 249) и `M-x run-lisp` (см. Раздел 23.10 [Запуск внешнего Лиспа], с. 256).

Вы можете использовать `M-x comint-run` для выполнения любой программы по вашему выбору в неизмененном режиме Comint — без особенностей режима Shell.

### 30.2.4 История команд оболочки

Буферы оболочки поддерживают три способа повторения более ранних команд. Вы можете использовать те же ключи, что используются в минибуфере; они работают во многом так же, как в минибуфере, вставляя текст предыдущих команд, всегда оставляя точку в конце буфера. Вы можете переместиться по буферу назад к предыдущим командам в их начальной позиции и затем снова послать их или скопировать в конец. Или вы можете использовать знак '`!`' для ссылок на старые команды.

#### 30.2.4.1 Список истории оболочки

`M-p` Извлекает следующую более старую команду оболочки.

`M-n` Извлекает следующую более новую команду оболочки.

`M-r` *regexp* ⟨RET⟩
`M-s` *regexp* ⟨RET⟩
Производит поиск команды оболочки, соответствующей регулярному выражению *regexp*, вперед или назад.

`C-c C-x` (режим Shell)
Извлекает следующую команду в списке истории.

Буферы оболочки предоставляют историю ранее введенных команд. Чтобы снова использовать команды оболочки, сохраненные в истории, используйте команды редактирования `M-p`, `M-n`, `M-r` и `M-s`. Они работают так же, как команды истории минибуфера, за тем лишь исключением, что действуют на текст в конце буфера оболочки, где вы вставляли бы текст для отправки оболочке.

`M-p` вставляет более раннюю команду оболочки в буфер оболочки. Последовательное применение `M-p` извлекает последовательно все более ранние команды оболочки, каждый раз замещая любой текст, уже существовавший в качестве потенциального ввода для оболочки. `M-n` работает похоже, но последовательно находит более поздние команды оболочки из этого буфера.

Команды поиска в истории, `M-r` и `M-s`, считывают регулярное выражение и производят поиск совпадающей команды в истории. Кроме предоставления выбора, какую именно команду вы хотите извлечь, они работают точно так же, как `M-p` и `M-r`. Если вы введете пустое регулярное выражение, будет использовано то регулярное выражение, которое вы предоставили в последний раз.

Когда вы нашли желаемый предыдущий ввод, вы можете снова послать его, нажав ⟨RET⟩, или сначала отредактировать и затем послать, если хотите.

Часто бывает полезно заново выполнить несколько последовательных команд оболочки, которые ранее выполнялись по порядку. Чтобы сделать это, сначала найдите и выполните первую команду в последовательности. Затем наберите `C-c C-x`; это извлечет следующую команду — ту, которая шла за только что повторенной. Затем нажмите ⟨RET⟩, чтобы заново выполнить эту команду. Вы можете повторить несколько последовательных команд, набирая `C-c C-x` ⟨RET⟩ снова и снова.

Эти команды получают текст более ранних команд оболочки из специального списка истории, не из самого буфера оболочки. Поэтому редактирование буфера оболочки или даже уничтожение больших его частей не влияет на историю, к которой обращаются эти команды.

Некоторые оболочки сохраняют истории их команд в файлах, чтобы вы могли сослаться на старые команды из предыдущих сеансовх. Emacs считывает файл истории команд для выбранной вами оболочки, чтобы проинициализировать свою собственную историю команд. Этот файл называется '`~/.bash_history`' в bash, '`~/.sh_history`' в ksh и '`~/.history`' в других оболочках.

### 30.2.4.2 Копирование истории оболочки

`C-c C-p` Перемещает точку к предыдущей подсказке (`comint-previous-prompt`).

`C-c C-n` Перемещает точку к следующей подсказке (`comint-next-prompt`).

`C-c` ⟨RET⟩ Копирует команду ввода, в которой находится точка, вставляя ее копию в конец буфера (`comint-copy-old-input`). Это полезно, если вы переместили точку назад к предыдущей команде. После того, как вы скопировали эту команду, вы можете послать копию в качестве ввода, нажав ⟨RET⟩. Если вы хотите, вы можете отредактировать копию перед отправлением.

Перемещение к предыдущему вводу и последующее его копирование с помощью `C-c` ⟨RET⟩ дает тот же результат — то же содержимое буфера — какой вы получили бы применением `M-p` достаточное число раз, чтобы извлечь эту старую команду из списка истории. Однако, `C-c` ⟨RET⟩ копирует текст из буфера, которые может отличаться от того, что находится в списке истории, если вы редактировали в буфере введенный текст после того, как он был послан.

### 30.2.4.3 Ссылки на историю оболочки

Различные оболочки, включая csh и bash, поддерживают *ссылки на историю*, которые начинаются с '`!`' и '`^`'. Режим Shell может понимать такие конструкции и делать для вас подстановку. Если вы вставили ссылку на историю и нажали ⟨TAB⟩, это приведет к поиску совпадающей команды в истории ввода, подстановке, если она необходима, и помещению в буфер результата на место ссылки. Например, вы можете извлечь самую недавнюю команду, начинающуюся на '`mv`', с помощью `! m v` ⟨TAB⟩. Вы можете отредактировать эту команду, если хотите, и затем послать ее оболочке, нажав ⟨RET⟩.

Ссылки на историю действуют только после подсказки оболочки. Переменная `shell-prompt-pattern` указывает, как распознать подсказку. Вообще, режимы Comint используют для определения подсказки переменную `comint-prompt-regexp`; режим Shell использует `shell-prompt-pattern`, чтобы установить локальное значение `comint-prompt-regexp`.

В режиме Shell есть возможность раскрывать ссылки на историю, когда вы отправляете их оболочке. Чтобы затребовать это, установите переменную `comint-input-autoexpand` равной `input`.

Вы можете сделать так, чтобы ⟨SPC⟩ производил раскрытие истории, привязав ⟨SPC⟩ к команде `comint-magic-space`.

### 30.2.5 Параметры режима Shell

Если переменная `comint-scroll-to-bottom-on-input` не равна `nil`, команды вставки и восстановления прокручивают выбранное окно книзу перед вставкой.

Если `comint-scroll-show-maximum-output` не равна `nil`, то прокрутка из-за поступления вывода старается разместить последнюю строку текста на нижней строке окна, чтобы вы видели как можно больше полезного текста. (Это имитирует поведение прокрутки на многих терминалах.) По умолчанию эта переменная равна `nil`.

Установкой `comint-scroll-to-bottom-on-output` вы можете сделать так, чтобы точка перескакивала в конец буфера всякий раз при поступлении вывода — независимо от того, где точка была раньше. Если значение равно `this`, точка перескакивает в выбранном окне. Если значение равно `all`, точка перескакивает в каждом окне, показывающем этот буфер Comint. Если значение равно `other`, точка перескакивает во всех невыбранных окнах, показывающих текущий буфер. По умолчанию это `nil`, что означает, что точка не должна перемещаться в конец.

Переменная `comint-input-ignoredups` говорит, нужно ли сохранять в истории последовательные одинаковые строки ввода. Отличное от `nil` значение велит опускать ввод, идентичный предыдущему. По умолчанию эта переменная равна `nil`; это значит, что сохраняется любой ввод, даже если он эквивалентен предыдущему.

Завершение имен файлов управляется тремя переменными. Переменная `comint-completion-addsuffix` говорит, вставляет ли завершение пробел или косую черту, чтобы обозначить полностью завершенное имя файла или каталога (не-`nil` велит вставлять пробел или косую черту). `comint-completion-recexact`, если не равна `nil`, указывает ⟨TAB⟩ выбирать наименьшее возможное завершение, если обычный алгоритм завершения Emacs не может добавить даже одного знака. `comint-completion-autolist`, если не равна `nil`, велит перечислять все возможные завершения, когда нельзя найти точное завершение.

Команда `comint-dynamic-complete-variable` завершает имя переменной, используя установки переменных среды внутри Emacs. Переменные, управляющие завершением имен файлов, применяются и к завершению имен переменных. Эта команда обычно доступна через меню.

При завершении команд обычно рассматриваются только исполняемые файлы. Если вы установите `shell-command-execonly` равной `nil`, будут рассматриваться также имена и неисполняемых файлов.

Вы можете сконфигурировать поведение ‘`pushd`’. Есть переменные, которые указывают, ведет ли себя `pushd`, как `cd`, если ей не задан аргумент (`shell-pushd-tohome`), выталкивает ли она каталог, а не прокручивает, если ей задан числовой аргумент (`shell-pushd-dextract`), и добавляет ли она каталоги в стек только в том случае, если их еще нет в нем (`shell-pushd-dunique`). Выбранные вами значения должны, разумеется, соответствовать вашей оболочке.

### 30.2.6 Оболочка на удаленной машине

Emacs предоставляет две команды для захода на другой компьютер и общения с ним через буфер Emacs.

`M-x telnet` ⟨RET⟩ *имя-машины* ⟨RET⟩
: Устанавливает с компьютером *имя-машины* соединение по Telnet.

`M-x rlogin` ⟨RET⟩ *имя-машины* ⟨RET⟩
: Устанавливает с компьютером *имя-машины* соединение по Rlogin.

Используйте `M-x telnet`, чтобы установить соединение по Telnet с другим компьютером. (Telnet — это стандартный протокол Internet для захода на удаленную систему.) Она

считывает в минибуфере имя другого компьютера в качестве аргумента. Когда соединение установлено, общение с другим компьютером работает похоже на общение с подоболочкой: вы можете редактировать ввод с помощью обычных команд Emacs и посылать его построчно, набирая ⟨RET⟩. Вывод вставляется в буфер вперемешку со вводом.

Используйте `M-x rlogin` для установки соединения по Rlogin. Rlogin — это другой протокол общения с удаленной системой, во многом похожий на Telnet, но не совместимый с ним и поддерживаемый только на некоторых системах. Преимущества Rlogin состоят в том, что вы можете сделать так, чтобы вам необязательно было задавать имя пользователя и пароль при общении между часто используемыми машинами, и что вы можете установить восьмибитное соединение. (Чтобы сделать это в Emacs, установите `rlogin-explicit-args` равной `("-8")` перед запуском Rlogin.)

`M-x rlogin` устанавливает каталог по умолчанию данного буфера Emacs, чтобы получать доступ к удаленной машине через FTP (см. Раздел 14.1 [Имена файлов], с. 105), и отслеживает команды оболочки, которые изменяют текущий каталог, так же, как режим Shell.

Есть два способа отслеживания каталогов в буфере Rlogin — либо с помощью имен удаленных каталогов ‘*/машина:кат/*’, либо с помощью локальных имен (это работает, если “удаленная” машина разделяет файловые системы с вашей начальной машиной). Вы можете использовать команду `rlogin-directory-tracking-mode`, чтобы переключать эти режимы. Отсутствие аргумента обозначает использование имен удаленных каталогов, положительный аргумент обозначает использование локальных имен, а отрицательный выключает отслеживание каталогов.

## 30.3 Использование Emacs в качестве сервера

Различные программы, такие как `mail`, могут вызывать выбранный вами редактор для редактирования определенного текста, например, отправляемого сообщения. По соглашению, большинство этих программ используют переменную среды `EDITOR`, чтобы определить, какой редактор надо запускать. Если вы установите `EDITOR` равной ‘`emacs`’, они вызовут Emacs — но неудобным способом, запуская новый отдельный процесс Emacs. Это неудобно, потому что занимает время и потому что новый процесс Emacs не разделяет буферы с существующим процессом.

Вы можете сделать так, чтобы в качестве редактора для программ вроде `mail` использовался ваш существующий процесс Emacs, путем применения клиента и сервера Emacs. Вот как это делается.

Во-первых, подготовка. Внутри Emacs, вызовите функцию `server-start`. (Ваш файл ‘`.emacs`’ может делать это автоматически, если вы добавите в него выражение `(server-start)`.) Затем, извне Emacs, установите переменную среды `EDITOR` равной ‘`emacsclient`’. (Заметьте, что некоторые программы используют другую переменную среды; например, чтобы TEX использовал ‘`emacsclient`’, вам нужно установить переменную среды `TEXEDIT` равной ‘`emacsclient +%d %s`’.)

Впоследствии, когда любая программа вызывает указанную программу `EDITOR`, в результате вашему главному Emacs будет отправлено сообщение, чтобы он обратился к файлу. (Программа `emacsclient` делает именно это.) Emacs немедленно показывает этот буфер, и вы сразу можете начать его редактирование.

Когда вы завершите редактирование этого буфера, наберите `C-x #` (`server-edit`). Это сохранит файл и пошлет программе `emacsclient` сообщение, приказывающее выйти. Программы, использующие `EDITOR`, ожидают, пока “редактор” (на самом деле, `emacsclient`) не выйдет. `C-x #` также проверяет другие отложенные внешние запросы на редактирование различных файлов и выбирает следующий.

Вы можете переключиться в серверный буфер вручную, если хотите; необязательно попадать в него с помощью `C-x #`. Но `C-x #` дает единственный способ сказать, что "закончили" с текущим.

Если вы установите переменную `server-window` равной окну или фрейму, `C-x #` будет показывать серверный буфер в этом окне или фрейме.

Пока `mail` или другое приложение ожидает завершения `emacsclient`, `emacsclient` не читает терминальный ввод. Поэтому терминал, который использовала `mail`, как бы блокируется на это время. Чтобы редактировать в вашем главном Emacs, вам нужна возможность использовать Emacs без этого терминала. Есть два способа добиться этого:

- Используя оконную систему, запускайте `mail` и главный Emacs в двух разных окнах. Пока `mail` ожидает `emacsclient`, окно, в котором она запущена, блокируется, но вы можете использовать Emacs, переключив окно.
- Используйте для запуска других программ, таких как `mail`, режим Shell в Emacs; тогда `emacsclient` блокирует только подоболочку в Emacs, и вы можете продолжать использовать Emacs для редактирования этого файла.

Некоторые программы записывают для вашего редактирования временные файлы. После того, как вы отредактировали такой временный файл, программа считывает его и удаляет. Если сервер Emacs позже попросят отредактировать файл с тем же именем, он не должен предполагать, что этот файл имеет какое-либо отношение к предыдущему появлению этого же имени. Сервер делает это, уничтожая буфер временного файла, когда вы закончили с ним. Используйте переменную `server-temp-file-regexp`, чтобы указать, какие файлы являются временными в этом смысле; ее значением должно быть регулярное выражение, совпадающее с именами временных файлов.

Если вы запускаете `emacsclient` с ключом '`-no-wait`', он возвращается сразу, не дожидаясь, пока вы "завершите" с буфером в Emacs.

## 30.4 Вывод твердой копии

Команды Emacs для создания твердой копии позволяют вам напечатать весь буфер или только его часть с заголовками или без них. Смотрите также команды печати Dired (см. Раздел 14.10 [Другие действия над файлами], с. 132) и дневника (см. Раздел 29.10.1 [Команды дневника], с. 313).

`M-x print-buffer`
: Выдать распечатку текущего буфера с заголовками, содержащими имя файла и номер страницы.

`M-x lpr-buffer`
: Выдать распечатку текущего буфера без заголовков страниц.

`M-x print-region`
: Как `print-buffer`, но печатать только текущую область.

`M-x lpr-region`
: Как `lpr-buffer`, но печатать только текущую область.

Все команды печати (кроме использующих Postscript) передают программе `lpr` дополнительные ключи, базирующиеся на значении переменной `lpr-switches`. Ее значение должно быть списком строк, причем каждая строка — это ключ, начинающийся с '`-`'. Например, чтобы сделать ширину строк равной восьмидесяти столбцам для всех распечаток, получаемых из Emacs, установите `lpr-switches` так:

```
(setq lpr-switches '("-w80"))
```

Вы можете указать, какой принтер должен использоваться, установив переменную `printer-name`.

Переменная `lpr-command` задает имя используемой программы печати; значение по умолчанию зависит от типа вашей операционной системы. На большинстве систем это `"lpr"`. Переменная `lpr-headers-switches` похожим образом задает дополнительные ключи для создания заголовков страниц. Переменная `lpr-add-switches` указывает, нужно ли передавать программе печати ключи '`-T`' и '`-J`' (подходящие для `lpr`): `nil` означает, что добавлять их не надо. `lpr-add-switches` должна быть равна `nil`, если ваша программа печати не совместима с `lpr`.

## 30.5 Печать через Postscript

Эти команды преобразуют содержимое буфера в Postscript и либо печатают его, либо оставляют в другом буфере Emacs.

`M-x ps-print-buffer`
: Выводит распечатку текущего буфера в форме Postscript.

`M-x ps-print-region`
: Выводит распечатку текущей области в форме Postscript.

`M-x ps-print-buffer-with-faces`
: Выводит распечатку текущего буфера в форме Postscript, показывая использованные в тексте начертания средствами Postscript.

`M-x ps-print-region-with-faces`
: Выводит распечатку текущей области в форме Postscript, показывая использованные в тексте начертания средствами Postscript.

`M-x ps-spool-buffer`
: Генерирует Postscript для текста текущего буфера.

`M-x ps-spool-region`
: Генерирует Postscript для текущей области.

`M-x ps-spool-buffer-with-faces`
: Генерирует Postscript для текущего буфера, показывая использованные начертания.

`M-x ps-spool-region-with-faces`
: Генерирует Postscript для текущей области, показывая использованные начертания.

Команды работы с Postscript, `ps-print-buffer` и `ps-print-region`, печатают содержимое буфера в форме Postscript. Одна команда печатает весь буфер, другая — только область. Соответствующие команды с окончанием '`-with-faces`', `ps-print-buffer-with-faces` и `ps-print-region-with-faces`, используют средства Postscript для передачи начертаний (шрифтов и цветов) в свойствах печатаемого текста.

Если вы используете цветной дисплей, вы можете напечатать буфер, содержащий код программы, с цветовой подсветкой, включив в этом буфере режим Font-Lock и вызвав `ps-print-buffer-with-faces`.

Команды, чьи имена содержат '`spool`' на месте '`print`', генерируют вывод Postscript в буфере Emacs, а не посылают его на принтер.

## 30.6 Переменные, управляющие печатью в Postscript

Все команды печати через Postscript используют переменные `ps-lpr-command` и `ps-lpr-switches`, указывающие, как нужно печатать. `ps-lpr-command` задает имя запускаемой команды, `ps-lpr-switches` задает ключи командной строки, а `ps-printer-name`

задает принтер. Если вы не установили первые две переменные сами, они получают свои начальные значения от `lpr-command` и `lpr-switches`. Если `ps-printer-name` равна `nil`, используется `printer-name`.

Переменная `ps-print-header` контролирует, будут ли эти команды добавлять строки заголовка для каждой страницы, — установите ее равной `nil`, чтобы выключить заголовки. Вы можете отключить обработку цветов, установив `ps-print-color-p` в значение `nil`.

Переменная `ps-paper-type` указывает, для какого размера станицы нужно форматировать; допустимые значения включают `a4`, `a3`, `a4small`, `b4`, `b5`, `executive`, `ledger`, `legal`, `letter`, `letter-small`, `statement`, `tabloid`. По умолчанию это `letter`. Вы можете определить дополнительные размеры бумаги, изменяя переменную `ps-page-dimensions-database`.

Переменная `ps-landscape-mode` указывает ориентацию текста на странице. По умолчанию она равна `nil`, что обозначает "портретный" режим. Любое отличное от `nil` значение задает "ландшафтный" режим.

Переменная `ps-number-of-columns` задает число колонок; она играет роль и в "портретном", и в "ландшафтном" режиме. По умолчанию это 1.

Переменная `ps-font-family` указывает, какое семейство шрифтов нужно использовать при печати обычного текста. Допустимые значения включают `Courier`, `Helvetica`, `NewCenturySchlbk`, `Palatino` и `Times`. Переменная `ps-font-size` задает размер шрифта для обычного текста. По умолчанию это 8.5 пунктов.

Многие другие переменные для настройки этих команд определены и описаны в файле на Лиспе '`ps-print.el`'.

## 30.7 Сортировка текста

Emacs предоставляет несколько команд для сортировки текста в буфере. Все они оперируют с содержимым области (текстом между точкой и меткой). Эти команды разделяют текст области на большое число *записей сортировки*, определяют *ключ сортировки* для каждой записи и затем переставляют записи в порядке, определяемом ключами сортировки. Записи располагаются таким образом, чтобы их ключи находились в алфавитном или, для числовой сортировки, числовом порядке. При алфавитной сортировке все буквы верхнего регистра от 'A' до 'Z' идут перед 'a' нижнего регистра, в соответствии с последовательностью знаков ASCII.

Различие команд сортировки состоит в том, как они делят текст на записи сортировки, и какая часть каждой записи используется в качестве ключа сортировки. Большинство команд считают каждую строку отдельной записью, но некоторые используют в качестве таких записей абзацы или страницы. Большинство команд сортировки используют всю запись сортировки в качестве своего собственного ключа, но некоторые используют в качестве ключа сортировки только часть записи.

`M-x sort-lines`
: Разделить область на строки и отсортировать в соответствии с полным текстом строки. Числовой аргумент означает сортировку по убыванию.

`M-x sort-paragraphs`
: Разделить область на абзацы и отсортировать, сравнивая текст абзацев целиком (за исключением пустых строк в начале). Числовой аргумент означает сортировку по убыванию.

`M-x sort-pages`
: Разделить область на страницы и отсортировать, сравнивая полный текст страниц (за исключением пустых строк в начале). Числовой аргумент означает, что сортировка производится по убыванию.

`M-x sort-fields`
: Разделить область на строки и отсортировать, сравнивая содержимое одного поля в каждой строке. Поля разделяются пропусками, так что первый отрезок последовательных непробельных знаков в строке составляет поле 1, второй такой отрезок составляет поле 2, и так далее.

  Поле, по которому должна производится сортировка, указывается с помощью числового аргумента: 1 используется для сортировки по полю 1, и так далее. Отрицательный аргумент означает отсчет полей справа, а не слева; то есть минус 1 обозначает сортировку по последнему полю. Если содержание полей сортировки в нескольких строках одинаково, эти строки остаются в том же относительном порядке, в каком они были изначально.

`M-x sort-numeric-fields`
: Эта команда подобна `M-x sort-fields`, за исключением того, что для каждой строки указанное поле превращается в число, и сравниваются уже эти числа. ‘`10`’ предшествует ‘`2`’, когда рассматривается как текст, но следует за ‘`2`’, когда рассматривается как число.

`M-x sort-columns`
: Как `M-x sort-fields`, за исключением того, что используемый для сравнения текст получается в пределах каждой строки из фиксированного диапазона столбцов. Объяснение приведено ниже.

`M-x reverse-region`
: Обратить порядок строк в области. Это полезно для сортировки в порядке убывания по полям или колонкам, так как данные команды сортировки не имеют средств для этого.

Например, если буфер содержит такой текст:

```
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.
```

применение `M-x sort-lines` ко всему буферу даст следующее:

```
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
```

где заглавная ‘`O`’ идет перед всеми строчными буквами. Если вы вместо этого примените `C-u 2 M-x sort-fields`, то получите следующее:

```
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
```

где ключами сортировки были ‘`Emacs`’, ‘`If`’, ‘`buffer`’, ‘`systems`’ и ‘`the`’.

`M-x sort-columns` требует более подробного объяснения. Вы указываете колонки, устанавливая точку на одном столбце, а метку — на другом. Так как это означает, что вы не можете поставить точку или метку в начале первой строки, с которой должна начаться сортировка, то эта команда использует необычное определение ‘области’: вся строка с точкой рассматривается как часть области, и так же все содержимое строки, в которой находится метка, и все строки между ними.

Например, для сортировки таблицы по информации, размещенной в столбцах с 10 по 15, метку можно поставить в первой строке таблицы в столбце 10, а точку в столбце 15 в последней строке таблицы и затем запустить `sort-columns`. Или вы можете поставить метку в столбце 15 в первой строке, а точку — в столбце 10 в последней строке.

Это можно рассматривать как сортировку прямоугольника, заданного точкой и меткой, за исключением того, что текст в каждой строке справа и слева от прямоугольника двигается вместе с текстом внутри прямоугольника. См. Раздел 9.4 [Прямоугольники], с. 74.

Многие команды сортировки игнорируют при сравнениях различие в регистре букв, если `sort-fold-case` не равна `nil`.

## 30.8 Сужение

*Сужение* означает сосредоточение на некоторой части буфера, при котором оставшаяся часть становится временно недоступной. Та часть, в которую вы все еще можете попасть, называется *доступной частью*. Отмена сужения и переход в состояние, когда весь буфер снова становится доступным, называется *расширением*. Размер действующего сужения буфера в какое-либо время называется *ограничением* этого буфера.

Сужение может помочь сконцентрироваться на единственной подпрограмме или абзаце, убирая лишнее. Оно также может быть использовано для ограничения диапазона действия команды замены или повторяющегося макроса клавиатуры.

`C-x n n` Сузить до области между точкой и меткой (`narrow-to-region`).

`C-x n w` Расширить так, чтобы буфер снова стал полностью доступным (`widen`).

`C-x n p` Сузить до текущей страницы (`narrow-to-page`).

`C-x n d` Сузить до текущего определения функции (`narrow-to-defun`).

Когда вы сделали сужение до части буфера, эта часть — все, что вы видите. Вы не можете видеть остальную часть, в нее невозможно перейти (команды движения не выйдут за пределы доступной части) и нельзя изменить ее никаким образом. Но она не пропала, и если вы сохраняете файл, то сохранится и недоступная часть текста. При действии сужения в строке режима появляется слово ‘`Narrow`’.

Основной командой сужения является `C-x n n` (`narrow-to-region`). Она устанавливает ограничения для текущего буфера таким образом, что остается доступным только текст текущей области, но весь текст перед областью и после нее становится недоступным. Точка и метка не изменяются.

Еще вы можете использовать `C-x n p` (`narrow-to-page`) для сужения до текущей страницы. См. Раздел 21.4 [Страницы], с. 184, определение страницы. `C-x n d` (`narrow-to-defun`) сужает до определения функции, содержащего точку (см. Раздел 22.4 [Определения функций], с. 208).

Отмена сужения выполняется командой `C-x n w` (`widen`). Она делает весь текст буфера снова доступным.

Вы можете получить информацию о том, до какой части сужен буфер, применив команду `C-x =`. См. Раздел 4.9 [Информация о позиции], с. 40.

Поскольку сужение может легко запутать пользователя, не понимающего его, команда `narrow-to-region` обычно заблокирована. При попытке ее использования запрашивается подтверждение и предоставляется возможность ее включения; если вы задействовали эту команду, подтверждение больше не требуется. См. Раздел 31.4.11 [Блокирование команды], с. 364.

## 30.9 Редактирование текста в две колонки

Режим Two-column позволяет вам удобно редактировать текст в две колонки. Он использует два расположенных рядом окна, каждое из которых показывает свой буфер.

Есть два способа войти в двухколоночный режим:

⟨F2⟩ 2 или C-x 6 2
: Входит в режим two-column, слева появляется текущий буфер, а справа появляется буфер, чье имя основано на имени текущего буфера (2C-two-columns). Если правый буфер еще не существует, изначально он делается пустым; содержимое текущего буфера не изменяется.

  Эта команда подходит, если текущий буфер пуст или содержит текст только одной колонки, и вы хотите добавить вторую.

⟨F2⟩ s or C-x 6 s
: Разделяет текущий буфер с текстом в двух колонках на два буфера и показывает их рядом друг с другом (2C-split). Текущий буфер становится левым буфером, но текст из правой колонки перемещается в правый буфер. Текущий столбец определяет точку раздела. Разделение начинается от текущей строки и продолжается до конца буфера.

  Эта команда полезна, если у вас уже есть буфер с двухколоночным текстом, и вы хотите временно разделить колонки.

⟨F2⟩ b *буфер* ⟨RET⟩
C-x 6 b *буфер* ⟨RET⟩
: Входит в режим two-column, используя текущий буфер в качестве левого и буфер *буфер* в качестве правого (2C-associate-buffer).

⟨F2⟩ s или C-x 6 s ищет разделитель колонок, который является строкой, стоящей между двух колонок на каждой строке. Вы можете задать ширину разделителя с помощью числового аргумента для ⟨F2⟩ s; столько знаков, стоящих перед точкой, выразят строку-разделитель. По умолчанию ширина равна 1, поэтому разделитель колонок — это знак перед точкой.

Когда строка содержит разделитель на своем месте, ⟨F2⟩ s помещает текст после разделителя в правый буфер и удаляет разделитель. Строки, не имеющие разделителя колонок, оставляются неразбитыми; они остаются в левом буфере, а в правый буфер помещается для соответствия пустая строка. (Таким образом можно написать строку, которая “проходит по обеим колонкам в двухколоночном режиме”: написать ее в левом буфере и сделать пустую строку в правом буфере.)

Команда C-x 6 ⟨RET⟩ или ⟨F2⟩ ⟨RET⟩ (2C-newline) вставляет знак новой строки в оба буфера в соответствующих позициях. Это самый простой способ добавить новую строку в двухколоночном тексте, когда вы редактируете его в разных буферах.

Когда вы отредактировали оба буфера, как вам хотелось, объедините их с помощью ⟨F2⟩ 1 или C-x 6 1 (2C-merge). Это копирует текст из правого буфера как вторую колонку другого буфера. Чтобы вернуться к редактированию в две колонки, используйте ⟨F2⟩ s.

Используйте ⟨F2⟩ d или C-x 6 d, чтобы разъединить два буфера, оставляя каждый как есть (2C-dissociate). Если другой буфер, не текущий в момент, когда вы набрали ⟨F2⟩ d, оказался пустым, ⟨F2⟩ d уничтожает его.

## 30.10 Редактирование двоичных файлов

Существует специальный основной режим для редактирования двоичных файлов: режим Hexl. Чтобы воспользоваться им, вызовите для обращения к файлу M-x hexl-find-file вместо C-x C-f. Эта команда преобразует содержимое файла в

шестнадцатиричные числа и позволяет вам редактировать их. Когда вы сохраняете этот файл, он автоматически преобразуется обратно в двоичный формат.

Вы можете также использовать `M-x hexl-mode` для преобразования существующего буфера в шестнадцатиричный вид. Это полезно, если вы обратились к файлу обычным способом и потом обнаружили, что это двоичный файл.

Обычные знаки в режиме Hexl перезаписывают поверх существующего текста. Это сделано для уменьшения риска нечаянной потери выравнивания данных в файле. Для вставки есть особые команды. Вот перечень команд режима Hexl:

`C-M-d` Вставляет байт с введенным десятичным кодом.

`C-M-o` Вставляет байт с введенным восьмиричным кодом.

`C-M-x` Вставляет байт с введенным шестнадцатиричным кодом.

`C-x [` Перемещает на начало 1k-байтной “страницы”.

`C-x ]` Перемещает на конец 1k-байтной “страницы”.

`M-g` Перемещает к адресу, заданному шестнадцатиричным числом.

`M-j` Перемещает к адресу, заданному десятичным числом.

`C-c C-c` Покидает режим Hexl, возвращаясь к тому основному режиму, который был в этом буфере до того, как вы вызвали `hexl-mode`.

## 30.11 Сохранение сеансов Emacs

Вы можете использовать библиотеку Desktop для сохранения состояния Emacs от одного сеанса к другому. Сохранение состояния означает, что Emacs запускается с тем же самым набором буферов, основных режимов, позиций в буферах и прочим, какие были в предыдущем сеансе Emacs.

Чтобы использовать Desktop, вы должны воспользоваться буфером Customization (см. Раздел 31.2.2 [Простая настройка], с. 344) и установить `desktop-enable` в отличное от `nil` значение или добавить такие строки в ваш файл ‘`.emacs`’:

```
(desktop-load-default)
(desktop-read)
```

Первый раз, когда вы сохраняете состояние сеанса Emacs, вы должны сделать это вручную, с помощью команды `M-x desktop-save`. Как только вы это сделали, при выходе из Emacs состояние будет сохраняться снова — не только в существующем сеансе Emacs, но и во всех последующих. Вы также можете записать состояние в любое время не выходя из Emacs, снова набрав `M-x desktop-save`.

Чтобы Emacs восстановил состояние предыдущего сеанса, вы обязаны запустить его из того же каталога, который вы использовали в прошлом сеансе. Это необходимо, потому что `desktop-read` просматривает в поисках файла, который она должна прочитать, именно текущий каталог. Это означает, что вы можете хранить отдельные сохраненные сеансы в разных каталогах; каталог, в котором вы запускаете Emacs, будет указывать, какой из сохраненных сеансов вы хотите восстановить.

Переменная `desktop-files-not-to-save` говорит, какие файлы исключаются при сохранении состояния. Ее значение — это регулярное выражение, совпадающее с именами исключаемых файлов. По умолчанию исключаются удаленные (полученные по FTP) файлы; потому что повторное к ним обращение в последующем сеансе может быть медленным. Если вы хотите включить эти файлы в сохраненное состояние, установите `desktop-files-not-to-save` равной `"^$"`. См. Раздел 14.12 [Удаленные файлы], с. 134.

## 30.12 Уровни рекурсивного редактирования

*Рекурсивное редактирование* — это ситуация, в которой вы используете команды Emacs для выполнения произвольного редактирования, находясь в середине другой команды Emacs. Например, при наборе команды `C-r` внутри `query-replace` происходит вход в рекурсивное редактирование, где можно изменить текущий буфер. Выходя из этого рекурсивного редактирования, вы возвращаетесь в `query-replace`.

*Выход* из рекурсивного редактирования означает возврат к незаконченной команде, которая продолжает выполняться. Выход производится с помощью команды `C-M-c` (`exit-recursive-edit`).

Вы можете также *прервать* рекурсивное редактирование. Это похоже на выход, но при этом происходит также и незамедлительный выход из незаконченной команды. Прерывание рекурсивного редактирования производится по команде `C-]` (`abort-recursive-edit`). См. Раздел 32.1 [Выход], с. 371.

Строка режима показывает вам, что вы находитесь в рекурсивном редактировании, при помощи квадратных скобок вокруг круглых скобок, которые всегда окружают названия основного и второстепенных режимов. Строка режима каждого окна показывает это таким же образом, поскольку нахождение в рекурсивном редактировании относится к Emacs в целом, а не к какому-либо отдельному окну или буферу.

Возможно пребывание в рекурсивных редактированиях внутри рекурсивных редактирований. Например, после набора команды `C-r` в `query-replace` можно набрать команду, которая входит в отладчик. Это начинает уровень рекурсивного редактирования для отладчика внутри уровня рекурсивного редактирования для `C-r`. Строки режима показывают пару квадратных скобок для каждого работающего в данный момент уровня рекурсивного редактирования.

Выход из внутреннего рекурсивного редактирования (например, с помощью команды `c` отладчика) возобновляет команду одним уровнем выше. По окончанию этой команды можно использовать `C-M-c`, чтобы выйти из еще одного уровня рекурсивного редактирования, и так далее. Выход относится только к самому внутреннему уровню. Прерывание тоже приводит к выходу лишь из одного уровня рекурсивного редактирования: происходит немедленный возврат на уровень команды предыдущего рекурсивного редактирования. Если хотите, то можете затем прервать и следующий уровень.

Напротив, команда `M-x top-level` прерывает все уровни рекурсивного редактирования, возвращаясь непосредственно на верхний командный уровень.

Текст, редактируемый внутри режима рекурсивного редактирования, не обязан быть тем же, что редактируется на верхнем уровне. Это зависит от того, для чего предназначалось рекурсивное редактирование. Если команда, которая запускает рекурсивное редактирование, выбирает вначале другой буфер, то он и будет буфером, который рекурсивно редактируется. В любом случае, внутри рекурсивного редактирования можно переключать буферы обычным способом (если ключи, переключающие буферы, не перепривязаны). Вероятно, можно оставаться внутри рекурсивного редактирования и до конца сеанса, обращаясь к файлам и делая там все прочее. Но время от времени это может приводить к неожиданным результатам (например, к переполнению стека). Поэтому не забывайте прерывать рекурсивное редактирование или выходить из него, если оно вам больше не нужно.

В основном мы стараемся минимизировать использование уровней рекурсивного редактирования в GNU Emacs. Это так, потому что они обязывают вас “возвращаться” в определенном порядке — от самого внутреннего уровня к верхнему уровню. Если возможно, мы представляем различные виды деятельности в разных буферах, чтобы вы могли переключаться между ними, как вам нравится. Некоторые команды переключают в новый основной режим, который предоставляет команду для переключения назад. Эти методы

дают вам большую гибкость для возврата к незаконченным занятиям в том порядке, в каком вы пожелаете.

## 30.13 Эмуляция

GNU Emacs может быть запрограммирован для эмуляции (в большей или меньшей степени) большинства других редакторов. Стандартные средства могут эмулировать следующее:

EDT (редактор DEC VMS)
: Эмуляция EDT включается с помощью команды `M-x edt-emulation-on`. Команда `M-x edt-emulation-off` восстанавливает обычные для Emacs привязки.

  Большинство команд эмуляции EDT являются ключами вспомогательной клавиатуры, и большинство стандартных привязок ключей в Emacs остаются доступными. Переприязки при эмуляции EDT выполняются в глобальной таблице ключей, таким образом, при нахождении в режиме эмуляции EDT нет проблемы переключения буферов или основных режимов.

vi (редактор Беркли)
: Viper — новейший эмулятор vi. Он реализует несколько уровней эмуляции: уровень 1 ближе всех к vi, тогда как уровень 5 отходит от строгой эмуляции, чтобы воспользоваться возможностями Emacs. Чтобы вызвать Viper, наберите `M-x viper-mode`; это проведет вас по остальному пути и спросит об уровне эмуляции. См. Info файл ‘`viper`’, node ‘`Top`’.

vi (другой эмулятор)
: `M-x vi-mode` входит в основной режим, который заменяет прежде установленный режим. Все команды vi, которые в настоящем vi входят в режим “ввода”, запрограммированы для возврата в предыдущий основной режим. Таким образом, обычный Emacs служит режимом “ввода” для vi.

  Поскольку эмуляция vi работает через основные режимы, переключать буфера в процессе эмуляции нельзя. Сначала необходимо вернуться в обычный Emacs.

  Если вы планируете часто использовать эмуляцию vi, то, вероятно, появится желание привязать ключ к команде `vi-mode`.

vi (еще один эмулятор)
: `M-x vip-mode` вызывает еще один эмулятор vi, про который говорят, что он соответствует настоящему vi более полно, чем `M-x vi-mode`. Режим “ввода” в этом эмуляторе отличается от обычного Emacs, так что для возврата в режим эмуляции vi можно использовать ⟨ESC⟩. Для возврата из режима эмуляции vi в обычный Emacs необходимо набрать `C-z`.

  Этот режим эмуляции не работает через основные режимы, что обеспечивает возможность различных вариантов переключения буферов внутри эмулятора. Приписывать ключ команде `vip-mode` нет так необходимо, как в случае `vi-mode`, поскольку завершение режима вставки не использует ее.

  См. Info файл ‘`vip`’, node ‘`Top`’, для получения полной информации.

## 30.14 Диссошиэйтед Пресс

`M-x dissociated-press` — это команда для перемешивания текстового файла слово за словом или знак за знаком. Имея в начале буфер с нормальным текстом, она формирует крайне забавный вывод. Ввод производится из текущего буфера Emacs. Диссошиэйтед Пресс записывает свой вывод в буфер с именем ‘`*Dissociation*`’, при этом, чтобы облегчить его постепенное чтение, через каждую пару строк (примерно) содержимое буфера показывается заново.

Диссошиэйтед Пресс время от времени спрашивает, продолжать ли действие. Для остановки необходимо ответить `n`. Остановить можно также в любое время с помощью `C-g`. Диссоциированная выдача сохраняется в буфере '`*Dissosiation*`', чтобы по желанию можно было скопировать ее в другое место.

Диссошиэйтед Пресс в процессе работы совершает беспорядочные прыжки из одной точки буфера в другую. Для получения правдоподобного вывода, а не тарабарщины, она соблюдает некоторое перекрытие между концом одного отрезка последовательности слов или знаков и началом следующего. Так, если только что она напечатала слово 'президент' и теперь решает прыгнуть в другую точку файла, то она может заметить 'ент' в слове 'пентагон' и продолжить вывод отсюда, выдавая в результате 'президентагон'.[2] Наилучшие результаты получаются на длинных выборках.

Положительный аргумент `M-x dissociated-press` велит ей работать познаково и определяет число перекрывающихся знаков. Отрицательный аргумент заставляет ее действовать слово за словом и определяет количество перекрывающихся слов. В этом режиме целые слова, а не знаки, трактуются как переставляемые элементы. Отсутствие аргумента эквивалентно аргументу, равному двум. К сновашему сведению, вывод осуществляется только в буфер '`*Dissociation*`'. Буфер, с которого вы начали, не изменяется.

Диссошиэйтед Пресс производит примерно те же результаты, что и марковская цепь, основанная на частотной таблице, построенной по выборочному тексту. Однако, этот метод является независимым, игнориганальным изобретением. Диссошиэйтед Пресс повсеметодически копирует несколько последовательных знаков из выборки от прыжка к прыжку, тогда как цепь Маркова делала бы случайный выбор для каждого слова или знака. В итоге все это работает быстрее и придает результату более благозвучный вид.

Несомнекоторые говорят, что излишне интенсивное использование Диссошиэйтед Пресс может стать постомехой в вашей реальной работе. Подчас до уровня безобрагедии. И избегайте диссошислов в своей документации, если вы хотите быть для пользователей вполнятными и правословными. Позабавьтесь. Ваши бредложения горячо приветствуются.

## 30.15 Другие развлечения

Если вы немного заскучали, можете попробовать `M-x hanoi`. Если вам очень скучно, то задайте ей численный аргумент. Если вам очень-очень скучно, то попробуйте задать аргумент 9. Откиньтесь на спинку кресла и наблюдайте.

Если вам хочется больше личного участия, попробуйте команду `M-x gomoku`, которая сыграет с вами в пять-в-ряд.

`M-x blackbox` и `M-x mpuz` — это две головоломки. `blackbox` предлагает вам определить с помощью томографии положение объектов внутри черного ящика. `mpuz` показывает задачу на умножение, где цифры заменены буквами, а как, вы должны догадаться. Чтобы сделать предположение, наберите букву, а затем цифру, которая, как вы думаете, обозначена этой буквой.

`M-x dunnet` запускает приключенческую игру; это большая головоломка.

Если вас расстроили, запустите знаменитую программу Eliza. Наберите просто `M-x doctor`. Каждый ввод заканчивайте двойным набором ⟨RET⟩.

Когда вам будет не по себе, наберите `M-x yow`.

[2] Это диссошислово действительно возникло во время войны во Вьетнаме, когда оно было очень актуально.

# 31 Настройка

В этой главе обсуждаются различные вопросы, относящиеся к простой адаптации поведения Emacs. Чтобы узнать, как сделать большие изменения, смотрите книгу *The Emacs Lisp Reference Manual*.

Все виды настройки воздействуют только на тот сеанс Emacs, в котором вы их делаете. Они полностью исчезают при завершении работы с Emacs и не действуют на другие сеансы Emacs, которые могут быть запущены в то же самое время или позже. Только в одном случае сеанс работы с Emacs может повлиять на что-либо вне его самого — при записи файла. В частности, чтобы сделать настройку "постоянной", существует единственный путь — поместить нечто в ваш файл '`.emacs`' или другой подходящий файл, что будет выполнять настройку в каждом сеансе. См. Раздел 31.7 [Файл инициализации], с. 366.

## 31.1 Второстепенные режимы

Второстепенные режимы — это необязательные возможности, которые вы можете включать и выключать. Например, режим Auto Fill — это второстепенный режим, в котором ⟨SPC⟩ разрывает строки на границе слов по мере того, как вы набираете. Все второстепенные режимы независимы друг от друга и от выбранного основного режима. Большинство второстепенных режимов сообщают, что они включены, в строке режима; например, надпись '`Fill`' в строке режима означает, что включен режим Auto Fill.

Для получения имени командной функции, включающей или выключающей второстепенный режим, добавьте к имени второстепенного режима слово `-mode`. Таким образом, команда запуска или выключения режима Auto Fill называется `M-x auto-fill-mode`. Подобные команды обычно запускаются через `M-x`, но при желании к ним можно привязать ключи. Без аргумента эти команды включают режим, если он был выключен, и выключают, когда он был включен. Эта техника известна как *переключение*. Положительный аргумент всегда включает режим, а явный нулевой или отрицательный аргумент всегда выключает его.

Включение или выключение некоторых второстепенных режимов применяется только к текущему буферу; каждый буфер независим от других. Следовательно, вы можете включить режим в одних буферах и выключить в других. К второстепенным режимам, которые могут так работать, относятся режим Abbrev, режим Auto Fill, режим Auto Save, режим Font-Lock, режим Hscroll, режим ISO Accents, второстепенный режим Outline, режим Overwrite и режим Binary Overwrite.

Режим Abbrev позволяет вам определить сокращения, которые автоматически расшифровываются при наборе. Например, '`amd`' может раскрываться в '`abbrev mode`'. См. Глава 24 [Сокращения], с. 257, для получения полной информации.

Режим Auto Fill позволяет вводить заполненный текст без явного прерывания строк. Emacs вставляет перевод строки, когда это нужно, чтобы строка не стала слишком длинной. См. Раздел 21.5 [Заполнение], с. 185.

Режим Auto Save заставляет Emacs периодически сохранять содержимое буфера, чтобы уменьшить возможный объем потерянной работы в случае краха системы. См. Раздел 14.5 [Самосохранение], с. 114.

Режим Enriched позволяет редактировать и сохранять форматированный текст. См. Раздел 21.11 [Форматированный текст], с. 198.

Режим Flyspell автоматически подсвечивает неправильно набранные слова. См. Раздел 13.4 [Правописание], с. 102.

Режим Font-Lock автоматически подсвечивает определенные текстовые единицы, используемые в программах, такие как комментарии, строки и имена определенных функций. Для этого требуется оконная система, которая может отображать разные шрифты. См. Раздел 17.13 [Начертания], с. 155.

Режим Hscroll выполняет автоматическую горизонтальную прокрутку экрана, чтобы точка всегда была видна. См. Раздел 11.2 [Горизонтальная прокрутка], с. 82.

Режим ISO Accents компонует знаки ‘`’, ‘'’, ‘"’, ‘^’, ‘/’ и ‘~’ со следующий буквой в букву с акцентом из набора знаков ISO Latin-1. См. Раздел 18.12 [Поддержка западноевропейских алфавитов], с. 172.

Второстепенный режим Outline обеспечивает те же возможности, что и основной режим, называемый Outline; но поскольку он является второстепенным режимом, то вы можете использовать его вместе с любым основным. См. Раздел 21.8 [Режим Outline], с. 190.

Режим Overwrite заменяет существующие знаки при вводе новых вместо обычной вставки со сдвигом вправо. Например, если точка находится перед ‘`B`’ в слове ‘`FOOBAR`’, то в режиме Overwrite ввод `G` изменяет это слово на ‘`FOOGAR`’, вместо ‘`FOOGBAR`’, как это делается обычно. В режиме Overwrite, команда `C-q` вставляет знак, каким бы он не был, даже если это цифра, — это дает вам способ вставки знака вместо замены существующего.

Режим Binary Overwrite — это вариант режима Overwrite для редактирования двоичных файлов; он обрабатывает знаки новой строки и табуляции точно также, как и другие знаки, так что они могут заменять другие знаки и сами могут быть заменены другими знаками.

Следующие второстепенные режимы обычно применяются ко всем буферам сразу. Поскольку каждый из них включается или выключается с помощью установки значения переменной, вы *можете* установить их по-разному для отдельных буферов, явно делая эти переменные локальными для этих буферов. См. Раздел 31.2.4 [Локальные переменные], с. 350.

Режим Icomplete показывает возможность завершения, когда вы находитесь в минибуфере и завершение активно. См. Раздел 5.3.4 [Параметры завершения], с. 49.

Режим Line Number постоянно отображает номер текущей строки в строке режима. См. Раздел 1.3 [Строка режима], с. 25.

Режим Resize-Minibuffer заставляет минибуфер расширяться так, чтобы он мог вместить весь текст, который вы в него помещаете. См. Раздел 5.2 [Редактирование в минибуфере], с. 46.

Режим Scroll Bar прикрепляет к каждому окну полоску прокрутки (см. Раздел 17.11 [Полоски прокрутки], с. 154). Режим Menu Bar прикрепляет к каждому фрейму полоску меню (см. Раздел 17.12 [Полоски меню], с. 155). Оба этих режима по умолчанию включены при использовании X Window System.

В режиме Transient Mark, каждое изменение содержимого буфера “деактивирует” метку, так что команды, которые оперируют с областью, буду выдавать ошибку. Это означает, что вы должны либо установить пометку, либо явно “активировать” ее перед каждой командой, которая работает с областью. Преимущество режима Transient Mark в том, что Emacs может отрисовывать область подсвеченной (в настоящее время только при работе в X). См. Раздел 8.1 [Установка метки], с. 63.

Для большинства второстепенных режимов, имя команды также совпадает с именем переменной, которая контролирует данный режим. Режим включается, когда значение переменной устанавливается в не-`nil`, и команда установки второстепенного режима работает путем установки этой переменной. Например, команда `outline-minor-mode` работает, устанавливая значение `outline-minor-mode` как переменной; именно эта переменная непосредственно включает и выключает второстепенный режим Outline. Чтобы проверить, работает ли некий второстепенный режим таким способом, используйте `C-h v`, чтобы запросить документацию на имя переменной.

Переменные второстепенных режимов обеспечивают хороший способ включения и выключения второстепенных режимов для программ на Лиспе; они также полезны в списках локальных переменных файлов. Но пожалуйста, дважды подумайте, прежде чем устанавливать второстепенные режимы через список локальных переменных, потому что большинство второстепенных режимов являются предметом предпочтений пользователя — другие

пользователи, которые будут редактировать этот же файл, могут не захотеть использовать те же режимы, что предпочитаете вы.

## 31.2 Переменные

*Переменная* — это лисповский символ, который имеет значение. Имя символа также называется именем соответствующей переменной. Имена переменных могут содержать любые знаки, которые могут появиться в файле, но принято составлять их из слов, разделенных дефисами. Переменная может иметь строку документации, которая описывает, значение какого вида она должна иметь, и как это значение будет использоваться.

Лисп позволяет хранить в любой переменной любой вид значения, но большинство используемых в Emacs переменных требуют значение определенного типа. Зачастую значение должно всегда быть строкой или всегда числом. Иногда мы говорим, что некоторое свойство включено, если некая переменная "отлична от `nil`", подразумевая, что если значение переменной равно `nil`, то это свойство выключено, но оно включено для *любого* другого значения. Поскольку при установке переменной вам нужно выбрать одно определенное значение, для включения свойства принято использовать значение `t`.

Как и всякая программа на Лиспе, Emacs использует много лисповских переменных для хранения внутренних записей. Но большинство переменных, которые представляют для вас интерес, — это те, что существуют ради настройки. Emacs (обычно) не меняет значения этих переменных, вместо этого вы сами устанавливаете значения, тем самым изменяя и управляя поведением некоторых команд Emacs. Эти переменные называются *пользовательскими параметрами*. Большинство параметров описаны в этом руководстве и приведены в указателе переменных (см. [Указатель переменных], с. 459).

Примером переменной, являющейся пользовательским параметром, служит `fill-column`; она определяет позицию правого края (как число знаков от левого края), которая используется командами заполнения (см. Раздел 21.5 [Заполнение], с. 185).

### 31.2.1 Просмотр и установка переменных

`C-h v` *пер* ⟨RET⟩
: Показывает значение переменной *пер* и документацию по ней (`describe-variable`).

`M-x set-variable` ⟨RET⟩ *пер* ⟨RET⟩ *значение* ⟨RET⟩
: Изменяет значение переменной *пер* на *значение*.

Чтобы посмотреть значение отдельной переменной, используется команда `C-h v` (`describe-variable`), которая считывает в минибуфере имя переменной с возможностью завершения. Эта команда печатает и значение переменной, и документацию по ней. Например,

```
C-h v fill-column ⟨RET⟩
```

выведет примерно следующее:

```
fill-column's value is 75

Documentation:
*Column beyond which automatic line-wrapping should happen.
Automatically becomes buffer-local when set in any fashion.
```

Звездочка в начале описания показывает, что эта переменная является пользовательским параметром. `C-h v` не ограничивается только пользовательскими параметрами, она принимает имя любой переменной.

Наиболее удобный способ установить конкретный параметр — выполнить `M-x set-variable`. Эта команда считывает имя переменной с помощью минибуфера (с завершением), а затем считывает лисповское выражение для нового значения, снова используя минибуфер. Например,

```
M-x set-variable ⟨RET⟩ fill-column ⟨RET⟩ 75 ⟨RET⟩
```

устанавливает `fill-column` равной 75.

Действие `M-x set-variable` ограничено пользовательскими параметрами, но вы можете установить значение любой переменной с помощью выражения на языке Лисп, используя функцию `setq`. Вот выражение для установки значения переменной `fill-column` через `setq`:

```
(setq fill-column 75)
```

Чтобы выполнить выражение, подобное этому, переключитесь в буфер '`*scratch*`', наберите выражение и затем нажмите `C-j`. См. Раздел 23.9 [Диалог с Лиспом], с. 255.

Установка переменных влияет только на текущий сеанс Emacs, так же, как и все остальные способы настройки Emacs, за исключением случаев, когда явно сказано иное.

### 31.2.2 Интерфейс для простой настройки

Удобный способ найти желаемые параметры и изменить их предоставляет команда `M-x customize`. Она создает *буфер настройки*, где вы можете просматривать пользовательские параметры Emacs, представленные логически организованной структурой, редактировать их и устанавливать их значения. Вы также можете использовать буфер настройки для постоянного сохранения этих установок. (Пока в эту структуру включены не все пользовательские параметры Emacs, но мы добавляем оставшиеся.)

#### 31.2.2.1 Группы настройки

В целях настройки пользовательские параметры собраны в *группы*, чтобы их было проще найти. Группы собраны в еще большие группы, и так до самой верхней группы, названной `Emacs`.

Команда `M-x customize` создает буфер настройки, который показывает группу верхнего уровня `Emacs` и группы, которые лежат на уровень ниже. Это выглядит примерно так, показана часть:

```
/- 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.

еще группы второго уровня

\- Emacs group end -------------------------------/
```

Это означает, что буфер показывает содержимое группы `Emacs`. Другие группы перечислены здесь, поскольку они являются ее содержимым. Но они перечислены иначе, без отступов

и тире, потому что *их* содержимое сюда не включено. Каждая группа имеет однострочное описание; у группы `Emacs` также есть строка '`[State]`'.

Большинство текста в буфере настройки находится в режиме только для чтения, но обычно в него включено несколько *редактируемых полей*, которые вы можете изменять. Существуют также *активные поля*; это означает, что они делают что-то, когда вы *активизируете* их. Для активизации активного поля либо щелкните на нем кнопкой `Mouse-1`, или установите на нем точку и нажмите ⟨RET⟩.

Например, фраза '`[Go to Group]`', появляющая в группе второго уровня, — активное поле. Активизация поля '`[Go to Group]`' для группы создает новый буфер настройки, который показывает эту группу и ее содержимое. Это поле является чем-то вроде гиперссылки на другую группу.

Сама группа `Emacs` не включает ни одного пользовательского параметра, но они есть в других группах. Исследуя разные группы, вы в конце концов найдете нужные вам параметры и начертания. Затем вы можете использовать буфер настройки, чтобы установить их.

Вы можете просмотреть структуру групп настройки в укрупненном виде, используя команду `M-x customize-browse`. Эта команда создает особый вид буфера настройки, который показывает только имена групп (а также параметры и начертания) и их структуру.

В этом буфере вы можете просматривать содержимое группы, активизируя кнопку '`[+]`'. Когда показывается содержимое группы, эта кнопка меняется на '`[-]`'; активизация этой кнопки прячет содержимое группы.

Имя каждой группы, параметра или начертания в этом буфере имеет активное поле, в котором написано '`[Group]`', '`[Option]`' или '`[Face]`'. При активизации этого активного поля создается обычный буфер настройки, показывающий только эту группу с ее содержимым, или только этот параметр или только это начертание. Таким способом устанавливают значения.

### 31.2.2.2 Изменение параметра

Здесь приведен пример того, как выглядят в буфере настройки пользовательский параметр:

```
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.
```

Текст, следующий за '`[Hide]`', — в нашем случае это '`30`' — показывает текущее значение параметра. Если вместо '`[Hide]`' вы видите '`[Show]`', то это означает, что значение скрыто; буфер настройки сначала скрывает значения, которые занимают несколько строк. Нажмите на '`[Show]`', чтобы открыть значение.

Строка, следующая за именем параметра, показывает *состояние настройки* для данного параметра: в вышеприведенном примере, она сообщает, что вы еще не изменили этот параметр. Слово '`[State]`' в начале строки является активным; вы можете получить меню разных операций, нажав на ней `Mouse-1` или ⟨RET⟩. Эти операции являются существенными для настройки переменной.

Строка после строки '`[State]`' показывает начало описания данного параметра. Если документация занимает несколько строк, то эта строка оканчивается кнопкой '`[More]`'; вы можете выбрать ее для того, чтобы посмотреть более полное описание.

Для того чтобы ввести новое значение для '`Kill Ring Max`', переместите точку к значению и отредактируйте его как обычный текст. Например, вы можете набрать `M-d` и ввести затем другое число.

Когда вы начинаете редактировать текст, вы увидите, что строка '`[State]`' изменилась, сообщая, что вы поменяли значение:

```
[State]: you have edited the value as text, but not set the option.
```

Изменение значения в действительности не устанавливает значение переменной. Для этого вы должны *установить* данный параметр. Чтобы сделать это, активизируйте кнопку '`[State]`' и выберите '`Set for Current Session`'.

Когда вы установите параметр, его состояние визуально изменится:

```
[State]: you have set this option, but not saved it for future sessions.
```

Не нужно беспокоиться о том, что вы указали недопустимое значение; при установке параметра значение проверяется, и недопустимое значение никогда не будет установлено.

При редактировании значения поля, которое является именем файла, каталога, команды или чем-нибудь еще, для чего определено завершение, вы можете нажимать `M-`⟨TAB⟩ (`widget-complete`), чтобы произвести завершение.

Некоторые параметры имеют небольшой фиксированный набор возможных значений. Эти параметры не позволяют вам редактировать значения как текст. Вместо этого перед значением появляется активное поле '`[Value Menu]`'; активизируйте это поле для изменения значения. Для логического значения "вкл/выкл" активное поле показывает надпись '`[Toggle]`', и оно переключает это значение. '`[Value Menu]`' и '`[Toggle]`' изменяют буфер; изменения вступают в силу, когда вы используете операцию '`Set for Current Session`'.

Некоторые параметры имеют значения со сложной структурой. Например, значение переменной `load-path` является списком каталогов. Здесь показано, как оно изображается в буфере настройки:

```
Load Path:
[INS] [DEL] [Current dir?]: /usr/local/share/emacs/20.3/site-lisp
[INS] [DEL] [Current dir?]: /usr/local/share/emacs/site-lisp
[INS] [DEL] [Current dir?]: /usr/local/share/emacs/20.3/leim
[INS] [DEL] [Current dir?]: /usr/local/share/emacs/20.3/lisp
[INS] [DEL] [Current dir?]: /build/emacs/e20/lisp
[INS] [DEL] [Current dir?]: /build/emacs/e20/lisp/gnus
[INS]
   [State]: this item has been changed outside the customization buffer.
List of directories to search for files to load....
```

Каждый каталог в этом списке изображается на отдельной строке, а каждая строка имеет несколько редактируемых или активных полей.

Вы можете изменять любое из имен каталогов. Для того чтобы удалить каталог из списка, выберите кнопку '`[DEL]`' в его строке. Для того чтобы вставить в список новый каталог, выберите кнопку '`[INS]`' в той точке, куда вы хотите вставить имя.

Вы также можете выбрать поле '`[Current dir?]`', чтобы переключиться между включением в путь конкретного указанного каталога или значения `nil`. (`nil` в пути поиска означает "попробовать текущий каталог").

Две специальные команды, ⟨TAB⟩ и `S-`⟨TAB⟩, полезны для перемещения по буферу настройки. ⟨TAB⟩ (`widget-forward`) перемещает вперед на следующее активное или редактируемое поле; `S-`⟨TAB⟩ (`widget-backward`) перемещает в обратном направлении на предыдущее активное или редактируемое поле.

Нажимая ⟨RET⟩ на редактируемом поле, вы также перемещаетесь на следующее поле, аналогично действию ⟨TAB⟩. Причина этого заключается в том, что люди часто нажимают ⟨RET⟩ по завершении редактирования поля. Если вам понадобится вставить в редактируемое поле перевод строки, используйте `C-o` или `C-q C-j`.

Установка параметра изменяет его значение в текущем сеансе Emacs; *сохранение* значения изменяет его и для будущих сеансов. Это работает путем записи кода в ваш файл '`~/.emacs`', так что значения параметров будут устанавливаться каждый раз, когда вы

запускаете Emacs. Для того чтобы сохранить параметр, активизируйте кнопку '`[State]`' и выберите операцию '`Save for Future Sessions`'.

Вы также можете восстановить стандартные значения параметров, активизируя кнопку '`[State]`' и выбирая операцию '`Reset to Standard Settings`'. В действительности существует три стандартных операции восстановления:

'`Reset`' Если вы внесли некоторые изменения и не установили параметр, то эта операция восстанавливает текст буфера настройки, чтобы он соответствал текущему значению.

'`Reset to Saved`'
Эта операция восстанавливает значение параметра в последнее сохраненное значение и соответственно обновляет текст.

'`Reset to Standard Settings`'
Эта операция устанавливает параметр в его стандартное значение и соответственно обновляет текст. Эта операция также уничтожает любое сохраненное значение для данного параметра, так что в будущих сеансах работы с Emacs вы будете получать стандартное значение.

Состояние группы показывает, было ли что-нибудь в этой группе изменено, установлено или сохранено. Вы можете выбрать операции '`Set for Current Session`', '`Save for Future Sessions`' и различные виды операции '`Reset`' для данной группы; эти операции над группой применяются сразу ко всем настройкам в группе и ее подгруппах.

В начале буфера настройки находятся две строки, содержащие несколько активных полей:

```
[Set for Current Session] [Save for Future Sessions]
[Reset] [Reset to Saved] [Reset to Standard]   [Bury Buffer]
```

Активизация кнопки '`[Bury Buffer]`' скрывает буфер настройки. Каждое из остальных полей выполняет операции — установку, сохранение или восстановление — над каждым из пунктов в буфере, которые могут быть установлены, сохранены или восстановлены.

### 31.2.2.3 Настройка начертаний

Помимо пользовательских параметров, некоторые группы настройки также включают в себя начертания. Когда вы просматриваете содержимое группы, в буфере настройки показываются и пользовательские параметры, и начертания из этой группы. Начертание может выглядеть так:

```
Custom Changed Face: (sample)
   [State]: this face is unchanged from its standard setting.
Face used when the customize item has been changed.
Attributes: [ ] Bold: [toggle] off
            [X] Italic: [toggle] on
            [ ] Underline: [toggle] off
            [ ] Inverse-Video: [toggle] on
            [ ] Foreground: black (sample)
            [ ] Background: white (sample)
            [ ] Stipple:
```

Каждый атрибут начертания располагается на отдельной строке. Поле '`[x]`' перед именем атрибута показывает, *включен* ли этот атрибут; знак '`X`' обозначает, что включен. Вы можете включать или выключать атрибут, выбирая данное поле. Когда атрибут включен, вы можете изменить его значение обычными способами.

На черно-белом дисплее вам доступны для использования в качестве фона следующие цвета: ‘`black`’, ‘`white`’, ‘`gray`’, ‘`gray1`’ и ‘`gray3`’. Emacs поддерживает эти оттенки серого, используя вместо цвета штрихование фона с помощью масок.

Установка, сохранение и сброс начертания работает точно также, как и с пользовательскими параметрами (см. Раздел 31.2.2.2 [Изменение параметра], с. 345).

Начертание может задавать разный вид для разных типов дисплеев. Например, начертание может сделать текст красным на цветном дисплее, а на монохромном отображать этот текст жирным шрифтом. Для того чтобы указать разный вид для данного начертания, выберите пункт ‘`Show Display Types`’ из меню, которое появляется при выборе кнопки ‘`[State]`’.

Другой, более общий способ установить атрибуты заданного начертания — использовать команду `M-x modify-face`. Эта команда считывает имя начертания и атрибуты, один за другим. Для атрибутов, задающих цвета и маски, текущим значением атрибута будет значение по умолчанию — просто нажмите ⟨RET⟩, если вы не хотите изменять этот атрибут. Наберите ‘`none`’ в том случае, когда вы хотите очистить данный атрибут.

### 31.2.2.4 Настройка отдельных пунктов

Вместо того чтобы находить параметр, который вы хотите изменить, перемещаясь сквозь дерево групп, вы можете указать, какой конкретно параметр, начертание или группу вы хотите настроить.

`M-x customize-option` ⟨RET⟩ *параметр* ⟨RET⟩
: Создать буфер настройки только для одного *параметра*.

`M-x customize-face` ⟨RET⟩ *начертание* ⟨RET⟩
: Создать буфер настройки только для одного *начертания*.

`M-x customize-group` ⟨RET⟩ *группа* ⟨RET⟩
: Создать буфер настройки только для одной *группы*.

`M-x customize-apropos` ⟨RET⟩ *regexp* ⟨RET⟩
: Создать буфер настройки для всех параметров, начертаний и групп, которые соответствуют *regexp*.

`M-x customize-changed-options` ⟨RET⟩ *версия* ⟨RET⟩
: Создать буфер настройки для всех параметров, начертаний и групп, чей смысл изменился начиная с указанной *версии* Emacs.

`M-x customize-saved`
: Создать буфер настройки, содержащий все параметры и начертания, которые вы сохранили с помощью буферов настройки.

`M-x customize-customized`
: Создать буфер настройки, содержащий все параметры и начертания, которые вы изменили, но не сохранили.

Если вы хотите изменить конкретный пользовательский параметр с помощью буфера настройки, и вы знаете его имя, то вы можете использовать команду `M-x customize-option` и указать имя этого параметра. Это создает буфер настройки только для одного параметра — для того, который вы запросили. Изменение, установка и сохранение значения работает точно так же, как описано выше, но только для указанного параметра.

Таким же образом вы можете изменить параметры конкретного начертания, выбранного по имени. Для этого используйте команду `M-x customize-face`.

Вы также можете создать буфер настройки для отдельной группы, используя команду `M-x customize-group`. В этом буфере появится непосредственное содержимое выбранной

группы, в том числе переменные-параметры, начертания и другие группы. Однако содержимое подгрупп изначально будет скрыто. Вы можете просмотреть их содержимое обычным способом, выбрав кнопку '`[Show]`'.

Для более точного указания того, что вы хотите настроить, вы можете использовать команду `M-x customize-apropos`. В качестве аргумента вы указываете регулярное выражение; затем в созданном буфере настройки отображаются все параметры, начертания и группы, чьи имена удовлетворяют этому регулярного выражению. Если вы укажете пустое регулярное выражение, то буфер настройки будет включать *все* группы, параметры и начертания (но это займет длительное время).

Когда вы устанавливаете новую версию Emacs, вы можете захотеть настроить новые параметры, а также те параметры, чей смысл или значения по умолчанию были изменены. Чтобы сделать это, воспользуйтесь командой `M-x customize-changed-options` и укажите в минибуфере номер предыдущей версии Emacs. Эта команда создаст буфер настройки, который покажет вам все параметры (и группы), чье определение изменилось по сравнению с указанной версией.

Если вы изменили значения параметров и затем решили, что сделали это по ошибке, то у вас есть две специальные команды для того, чтобы пересмотреть предыдущие изменения. Используйте команду `customize-saved`, чтобы посмотреть на параметры и начертания, которые вы уже сохранили. А команду `M-x customize-customized` используйте для того, чтобы посмотреть на параметры и начертания, которые вы установили, но еще не сохранили.

### 31.2.3 Ловушки

*Ловушки* — это важный механизм настройки Emacs. Ловушкой называют лисповскую переменную, содержащую список функций, которые вызываются в некоторой определенной ситуации. (Это называется *запуском ловушки*.) Отдельные функции в этом списке называются *функциями* этой *ловушки*. За редкими исключениями, сразу после запуска ловушки в Emacs пустые, так что все функции в любой ловушке — это явно помещенные туда вами для настройки.

Большинство основных режимов запускают на последнем этапе инициализации одну или более *ловушек режима*. Это облегчает настройку поведения режима путем перекрытия локальных переменных, значения которых уже были присвоены данным режимом. Но ловушки также используются в других контекстах. Например, ловушка `suspend-hook` запускается непосредственно перед тем, как Emacs приостанавливает свою работу см. Раздел 3.1 [Выход из Emacs], с. 33).

Большинство ловушек в Emacs являются *нормальными ловушками*. Это означает, что запуск такой ловушки действует путем безусловного вызова всех ее функций без аргументов. Мы старались сохранить большую часть ловушек нормальными, чтобы вы могли использовать их одним и тем же способом. Каждая переменная в Emacs, чье имя оканчивается на '`-hook`', является нормальной ловушкой.

Также есть несколько *аномальных ловушек*. Имена этих переменных оканчиваются на '`-hooks`' или '`-functions`', а не на '`-hook`'. Аномальность этих ловушек заключается в том, что их функции вызываются с какой-то особенностью — возможно, им передаются аргументы, а может возвращаемое ими значение как-то используется. Например, `find-file-not-found-hooks` (см. Раздел 14.2 [Обращение], с. 106) — аномальная ловушка, потому что как только одна из ее функций возвращает отличное от `nil` значение, остальные функции не вызываются совсем. В документации на каждую аномальную ловушку подробно описано, в чем состоит ее особенность.

Для добавления функции к ловушке (как нормальной, так и аномальной) рекомендуется вызывать функцию `add-hook`. Вы можете использовать любую правильную функцию языка Лисп как функцию ловушки, при условии, что она может обработать правильное число

аргументов (нуль в случае нормальной ловушки). Конечно, не всякая лисповская функция *полезна* в каждой конкретной ловушке.

Например, вот как установить ловушку для включения режима Auto Fill при входе в режим Text и в другие режимы, основанные на режиме Text:

```
(add-hook 'text-mode-hook 'turn-on-auto-fill)
```

Следующий пример показывает, как использовать ловушки для настройки отступов в программах на Си. (Люди часто отдают строгое предпочтение одному формату по сравнению с другим). Приводимая здесь функция ловушки является анонимным лямбда-выражением.

```
(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
  (function (lambda ()
    (c-add-style "my-style" my-c-style t))))
```

Лучше всего проектировать функции ловушек таким образом, чтобы порядок выполнения не играл роли. Создавать любую зависимость от порядка вызова — “напрашиваться на проблемы”. Однако, порядок предсказуем: функции, добавленные последними, выполняются первыми.

### 31.2.4 Локальные переменные

`M-x make-local-variable` ⟨RET⟩ *пер* ⟨RET⟩
: Сделать переменную *пер* локальной в текущем буфере.

`M-x kill-local-variable` ⟨RET⟩ *пер* ⟨RET⟩
: Сделать так, что переменная *пер* использовала в текущем буфере свое глобальное значение.

`M-x make-variable-buffer-local` ⟨RET⟩ *пер* ⟨RET⟩
: Пометить переменную *пер* так, чтобы ее установка делала ее локальной для текущего в тот момент буфера.

Почти любая переменная может быть сдалана в *локальной* для определенного буфера Emacs. Это означает, что ее значение в этом буфере не зависит от ее значения в других буферах. Несколько переменных всегда являются локальными в каждом буфере. Любая другая переменная Emacs имеет *глобальное* значение, которое действует во всех буферах, в которых эта переменная не сделана локальной.

Команда `M-x make-local-variable` считывает имя переменной и делает ее локальной для данного буфера. Будущие изменения в данном буфере не затронут другие буферы, а будущие изменения глобального значения не затронут значения для данного буфера.

`M-x make-variable-buffer-local` считывает имя переменной и изменяет будущее ее поведение таким образом, что при установке она автоматически становится локальной. Точнее, если переменная помечается таким способом, то обычные способы установки переменной сначала будут автоматически выполнять `make-local-variable`. Такие переменные мы называем *переменными буфера*.

Основные режимы (см. Глава 19 [Основные режимы], с. 175) всегда делают переменные локальными для буфера до их установки. Поэтому изменение основного режима в одном

буфере не влияет на другие буферы. Второстепенные режимы также работают путем установки переменных — обычно каждый второстепенный режим имеет одну управляющую переменную, которая отлична от `nil`, когда данный режим включен (см. Раздел 31.1 [Второстепенные режимы], с. 341). Для большинства второстепенных режимов управляющая переменная является переменной буфера.

Emacs содержит некоторое количество переменных, которые всегда являются переменными буфера. Сюда включаются `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` и `truncate-lines`. Некоторые другие переменные также всегда являются локальными для текущего буфера, но они используются для внутренних целей.

Некоторые переменные не могут быть локальными для буфера, потому что они вместо этого всегда локальны для каждого дисплея (см. Раздел 17.8 [Множество дисплеев], с. 152). Если вы попытаетесь сделать какую-либо из этих переменных локальной для буфера, то вы получите сообщение об ошибке.

Команда `M-x kill-local-variable` считывает из минибуфера имя переменной и прекращает ее существование как локальной для текущего буфера. С этого момента в этом буфере начинает действовать глобальное значение переменной. Установка основного режима уничтожает все локальные переменные буфера за исключением нескольких, специально отмеченных как *постоянно локальные*.

Для установки глобального значения переменной, независимо от того, имеет ли она локальное значение в текущем буфере, можно использовать функцию Лиспа `setq-default`. Она работает аналогично `setq`, но устанавливает глобальные значения переменных, а не локальные (если они есть). Если в текущем буфере действительно существует локальное значение, то новое глобальное значение может быть невидимо до тех пор, пока вы не переключитесь в другой буфер. Вот пример:

```
(setq-default fill-column 75)
```

`setq-default` — это единственный способ установки глобального значения переменной, которая была помечена функцией `make-variable-buffer-local`.

Программы на Лиспе могут посмотреть на значение некоторой переменной по умолчанию с помощью функции `default-value`. Эта функция принимает в качестве аргумента символ и возвращает его значение по умолчанию. Аргумент вычисляется; обычно вы должны явно поставить перед ним кавычку. Например, вот как можно получить значение по умолчанию для переменной `fill-column`:

```
(default-value 'fill-column)
```

### 31.2.5 Локальные переменные в файлах

Файл может указывать список значений локальных переменных, которые должны использоваться при редактировании этого файла в Emacs. Обращение к файлу проверяет список локальных переменных, при этом каждая из этих переменных делается локальной для буфера, и для нее устанавливается значение, указанное в файле.

Есть два способа указания локальных переменных: в первой строке или с помощью списка локальных переменных. Здесь мы покажем, как задать переменные в первой строке:

`-*- mode:` *имя-режима*`;` *пер*`:` *значение*`; ... -*-`

Таким способом вы можете написать любое количество пар переменная/значение, каждая пара разделяется двоеточием и точкой с запятой, как показано выше. `mode:` *имя-режима*`;` задает основной режим; эта пара должна быть первой в строке. *Значения* не вычисляются, а используются буквально. Вот пример, который задает режим Lisp и устанавливает две переменные с числовыми значениями:

```
;; -*-mode: Lisp; fill-column: 75; comment-column: 50; -*-
```

Этим способом вы можете также указать систему кодирования для данного файла: просто задайте значение для “переменной” с именем `coding`. Значением должно быть имя системы кодирования, которое Emacs может распознать. См. Раздел 18.7 [Системы кодирования], с. 165.

*Список локальных переменных* находится в конце файла, на последней странице. (Часто бывает лучше всего поместить его на отдельную страницу.) Список локальных переменных начинается со строки, содержащей ‘`Local Variables:`’, и оканчивается строкой, содержащей ‘`End:`’. Между ними идут имена переменных и их значения, по одному на строке, в виде ‘*переменная*`:` *значение*’. *Значения* не вычисляются, они используются буквально. Если в файле используются и список локальных переменных, и строка ‘`-*-`’, то Emacs обрабатывает сначала *все* в строке ‘`-*-`’, а затем *все* в списке локальных переменных.

Вот пример списка локальных переменных:

```
;;; Local Variables: ***
;;; mode:lisp ***
;;; comment-column:0 ***
;;; comment-start: ";;; "  ***
;;; comment-end:"***" ***
;;; End: ***
```

Как вы видите, каждая строка начинается с префикса ‘`;;; `’ и заканчивается суффиксом ‘` ***`’. Emacs распознает их как префикс и суффикс, основываясь на первой строке списка, так как они окружают магическую строку ‘`Local Variables:`’; затем они автоматически исключаются из остальных строк списка.

Обычно префиксы и/или суффиксы используются для встраивания списка локальных переменных в комментарии, чтобы он не смущал другие программы, на вход которым подается этот файл. Пример выше написан для языка, где комментарий начинается с ‘`;;; `’ и заканчивается на ‘`***`’; значения локальных переменных `comment-start` и `comment-end` настраивают Emacs на понимание этого необычного синтаксиса. Не используйте префикс или суффикс, если они вам не нужны.

Два “имени переменных” имеют особый смысл в списке локальных переменных: значение для переменной `mode` в действительности устанавливает основной режим, а значение для переменной `eval` просто вычисляется как выражение, а его значение игнорируется. `mode` и `eval` не являются настоящими переменными; установка переменных с именами `mode` и `eval` в любом другом контексте не имеет особого смысла. Если `mode` используется для установки основного режима, то она должна быть первой “переменной” в списке.

Вы можете использовать “переменную” `mode` для установки второстепенных режимов точно так же, как и основных режимов; в действительности, вы можете использовать ее несколько раз, сначала для установки основного режима, а затем для установки второстепенных режимов, которые будет действовать для выбранного буфера. Но большинство второстепенных режимов не стоит никак указывать в файле, поскольку они представляют предпочтения пользователя.

Например, у вас можете появиться искушение включить режим Auto Fill с помощью списка локальных переменных. Это будет ошибкой. Использовать режим Auto Fill или нет — это дело личного вкуса, а не свойство содержимого файла. Если вы хотите использовать режим Auto Fill, то установите ловушки основных режимов в вашем файле ‘`.emacs`’, чтобы он включался (когда нужно) только для вас (см. Раздел 31.7 [Файл инициализации], с. 366). Не используйте список локальных переменных для навязывания вашего вкуса всем остальным.

Список локальных переменных должен начинаться не далее, чем за 3000 знаков от конца файла, и он должно находиться на последней странице, если файл поделен на страницы. Иначе Emacs не заметит его там. Цель этого в том, чтобы случайная ‘`Local Variable:`’,

появившаяся не на последней странице, не путала Emacs, и чтобы при обращении к длинному файлу, который полностью является одной страницей и не имеет списка локальных переменных, не тратилось время на просмотр всего файла.

Используйте команду `normal-mode` для переустановки локальных переменных и основного режима данного буфера соответственно имени файла и его содержимого, включая списки локальных переменных, если они есть. См. Раздел 19.1 [Выбор режима], с. 175.

Переменная `enable-local-variables` говорит, нужно ли обрабатывать локальные переменные в файлах, и таким образом дает вам шанс перекрыть их. По умолчанию ее значение равно `t`, что означает обработку локальных переменных в файлах. Если вы установите значение переменной равным `nil`, то Emacs просто будет игнорировать локальные переменные в файлах. Любое другое значение велит делать запрос у пользователя о каждом файле, в котором имеются локальные переменные, показывая определения локальных переменных, чтобы вы могли принять решение.

"Переменная" `eval` и некоторые настоящие переменные создают некий риск; когда вы обращаетесь к чужим файлам, определения локальных переменных для них могут произвольно воздействовать на ваш Emacs. Поэтому параметр `enable-local-eval` контролирует, будет ли Emacs обрабатывать переменные `eval`, а так же переменные, чьи имена оканчиваются на '`-hook`', '`-hooks`', '`-function`' или '`-functions`', а также некоторые другие переменные. Существует три возможных значения для данного параметра: `t`, `nil` и что-нибудь другое, точно так же, как и для `enable-local-variables`. Значением по умолчанию является `maybe`, это не `t` и не `nil`, так что обычно Emacs спросит подтверждение об установке этих переменных.

## 31.3 Клавиатурные макросы

*Клавиатурный макрос* — это определенная пользователем команда, обозначающая другую последовательность ключей. Например, если вы обнаружили, что вам нужно набрать `C-n C-d` сорок раз, то вы можете ускорить работу, определив клавиатурный макрос для `C-n C-d` и вызвав его со счетчиком повторений 40.

`C-x (` Начать определение клавиатурного макроса (`start-kbd-macro`).

`C-x )` Закончить определение клавиатурного макроса (`end-kbd-macro`).

`C-x e` Выполнить самый последний клавиатурный макрос (`call-last-kbd-macro`).

`C-u C-x (` Заново выполнить последний клавиатурный макрос и затем добавить дополнительные ключи к его определению.

`C-x q` Когда достигается эта точка при выполнении макроса, сделать запрос о подтверждении (`kbd-macro-query`).

`M-x name-last-kbd-macro`
Задать имя команды (на время текущего сеанса) для последнего определенного клавиатурного макроса.

`M-x insert-kbd-macro`
Вставить в буфер определение клавиатурного макроса как код на Лиспе.

`C-x C-k` Отредактировать ранее определенный клавиатурный макрос (`edit-kbd-macro`).

`M-x apply-macro-to-region-lines`
Запустить последний клавиатурный макрос на каждой полной строке в области.

Клавиатурные макросы отличаются от обычных команд Emacs тем, что они написаны на командном языке Emacs, а не на Лиспе. Это облегчает их написание для новичков и делает их более удобными в качестве временных хаков. Однако, мощности командного языка

Emacs как языка программирования недостаточно, чтобы он был удобным для написания чего-либо умного или универсального. Для таких вещей надо использовать Лисп.

Вы определяете клавиатурные макросы во время выполнения команд, являющихся определением. Говоря другими словами, во время определения клавиатурного макроса его определение выполняется в первый раз. Таким образом, вы можете видеть, каково действие ваших команд, так что вы не должны просчитывать его в уме. Когда вы кончили, макрос клавиатуры определен, а также один раз фактически выполнен. После этого вы можете снова и снова выполнять все целиком, вызывая макрос.

### 31.3.1 Основы использования

Для начала определения клавиатурного макроса наберите команду `C-x (` (`start-kbd-macro`). Начиная с этого момента ваши нажатия на клавиши по-прежнему выполняются, но также становятся частью определения макроса. В строке режима появляется слово '`Def`', чтобы напомнить вам о том, что происходит. Когда вы закончите, команда `C-x )` (`end-kbd-macro`) закончит определение макроса (но не станет его частью!). Например,

```
C-x ( M-f foo C-x )
```

определяет макрос для перемещения вперед на слово и вставки '`foo`'.

Определенный таким образом макрос может запускаться снова с помощью команды `C-x e` (`call-last-kbd-macro`), в качестве числового аргумента которой можно задать счетчик повторов для многократного выполнения макроса. Команде `C-x )` также можно задать счетчик повторов в качестве аргумента, в этом случае именно столько раз она повторяет макрос сразу после его определения, но само определение макроса засчитывается как первое повторение (так как оно выполняется в то время, как вы его определяете). Таким образом, `C-x )` с аргументом 4 приводит к непосредственному выполнению макроса еще 3 раза. Аргумент 0 для `C-x e` или `C-x )` означает повторение макроса бесконечное число раз (пока он не получит ошибку, или вы не наберете `C-g` или, в MS-DOS, `C-`⟨BREAK⟩).

Если вы хотите повторять операцию в регулярно расположенных местах в тексте, то определите макрос и включите в него команды для передвижения к следующему месту, в котором вы хотите его использовать. Например, если вы хотите изменить каждую строку, вам нужно поставить точку в начало строки и определить макрос, изменяющий эту строку и перемещающий точку в начало следующей строки. После этого повторение макроса будет обрабатывать строки одну за другой.

После того как вы завершили определение макроса, вы можете добавить что-либо в его конец, набрав `C-u C-x )`. Это эквивалентно обычной `C-x (`, за которой следует повторный набор всего имевшегося до сих пор определения. Как следствие этого, она повторно выполнит макрос, как было определено ранее.

Вы можете использовать в клавиатурных макросах функциональные клавиши, точно так же, как клавиши клавиатуры. Вы даже можете использовать события от мыши, но будьте внимательны с ними: когда макрос проигрывает событие мыши, он использует оригинальную позицию мыши для этого события, ту, которую мышь имела во время определения макроса. Что может при этом произойти, трудно предсказать. (Эффект от использования текущей позиции мыши был бы еще менее предсказуем.)

Одна из вещей, которая не всегда правильно работает в клавиатурных макросах, — это команда `C-M-c` (`exit-recursive-edit`). Когда эта команда выводит из рекурсивного редактирования, которое было начато внутри макроса, то она работает так, как вы ожидали. Но если вы выходите из рекурсивного редактирования, которое было начато до того, как вы запустили клавиатурный макрос, то также происходит выход из клавиатурного макроса как из части этого процесса.

Вы можете отредактировать уже существующий клавиатурный макрос, используя `C-x C-k` (`edit-kbd-macro`). Затем вы должны ввести то, что вы будете использовать для

вызова макроса — `C-x e` или `M-x` *имя* или какую-то другую последовательность ключей. Это форматирует определение макроса в буфере и входит в специальный основной режим для его редактирования. Наберите в этом буфере `C-h m`, чтобы получить подробности о редактировании макроса. Когда вы закончите редактирование, нажмите `C-c C-c`.

Команда `M-x apply-macro-to-region-lines` повторяет последний определенный клавиатурный макрос для каждой полной строки внутри текущей области. Она делает это строка за строкой, перемещая точку в начало строки и выполняя затем макрос.

### 31.3.2 Именование и сохранение клавиатурных макросов

Если вы хотите сохранить макрос клавиатуры дольше, чем до следующего определения, то вы должны дать ему имя, используя `M-x name-last-kbd-macro`. Эта команда считывает имя как аргумент, используя минибуфер, и определяет это имя для выполнения макроса. Имя макроса — это лисповский символ, а определение его таким способом делает его допустимым именем команды для вызова при помощи `M-x` или для привязывания ключа с помощью `global-set-key` (см. Раздел 31.4.1 [Таблицы ключей], с. 356). Если вы укажете имя, уже имеющее определение, отличное от макроса клавиатуры, то печатается сообщение об ошибке, и ничего не изменяется.

Когда макрос получает имя команды, вы можете записать его определение в файл. Потом его можно будет использовать в другом сеансе редактирования. Сначала обратитесь к файлу, в котором хотите записать определение. Затем используйте эту команду:

`M-x insert-kbd-macro` ⟨RET⟩ *имя-макро* ⟨RET⟩

Это вставляет Лисп-код, который, будучи выполнен позднее, определит тот же самый макрос с тем же самым определением, которое он имеет сейчас. (Чтобы сделать это, вам не нужно разбираться в Лисп-коде, так как `insert-kbd-macro` напишет его за вас.) Затем сохраните файл. Позже вы можете загрузить этот файл с помощью `load-file` (см. Раздел 23.7 [Библиотеки Лиспа], с. 253). Если файл, в который вы записываете, является вашим файлом инициализации ‘`~/.emacs`’ (см. Раздел 31.7 [Файл инициализации], с. 366), то макрос будет определяться каждый раз, когда вы запускаете Emacs.

Если вы зададите команде `insert-kbd-macro` числовой аргумент, то она создаст дополнительный Лисп-код для записи ключей (если они есть), которые вы привязали к макросу клавиатуры; таким образом, когда вы загрузите файл, макрос будут вновь привязан к тем же самым ключам.

### 31.3.3 Выполнение макроса с вариациями

Используя `C-x q` (`kbd-macro-query`), вы можете достигнуть эффекта, сходного с действием `query-replace`, когда макрос каждый раз запрашивает у вас, должен ли он производить изменения. Во время определения макроса наберите `C-x q` в той точке, где вы хотите получать запрос. При определении макроса эта команда ничего не делает, но когда вы запускаете макрос, то `C-x q` произведет интерактивный запрос о продолжении действий.

Правильными ответами на запрос от `C-x q` являются ⟨SPC⟩ (или `y`), ⟨DEL⟩ (или `n`), ⟨RET⟩ (или `q`), `C-l` и `C-r`. Ответы те же самые, что и для `query-replace`, хотя не все варианты ответа для `query-replace` имеют смысл.

Эти ответы включают ⟨SPC⟩ для продолжения, а ⟨DEL⟩ для пропуска остатка этого повторения макроса и начала нового повторения макроса. ⟨RET⟩ означает пропуск остатка данного повторения и отмены остальных повторений. `C-l` перерисовывает экран и снова делает запрос.

`C-r` входит на уровень рекурсивного редактирования, где вы можете выполнить редактирование, которое не является частью макроса. Когда вы выйдете из рекурсивного

редактирования с помощью `C-M-c`, у вас снова спросят о том, как продолжать выполнение клавиатурного макроса. Если тогда вы нажмете ⟨SPC⟩, то будет выполнен остаток макроса.

`C-u C-x q`, то есть `C-x q` с числовым аргументом, осуществляет совершенно другую функцию. Она входит в рекурсивное редактирование, считывая ввод с клавиатуры, и когда вы набираете его во время определения макроса, и когда он выполняется из макроса. Во время определения, редактирование, которое вы делаете внутри рекурсивного редактирования, не становится частью макроса. Во время выполнения макроса рекурсивное редактирование дает вам возможность выполнить какое-либо особенное редактирование в каждом повторении. См. Раздел 30.12 [Рекурсивное редактирование], с. 338.

Другой способ изменить поведение клавиатурного макроса — использовать в качестве счетчика регистр, увеличивая его при каждом повторе макроса. См. Раздел 10.5 [РегЧисла], с. 78.

## 31.4 Настройка привязок ключей

Этот раздел описывает *привязки ключей*, которые отображают ключи в команды, и *таблицы ключей*, которые сохраняют привязки. Здесь также объясняется, как изменять привязки ключей для своих нужд.

Напомним, что команда — это функция Лиспа, чье определение обеспечивает интерактивное использование. Подобно любой лисповской функции, каждая команда имеет имя, которое обычно состоит из букв нижнего регистра и дефисов.

### 31.4.1 Таблицы ключей

Привязки между последовательностями ключей и функциями-командами сохраняются в структурах данных, называемых *таблицами ключей*. В Emacs много таких таблиц, каждая из которых используется в особых случаях.

Напомним, что *последовательность ключей* (или коротко, *ключ*) — это последовательность *событий ввода*, которая имеет смысл как одно целое. События ввода включают в себя знаки, функциональные клавиши и кнопки мыши — все виды ввода, какие вы можете послать компьютеру с вашего терминала. Последовательность ключей получает свой смысл из *привязки*, которая говорит, какую команду данный ключ запускает. Назначение таблиц ключей состоит в сохранении этих привязок.

*Глобальная* таблица ключей является наиболее важной, потому что она действует всегда. Глобальная таблица определяет ключи для режима Fundamental; большинство из этих определений являются общими для всех основных режимов. Каждый основной или второстепенный режим может иметь свою собственную таблицу ключей, которая перекрывает глобальные определения некоторых ключей.

Например, самовставляющийся знак, такой как `g`, является самовставляющимся, потому что глобальная таблица ключей привязывает его к команде `self-insert-command`. Стандартные знаки редактирования в Emacs, такие как `C-a`, также получают свой стандартный смысл из глобальной таблицы ключей. Команды для переприязки ключей, такие как `M-x global-set-key`, на самом деле работают путем сохранения новой привязки в соответствующем месте глобальной таблицы ключей. См. Раздел 31.4.5 [Переприязка], с. 359.

Meta-знаки работают по другому; Emacs транслирует каждый Meta-знак в пары знаков, начинающиеся с ⟨ESC⟩. Когда вы набираете знак `M-a`, Emacs заменяет ее на ⟨ESC⟩ `a`. Meta-клавиша приходит как одиночное событие ввода, но для целей привязки ключей становится двумя событиями. Это происходит по историческим причинам, и мы можем когда-нибудь это изменить.

На большинстве современных клавиатур помимо знаковых клавиш есть функциональные клавиши. Функциональные клавиши посылают события ввода, точно так же, как и знаковые клавиши, и таблицы ключей могут содержать привязки для них.

На многих терминалах нажатие на функциональную клавишу в действительности посылает компьютеру последовательность знаков; точная информация о том, что это за последовательность, зависит от самой функциональной клавиши и от того, какую модель терминала вы используете. (Часто такая последовательность начинается с ⟨ESC⟩ `[`.) Если Emacs понимает ваш тип терминала правильно, то он распознает последовательности знаков, формирующие функциональные клавиши, в каком бы месте последовательности ключей они не встречались (не только в начале). Таким образом, для большинства целей вы можете считать, что функциональные клавиши достигают Emacs непосредственно, и игнорировать их кодирование как последовательность знаков.

Кнопки мыши также производят события ввода. Эти события приносят другую информацию — окно и позицию, где была нажата или отпущена кнопка мыши, и временную метку. Но для привязок ключей важно лишь знать кнопку; другие данные играют роль, только если это нужно команде. (Команды, разработанные для вызова с помощью мыши, на самом деле обычно обращают внимание на эти данные).

Таблица ключей записывает определения для одиночных событий. Для интерпретации последовательности ключей, состоящей из нескольких событий, нужна цепочка таблиц ключей. Первая таблица дает определение первого события; это определение является другой таблицей ключей, которая используется для поиска второго события в последовательности, и так далее.

Последовательность ключей может содержать нажатия на функциональные и знаковые клавиши. Например, `C-x` ⟨SELECT⟩ имеет смысл. Если вы сделаете клавишу ⟨SELECT⟩ префиксным ключом, то ⟨SELECT⟩ `C-n` тоже будет иметь смысл. Вы даже можете смешивать события от мыши с событиями от клавиатуры, но мы не рекомендуем делать так, потому что такие последовательности неудобно набирать.

Как пользователь вы можете перепривязать любой ключ; но будет лучше, если вы всегда будете использовать последовательности ключей, состоящие из `C-c`, за которым следует буква. Эти клавиши “зарезервированы для пользователей”, так что они не будут конфликтовать ни с одним правильно спроектированным расширением Emacs. Функциональные клавиши от ⟨F5⟩ до ⟨F9⟩ также зарезервированы для пользователей. Если вы перепривязываете какой-то другой ключ, ваше определение может быть перекрыто некоторыми расширениями или основными режимами, которые переопределяют тот же самый ключ.

### 31.4.2 Таблицы префиксных ключей

Префиксный ключ, такой как `C-x` или ⟨ESC⟩, имеют собственную таблицу ключей, которая хранит определения для событий, непосредственно следующих за этим префиксом.

Определение префиксного ключа — это обычно таблица ключей, в которой ищется следующее событие. Это определение также может быть лисповским символом, чье определение функции является следующей таблицей ключей; результат этого тот же самый, но он предоставляет для префиксного ключа командное имя, которое может быть использовано как описание того, для чего предназначен этот префиксный ключ. Таким образом, привязка `C-x` — это символ `Ctl-X-Prefix`, чье определение функции является таблицей ключей для команд на `C-x`. Определения `C-c`, `C-x`, `C-h` и ⟨ESC⟩ как префиксных ключей появляются в глобальной таблице, так что эти префиксные ключи доступны всегда.

Помимо обычных префиксных ключей существуют фиктивные “префиксные ключи”, которые представляют полоску меню; смотрите раздел “Menu Bar” в *The Emacs Lisp Reference Manual*, для дополнительной информации о привязках ключей полоски меню. События от кнопок мыши, которые запускают всплывающие меню, также являются префиксными

ключами; смотрите раздел “Menu Keymaps” в *The Emacs Lisp Reference Manual*, для дополнительной информации.

Некоторые таблицы префиксных ключей хранятся в именованных переменных:

- `ctl-x-map` — это имя переменной для таблицы, используемой для знаков, следующих за `C-x`.
- `help-map` для знаков, следующих за `C-h`.
- `esc-map` для знаков, следующих за ⟨ESC⟩. Таким образом, все Meta-знаки в действительности определяются этой таблицей.
- `ctl-x-4-map` для знаков, следующих за `C-x 4`.
- `mode-specific-map` для знаков, следующих за `C-c`.

### 31.4.3 Локальные таблицы ключей

До сих пор мы объясняли принципы работы глобальной таблицы. Основные режимы настраивают Emacs, предоставляя собственные привязки ключей в *локальных таблицах ключей*. Например, режим C переопределяет ⟨TAB⟩ для создания отступа текущей строки кода на Си. Куски текста в буфере могут указывать свои собственные таблицы ключей взамен таблицы основного режима этого буфера.

Второстепенные режимы также могут иметь локальные таблицы ключей. Когда действует второстепенный режим, определения из его таблицы ключей перекрывают и локальную таблицу основного режима, и глобальную таблицу.

Локальные таблицы ключей для режима Lisp и некоторых других основных режимов всегда существуют, даже когда не используются. Они хранятся в переменных с именами `lisp-mode-map` и так далее. Для менее часто используемых основных режимов локальная таблица ключей обычно создается при первом использовании в сеансе. Это сберегает ресурсы. Если вы хотите изменить одну из этих таблиц ключей, то вы должны использовать *ловушку режима* — смотрите ниже.

Все таблицы ключей второстепенных режимов создаются заранее. Не существует способа задержать их создание до тех пор, пока этот второстепенный режим не будет включен первый раз.

Локальная таблица ключей может локально переопределять ключ как префиксный, определяя его как префиксную таблицу ключей. Если этот ключ определен как префикс и в глобальной таблице, то его локальное и глобальное определения (из обоих таблиц ключей) эффективно комбинируется: обе они используются для поиска события, которое следует за префиксным ключом. Таким образом, если локальная таблица ключей определяет `C-c` как еще одну таблицу ключей, а эта таблица определяет `C-z` как команду, то это придает локальный смысл для `C-c C-z`. Это не затрагивает другие последовательности, которые начинаются с `C-c`; если у этих последовательностей нет собственной локальной привязки, то продолжают действовать глобальные привязки.

Другой способ понять это — считать, что Emacs обрабатывает последовательности ключей, состоящие из нескольких событий, просматривая несколько таблиц ключей одну за одной в поисках этой последовательности целиком. Сначала он проверяет таблицы ключей текущих второстепенных режимов, затем таблицу ключей основного режима и затем он ищет в глобальной таблице. Это не совсем точно описывает то, как работает поиск, но достаточно хорошо для понимания обычного положения дел.

Для изменения локальных привязок основного режима вы должны изменить локальную таблицу ключей этого режима. Обычно вы должны дождаться первого использования режима, поскольку до того момента большинство режимов не создают свои таблицы ключей. Если вы хотите указать что-нибудь в вашем файле ‘`~/.emacs`’ для изменения привязок основного режима, то вы должны использовать ловушку этого режима, чтобы задержать изменения до его первого использования.

Например, команда `texinfo-mode` для выбора режима Texinfo запускает ловушку `texinfo-mode-hook`. Продемонстрируем, как можно использовать эту ловушку для добавления локальных привязок (мы признаем, не слишком полезной) для клавиш `C-c n` и `C-c p` в режиме Texinfo:

```
(add-hook 'texinfo-mode-hook
          '(lambda ()
             (define-key texinfo-mode-map
                         "\C-cp"
                         'backward-paragraph)
             (define-key texinfo-mode-map
                         "\C-cn"
                         'forward-paragraph)
             ))
```

См. Раздел 31.2.3 [Ловушки], с. 349.

### 31.4.4 Таблицы ключей минибуфера

Минибуфер имеет свой собственный набор локальных таблиц ключей; они содержат разные команды завершения и выхода.

- `minibuffer-local-map` используется для обыкновенного ввода (без завершения).
- `minibuffer-local-ns-map` похожа, но ⟨SPC⟩ выходит, точно так же, как ⟨RET⟩. Это используется главным образом для совместимости с Mocklisp.
- `minibuffer-local-completion-map` для свободного завершения.
- `minibuffer-local-must-match-map` для строгого и осторожного завершения.

### 31.4.5 Интерактивное изменение привязок ключей

Чтобы переопределить ключ в Emacs, надо изменить ее запись в таблице ключей. Вы можете изменить глобальную таблицу ключей, тогда изменение подействует на все основные режимы (за исключением тех, в которых есть свои локальные, перекрывающие определения для того же самого ключа). Или вы можете изменить локальную таблицу ключей текущего буфера, что затронет все буферы, использующие тот же самый основной режим.

`M-x global-set-key` ⟨RET⟩ *ключ кмд* ⟨RET⟩
: Глобально определяет *ключ* для запуска команды *кмд*.

`M-x local-set-key` ⟨RET⟩ *ключ кмд* ⟨RET⟩
: Локально определяет *ключ* (в текущем основном режиме) для запуска команды *кмд*.

`M-x global-unset-key` ⟨RET⟩ *ключ*
: Делает *ключ* неопределенным в глобальной таблице ключей.

`M-x local-unset-key` ⟨RET⟩ *ключ*
: Делает *ключ* неопределенным в локальной таблице ключей (в текущем основном режиме).

Допустим например, что вы хотите выполнять команды в подоболочке внутри буфера Emacs, а не приостанавливать Emacs и выполнять команды в вашей входной оболочке. Обычно `C-z` привязан к функции `suspend-emacs` (когда не используется система X Windows), но вы можете изменить `C-z` на запуск интерактивной подоболочки внутри Emacs, привязав ее к команде `shell`, как показано:

```
M-x global-set-key ⟨RET⟩ C-z shell ⟨RET⟩
```

`global-set-key` считывает имя команды после клавиши. После того как вы нажмете клавишу, появится примерно такое сообщение, так что вы сможете убедиться, что назначаете ту клавишу, которую хотите:

```
Set key C-z to command:
```

Вы можете переопределить функциональные клавиши и события от мыши тем же самым способом; просто нажмите на функциональную клавишу или щелкните кнопкой мыши, когда у вас спрашивают, какой ключ нужно перепривязать.

Вы можете перепривязать ключ, который содержит больше одного события, тем же самым способом. Emacs продолжает считывать ключ для перепривязки до тех пор, пока он не станет полным ключом (то есть непрефиксным). Таким образом, если вы нажмете `C-f` как *ключ*, то на этом все закончится; вы сразу попадете в минибуфер для считывания команды *кмд*. Но если вы нажмете `C-x`, считывается еще один знак; если это `4`, считывается еще один, и так далее. Например,

```
M-x global-set-key ⟨RET⟩ C-x 4 $ spell-other-window ⟨RET⟩
```

переопределяет `C-x 4 $` для запуска команды (фиктивной) `spell-other-window`.

Двухзнаковые ключи, состоящие из `C-c` и буквы, зарезервированы для пользовательской настройки. Предполагается, что программы на Лиспе не определяют эти ключи, так что ваши привязки для них будут доступны во всех основных режимах и не будут ничему мешать.

Вы можете убрать глобальное определение ключа с помощью команды `global-unset-key`. Это делает ключ *неопределенным*; если вы наберете его, Emacs просто подаст звуковой сигнал. Сходным образом `local-unset-key` делает ключ неопределенным в таблице ключей текущего основного режима, и в этом основном режиме вступает в действие определение (или отсутствия определения) из глобальной таблицы ключей.

Если вы переопределили (или сделали неопределенным) какой-то ключ и затем хотите отказаться от изменения, то отмена определения не будет работать — вам необходимо переопределить ключ, используя стандартное определение. Чтобы найти имя стандартного определения ключа, перейдите в буфер с режимом Fundamental и примените `C-h c`. Описания ключей в данном руководстве также сообщают имена соответсвующих команд.

Если вы хотите обезопасить себя от ошибочного вызова команды, то лучше блокировать команду, а не отменять определение ключа. Блокированную команду проще вызвать, когда вы на самом деле захотите этого. См. Раздел 31.4.11 [Блокирование команды], с. 364.

### 31.4.6 Перепривязка ключей в файле инициализации

Если у вас есть набор привязок ключей, которые вы хотите использовать всегда, вы можете указать их в вашем файле ‘`.emacs`’, используя синтаксис Лиспа.

Простейший способ работает только со знаками ASCII и их Meta-вариантами. Этот метод использует для представления перепривязываемой последовательности ключей строку. Например, как привязать `C-z` к команде `shell`:

```
(global-set-key "\C-z" 'shell)
```

Этот пример использует строковую константу, содержащую один знак, `C-z`. Одинарная кавычка перед именем команды, `shell`, отмечает ее как константный символ, а не как переменную. Если вы опустите кавычку, Emacs попытается сразу вычислить `shell` как переменную. Это вероятно вызовет ошибку; это явно не то, что вы хотите.

Вот другой пример, который привязывает последовательность ключей из двух знаков:

```
(global-set-key "\C-xl" 'make-symbolic-link)
```

Когда последовательность ключей включает в себя функциональные клавиши, или события от кнопок мыши или не входящие в ASCII знаки, такие как `C-=` или `H-a`, вы должны

использовать более общий метод перепривязки, в котором для указания последовательности ключей используется вектор.

Для того чтобы записать вектор в Emacs Lisp, необходимо заключить элементы вектора в квадратные скобки. Для разделения элементов используйте пробелы. Если элемент является символом, то просто напишите имя символа — никаких других разделителей или пунктуации не нужно. Если элемент вектора является знаком, то пишите его как знаковую константу Лиспа: '`?`', за которым следует сам знак в том виде, как он появился бы в строке.

Вот примеры использования векторов для перепривязки `C-=` (управляющий знак, не входящий в ASCII), `H-a` (Hyper-знак; в ASCII вообще нет Hyper); ⟨F7⟩ (функциональная клавиша) и `C-Mouse-1` (модифицированная с клавиатуры кнопка мыши):

```
(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)
```

Вы также можете использовать вектор и для простых случаев. Вот как можно переписать первые два примера, приведенные выше, с использованием векторов:

```
(global-set-key [?\C-z] 'shell)

(global-set-key [?\C-x ?l] 'make-symbolic-link)
```

### 31.4.7 Перепривязка функциональных клавиш

Последовательности ключей могут содержать функциональные клавиши, как и обычные знаки. Так же, как лисповские знаки (на самом деле целые числа) представляют знаки клавиатуры, лисповские символы представляют функциональные клавиши. Если на функциональной клавиша надписано слово, то это слово также будет именем соответствующего лисповского символа. Здесь приводятся лисповские имена для часто встречающихся функциональных клавиш:

`left`, `up`, `right`, `down`
: Курсорные стрелки.

`begin`, `end`, `home`, `next`, `prior`
: Другие клавиши перемещения курсора.

`select`, `print`, `execute`, `backtab`
`insert`, `undo`, `redo`, `clearline`
`insertline`, `deleteline`, `insertchar`, `deletechar`,
: Различные функциональные клавиши.

`f1`, `f2`, `...` `f35`
: Нумерованные функциональные клавиши (расположенные сверху клавиатуры).

`kp-add`, `kp-subtract`, `kp-multiply`, `kp-divide`
`kp-backtab`, `kp-space`, `kp-tab`, `kp-enter`
`kp-separator`, `kp-decimal`, `kp-equal`
: Клавиши дополнительной клавиатуры (справа от основной клавиатуры), с именами или знаками пунктуации.

`kp-0`, `kp-1`, `...` `kp-9`
: Клавиши дополнительной клавиатуры с цифрами.

`kp-f1`, `kp-f2`, `kp-f3`, `kp-f4`
: Функциональные клавиши дополнительной клавиатуры.

Эти имена являются общепринятыми, но некоторые системы (особенно при использовании X Windows) могут использовать другие имена. Чтобы убедиться, какой символ используется для определенной функциональной клавиши на вашем терминале, наберите `C-h c` и затем нужную клавишу.

Последовательность ключей, которая содержит символы функциональных клавиш (или что-то другое кроме ASCII-знаков), должны быть вектором, а не строкой. Синтаксис векторов использует пробел между элементами и квадратные скобки вокруг всего вектора. Таким образом, для привязки функциональной клавиши '`f1`' к команде `rmail`, напишите следующее:

```
(global-set-key [f1] 'rmail)
```

Для привязки клавиши с правой стрелкой к команде `forward-char`, вы можете использовать такое выражение:

```
(global-set-key [right] 'forward-char)
```

Здесь используется лисповский синтаксис для вектора, содержащего символ `right`. (Эта привязка существует по умолчанию в Emacs).

См. Раздел 31.4.6 [Переприязка при старте], с. 360, для дополнительной информации об использовании векторов для переприязки.

В последовательности ключей вы можете смешивать функциональные клавиши и знаки. Этот пример привязывает `C-x` ⟨NEXT⟩ к команде `forward-page`.

```
(global-set-key [?\C-x next] 'forward-page)
```

где `?\C-x` — это знаковая константа Лиспа для знака `C-x`. Элемент вектора `next` является лисповским символом, и поэтому к нему не приписан вопросительный знак.

Вы можете использовать клавиши-модификаторы ⟨CTRL⟩, ⟨META⟩, ⟨HYPER⟩, ⟨SUPER⟩, ⟨ALT⟩ и ⟨SHIFT⟩ вместе с функциональными клавишами. Для представления этих модификаторов добавьте строки '`C-`', '`M-`', '`H-`', '`s-`', '`A-`' и '`S-`' в начало имени символа. Таким образом, чтобы `Hyper-Meta-`⟨RIGHT⟩ перемещал вперед на слово, нужно сделать так:

```
(global-set-key [H-M-right] 'forward-word)
```

### 31.4.8 Именованные управляющие ASCII-знаки

⟨TAB⟩, ⟨RET⟩, ⟨BS⟩, ⟨LFD⟩, ⟨ESC⟩ и ⟨DEL⟩ начали свою жизнь как имена определенных управляющих знаков ASCII, использовавшихся так часто, что для них были сделаны специальные клавиши. Позднее, пользователи сочли удобным различать в Emacs эти клавиши и "те же самые" управляющие знаки, набранными с помощью клавиши ⟨CTRL⟩.

Emacs различает эти два типа ввода, когда используется с X Window System. Он обрабатывает эти "особые" клавиши как функциональные с именами `tab`, `return`, `backspace`, `linefeed`, `escape` и `delete`. Эти функциональные клавиши автоматически транслируются в соответствующие ASCII-знаки, *если* у них нет собственных привязок. В результате ни пользователи, ни программы на Лиспе не обязаны помнить об этом различии, если только им это не нужно.

Если вы не хотите различать (например) ⟨TAB⟩ и `C-i`, просто сделайте одну привязку для ASCII-знака ⟨TAB⟩ (восьмиричный код 011). Если вы действительно хотите различать их, то сделайте одну привязку для этого ASCII-знака, а вторую для "функциональной клавиши" `tab`.

На обычных ASCII-терминалах нет способа различать ⟨TAB⟩ и `C-i` (и аналогично для других таких пар), потому что терминал посылает один и тот же знак в обоих случаях.

### 31.4.9 Не-ASCII-знаки на клавиатуре

Если на вашей клавиатуре есть клавиши, которые посылают знаки, не входящие в ASCII, например акцентированные буквы, перепривязка их делается несколько хитро. Есть два решения. Одно — указать систему кодирования для клавиатуры, используя `set-keyboard-coding-system` (см. Раздел 18.9 [Задание кодирования], с. 168). Затем вы можете привязывать эти клавиши обычным способом, но записывая

```
(global-set-key [?знак] 'какая-то-функция)
```

и подставляя *знак*, который вы хотите привязать

Если вы не указываете систему кодирования для клавиатуры, этот способ не сработает. Вместо этого вам нужно выяснить код, который в действительности посылает терминал. Простейший способ сделать это в Emacs — создать пустой буфер с помощью `C-x b temp` ⟨RET⟩, сделать его однобайтным, набрав `M-x toggle-enable-multibyte-characters` ⟨RET⟩, а затем нажать клавишу, которая вставит в этот буфер нужный знак.

Расположите точку сразу перед этим знаком, затем наберите `C-x =`. Это отобразит в минибуфере сообщение, показывающее восьмиричный, шестнадцатиричный и десятичный код знака, все в круглых скобках. Используйте второе из этих трех чисел, десятичное, внутри вектора для привязки:

```
(global-set-key [десятичный-код] 'какая-то-функция)
```

### 31.4.10 Перепривязка кнопок мыши

Emacs использует лисповские символы и для обозначения кнопок мыши. Обычными событиями от мыши в Emacs являются события-*щелчки*; это случается, когда вы нажимаете и отпускаете кнопку без перемещения мыши. Вы можете также получить событие-*проведение*, когда вы перемещаете мышь, держа нажатой кнопку. События-проведения в действительности происходят, когда вы отпускаете кнопку мыши.

Символами для основных событий-щелчков являются `mouse-1` для левой кнопки мыши, `mouse-2` для следующей кнопки и так далее. Вот как можно переопределить вторую кнопку мыши для разделения текущего окна:

```
(global-set-key [mouse-2] 'split-window-vertically)
```

Символы для события-проведения похожи на предыдущие, но имеют префикс ‘`drag-`’ перед словом ‘`mouse`’. Например, проведение с помощью первой кнопки генерирует событие `drag-mouse-1`.

Вы также можете определить привязки для событий, которые возникают в момент нажатия на кнопку мыши. Имена этих событий начинаются со слова ‘`down-`’ вместо ‘`drag-`’. Такие события генерируются, только если они имеют привязку. Когда вы получите событие-нажатие, то за ним всегда будут следовать соответствующее событие-щелчок или проведение.

Если хотите, вы можете различать одиночные, двойные и тройные щелчки. Двойной щелчок означает щелканье кнопкой мыши дважды почти в одном и том же месте. Первый щелчок генерирует обычное событие-щелчок. Второй щелчок, если он приходит достаточно быстро, генерирует событие-двойной щелчок. Тип события для двойного щелчка начинается с ‘`double-`’: например, `double-mouse-3`.

Это означает, что вы можете придать особый смысл второму щелчку в том же самом месте, но он должен действовать в предположении, что обычное определение одинарного щелчка уже обработалось, когда был получен первый щелчок.

Это ограничивает ваши возможности по использованию двойных щелчков, но дизайнеры пользовательских интерфейсов говорят, что этому ограничению нужно следовать в любом случае. Двойной щелчок должен делать что-то подобное одиночному щелчку, только

“чуть больше”. Команда для события-двойного щелчка должна выполнять больше работы для двойного щелчка.

Если для события двойного щелчка нет привязки, оно изменяется на соответствующее событие одиночного щелчка. Таким образом, если вы не определите специально двойной щелчок, то будет дважды выполняться команда для одиночного щелчка.

Emacs также поддерживает события-тройные щелчки, чьи имена начинаются с ‘`triple-`’. Emacs не различает четвертное нажатие как тип события; щелчки свыше третьего генерируют дополнительные события-тройные щелчки. Однако полное количество щелчков сохраняется в списке событий, так что вы можете различать такие случаи, если вам это действительно нужно. Мы не рекомендуем особых значений для более чем трех щелчков, но иногда полезно, чтобы последующие щелчки проходили по циклу того же набора трех значений, так что четыре щелчка эквивалентны одному, пять эквивалентны двум, а шесть — трем.

Emacs также записывает множественные нажатия в событиях проведения и нажатия. Например, когда вы дважды нажмете кнопку, а затем переместите мышь, держа кнопку нажатой, Emacs получит событие ‘`double-drag-`’. А когда вы нажимаете на кнопку второй раз, Emacs получит событие ‘`double-down-`’ (которое игнорируется, подобно всем событиям нажатия, если у них нет привязки).

Переменная `double-click-time` задает промежуток времени, который может пройти между двумя щелчками, чтобы они считались парой. Ее значение измеряется в миллисекундах. Если значение равно `nil`, то двойной щелчок не распознается совсем. Если значение равно `t`, то нет никакого временного ограничения.

Символы для событий мыши также показывают статус клавиш-модификаторов, с обычными префиксами ‘`C-`’, ‘`M-`’, ‘`H-`’, ‘`s-`’, ‘`A-`’ и ‘`S-`’. Они всегда стоят перед ‘`double-`’ или ‘`triple-`’, которые всегда предшествуют ‘`drag-`’ или ‘`down-`’.

Фрейм включает области, которые не отображают текст буфера, такие как строка режима и линейка прокрутки. Вы можете судить о том, что кнопка мыши нажата на специальной области экрана, посредством префиксных псевдо-ключей. Например, если вы щелкнете мышью в строке режима, то вы получите префиксный ключ `mode-line` перед обычным символом, обозначающем кнопку мыши. Вот как определить щелчок первой кнопкой на строке режима для запуска `scroll-up`:

```
(global-set-key [mode-line mouse-1] 'scroll-up)
```

Вот полный список таких префиксных псевдо-ключей и их значений:

`mode-line`
: Мышь находилась на строке режима окна.

`vertical-line`
: Мышь находилась на вертикальной линии, разделяющей окна. (Если вы используете полоски прокрутки, они появляются вместо этих вертикальных линий).

`vertical-scroll-bar`
: Мышь находилась на вертикальной полоске прокрутки. (Пока это единственная разновидность полосок прокрутки, которые поддерживает Emacs).

Вы можете поместить в последовательность ключей больше одного события от кнопок мыши, но обычно так не делают.

### 31.4.11 Блокирование команд

Блокирование помечает команду как требующую подтверждения до того, как она будет выполнена. Цель блокирования состоит в том, чтобы защитить начинающих пользователей от случайного выполнения команд, которые могли бы их запутать.

Попытка интерактивного вызова блокированной команды в Emacs отображает окно, содержащее имя команды, ее описание и некоторые рекомендации о том, что надо сделать немедленно; затем Emacs спросит у вас ввод, указывающий, нужно ли выполнять эту команду, как запрошено, разблокировать ее или отменить запрос. Если вы решите разблокировать команду, то вас спросят, выполнить ли это постоянно или только для текущего сеанса. Постоянное разблокирование производится путем автоматического редактирования вашего файла '`.emacs`'.

Прямой механизм блокирования команды — помещение отличного от `nil` свойства `disabled` в лисповский символ для данной команды. Вот программа на Лиспе, которая делает это:

```
(put 'delete-region 'disabled t)
```

Если значение свойства `disabled` является строкой, то эта строка включается в сообщение, выводимое при использовании этой команды:

```
(put 'delete-region 'disabled
     "It's better to use `kill-region' instead.\n")
```

Вы можете блокировать команду либо непосредстенным редактированием файла '`.emacs`', либо с помощью команды `M-x disable-command`, которая радактирует файл '`.emacs`' за вас. Подобным образом команда `M-x enable-command` изменяет '`.emacs`' для постоянного разрешения команды. См. Раздел 31.7 [Файл инициализации], с. 366.

Блокирование команды не зависит от того, какой ключ используется для ее запуска; блокирование также действует, если команда вызывается через `M-x`. Блокирование не имеет эффекта при вызове команды как функции из программ на Лиспе.

## 31.5 Перевод клавиатуры

Некоторые клавиатуры не позволяют удобно посылать все специальные знаки, которые использует Emacs. Наиболее общим случаем этой проблемы является знак ⟨DEL⟩. Некоторые клавиатуры не обеспечивают удобного способа для набора этого очень важного знака — обычно потому что они спроектированы в предположении, что для удаления будет использоваться знак `C-h`. На таких клавиатурах, если вы нажмете клавишу, обычно используемую для удаления, то Emacs обработает `C-h` как префиксный ключ и предложит вам список параметров, а это не то, что вам нужно.

Вы можете обойти эту проблему внутри Emacs, подготовив перевод клавиатуры для превращения `C-h` в ⟨DEL⟩ и ⟨DEL⟩ в `C-h`, как показано:

```
;; Переводит C-h в ⟨DEL⟩.
(keyboard-translate ?\C-h ?\C-?)

;; Переводит ⟨DEL⟩ в C-h.
(keyboard-translate ?\C-? ?\C-h)
```

Перевод клавиатуры — это не то же самое, что и привязка ключей в таблицах ключей (см. Раздел 31.4.1 [Таблицы ключей], с. 356). Emacs содержит многочисленные таблицы ключей, которые применяются в разных ситуациях, но только один набор переводов клавиш, и он применяется к каждому знаку, который Emacs считывает с терминала. Перевод клавиатуры происходят на низшем уровне обработки ввода; ключи, поиск которых происходит в таблицах ключей, содержат знаки, появляющиеся после перевода клавиатуры.

При работе в X, клавиша с именем ⟨DELETE⟩ является функциональной клавишей, и она отличается от ASCII-знака с именем ⟨DEL⟩. См. Раздел 31.4.8 [Именованные ASCII-знаки], с. 362. Перевод клавиатуры затрагивает только ввод ASCII-знаков, но не функциональных клавиш; таким образом, пример выше, примененный под X, не влияет на клавишу ⟨DELETE⟩. Однако такой перевод не является необходимым под X, потому что Emacs может также различать клавишу ⟨BACKSPACE⟩ и `C-h`; и обычно он рассматривает ⟨BACKSPACE⟩ как ⟨DEL⟩.

Для более полной информации об использовании перевода клавиатуры смотрите раздел “Translating Input” в *The Emacs Lisp Reference Manual*.

## 31.6 Синтаксическая таблица

Все команды Emacs, которые разбирают слова или балансируют скобки, управляются *синтаксической таблицей*. Синтаксическая таблица говорит, какие знаки являются открывающими ограничителями, частями слов, кавычками строковых констант и так далее. У каждого основного режима есть своя синтаксическая таблица (хотя родственные основные режимы часто разделяют одну), которая устанавливается в любом буфере, использующем этот режим. Все команды применяют синтаксическую таблицу, установленную в текущем буфере; именно ее мы имеем в виду, когда мы говорим просто “синтаксическая таблица”. Синтаксическая таблица — это лисповский объект, таблица знаков, чьими элементами являются числа.

Чтобы вывести на экран описание содержимого текущей синтаксической таблицы, наберите `C-h s` (`describe-syntax`). Описание каждого знака включает строку, которую вы передали бы `modify-syntax-entry` для установки текущего синтаксиса, и небольшое объяснение этой строки на английском, если необходимо.

Для получения полной информации о синтаксических таблицах смотрите раздел “Syntax Tables” в *The Emacs Lisp Reference Manual*.

## 31.7 Файл инициализации, ‘`~/.emacs`’

Когда Emacs запускается, он обычно загружет Лисп-программу из файла ‘`.emacs`’ в вашем начальном каталоге. Мы называем этот файл вашим *файлом инициализации*, поскольку он указывает, как инициализировать Emacs для вас. Вы можете использовать ключи командной строки ‘`-q`’, чтобы предотвратить загрузку файла инициализации, и ‘`-u`’ (или ‘`-user`’), чтобы указать файл инициализации другого пользователя (см. Глава 3 [Вход в Emacs], с. 33).

Также может быть *файл инициализации по умолчанию*, это библиотека с именем ‘`default.el`’, находимая по стандартному пути поиска для библиотек. В дистрибутиве Emacs нет такой библиотеки; она может быть создана на вашей системе для локальных настроек. Если эта библиотека существует, она загружается при запуске Emacs (только если вы не задали ‘`-q`’). Но ваш файл инициализации, если он существует, загружается первым; если он устанавливает `inhibit-default-init` в отличное от `nil` значение, то ‘`default`’ не загружается.

На вашей системе также может быть *системный файл запуска*; он называется ‘`site-start.el`’, если существует. Emacs загружает эту библиотеку до вашего файла инициализации. Чтобы подавить загрузку этой библиотеки, используйте ключ ‘`-no-site-file`’.

Если ваш файл ‘`.emacs`’ содержит большой объем кода, вам стоит переместить его в другой файл, как ‘`~/`*что-нибудь*`.el`’, скомпилировать его и сделать так, чтобы ваш файл ‘`.emacs`’ загружал его через `(load "~/`*что-нибудь*`")`. См. раздел “Byte Compilation” в *the Emacs Lisp Reference Manual*, для получения большей информации о компилировании программ на Emacs Lisp.

Если вы собираетесь писать настоящие программы на Emacs Lisp, которые идут дальше простой настройки, вам нужно прочитать книгу *Emacs Lisp Reference Manual*.

### 31.7.1 Синтаксис файла инициализации

Файл ‘`.emacs`’ содержит одно или несколько лисповских выражений вызовов функций. Каждое из них состоит из имени функции, за которым следуют аргументы, всё в окру-

жении круглых скобок. Например, `(setq fill-column 60)` вызывает функцию `setq` для установки переменной `fill-column` (см. Раздел 21.5 [Заполнение], с. 185) в значение 60.

Второй аргумент `setq` — это выражение для нового значения переменной. Это может быть константа, переменная или вызов функции. В файле '`.emacs`' чаще всего используются константы. Это могут быть:

Числа: Числа записываются в десятичной форме, с необязательным знаком минус в начале.

Строки: Синтаксис лисповских строк такой же, как в Си, но с некоторыми дополнительными чертами. Для начала и завершения строковой константы используйте знак двойных кавычек.

В строковых константах вы можете писать переводы строк и специальные знаки буквально. Но часто для ясности лучше использовать для них последовательности с обратной косой чертой: '`\n`' для перевода строки, '`\b`' для возврата на одну позицию, '`\r`' для возврата каретки, '`\t`' для табуляции, '`\f`' для прогона страницы (control-L), '`\e`' для escape, '`\\`' для обратной косой черты, '`\"`' для двойных кавычек или '`\`*ooo*' для знака с восьмиричным кодом *ooo*. Обратная косая черта и двойные кавычки — это единственные знаки, для которых такие последовательности обязательны.

'`\C-`' можно использовать как префикс для управляющего знака, как '`\C-s`' для ASCII control-S, а '`\M-`' можно применять в качестве префикса для Meta-знака, как '`\M-a`' для `Meta-A` или '`\M-\C-a`' для `Control-Meta-A`.

Знаки: Синтаксис лисповских знаковых констант состоит из '`?`', за которым идет либо знак, либо последовательность, начинающаяся с '`\`'. Примеры: `?x`, `?\n`, `?\"`, `?\)`. Заметьте, что строки и знаки не взаимозаменимы в Лиспе; в некоторых контекстах требуется одно, в некоторых другое.

Истина: `t` обозначает 'истину'.

Ложь: `nil` обозначает 'ложь'.

Другие лисповские объекты:
Пишите одиночную кавычку (') и за ней желаемый лисповский объект.

### 31.7.2 Примеры файла инициализации

Вот несколько примеров выполнения часто нужных вещей с помощью лисповских выражений:

- Сделать так, чтобы ⟨TAB⟩ в режиме C просто вставлял табуляцию, если точка находится в середине строки.

  ```
  (setq c-tab-always-indent nil)
  ```

  Здесь мы имеем переменную, чье значение обычно равно `t`, то есть 'истина', а альтернатива ему — `nil`, 'ложь'.

- Сделать так, чтобы поиск производился с учетом регистра (во всех буферах, которые не перекрывают это).

  ```
  (setq-default case-fold-search nil)
  ```

  Это устанавливает значение по умолчанию, оно распространяется на все буферы, в которых нет локальных значений для этой переменной. Установка `case-fold-search` с помощью `setq` затрагивает только локальное значение текущего буфера, а это не то, что вы, скорее всего, хотите сделать в файле инициализации.

- Указать ваш адрес электронной почты, если Emacs не может правильно выяснить его сам.

```
(setq user-mail-address "coon@yoyodyne.com")
```

Различные пакеты Emacs, которым нужно знать ваш адрес, используют значение `user-mail-address`.

- Сделать режим Text режимом для новых буферов по умолчанию.

```
(setq default-major-mode 'text-mode)
```

Заметьте, что используется `text-mode`, поскольку это команда для входа в режим Text. Одиночная кавычка перед ним делает этот символ константой; иначе `text-mode` рассматривался бы как имя переменной.

- Подготовить установки по умолчанию для набора знаков Latin-1, который поддерживает большинство западноевропейских языков.

```
(set-language-environment "Latin-1")
```

- Включать режим Auto Fill автоматически в режиме Text и родственных с ним.

```
(add-hook 'text-mode-hook
  '(lambda () (auto-fill-mode 1)))
```

Это показывает, как добавить функцию-ловушку к переменной-ловушке (см. Раздел 31.2.3 [Ловушки], с. 349). Функция, которую мы предоставляем — это список, начинающийся с `lambda`, с одиночной кавычкой в начале, чтобы сделать этот список константой, а не выражением.

Объяснение функций Лиспа выходит за рамки данного руководства, но для этого примера достаточно знать, что действием будет вычисление `(auto-fill-mode 1)`, когда вы входите в режим Text. При желании вы можете заменить это на другое выражение или на несколько выражений подряд.

Emacs поставляется с функцией с именем `turn-on-auto-fill`, чье определение — это `(lambda () (auto-fill-mode 1))`. Таким образом, более простой способ написать приведенное выше выражение выглядит так:

```
(add-hook 'text-mode-hook 'turn-on-auto-fill)
```

- Загрузить установленную лисповскую библиотеку с именем '`foo`' (на самом деле файл '`foo.elc`' или '`foo.el`' из стандартного каталога Emacs).

```
(load "foo")
```

Когда аргументом `load` является относительное имя файла, не начинающееся с '`/`' или '`~`', `load` просматривает каталоги из `load-path` (см. Раздел 23.7 [Библиотеки Лиспа], с. 253).

- Загрузить скомпилированный Лисп-файл '`foo.elc`' из вашего начального каталога.

```
(load "~/foo.elc")
```

Здесь использовано абсолютное имя файла, поэтому поиск не производится.

- Перепривязать ключ `C-x l` на запуск функции `make-symbolic-link`.

```
(global-set-key "\C-xl" 'make-symbolic-link)
```

или

```
(define-key global-map "\C-xl" 'make-symbolic-link)
```

Еще раз обратите внимание, одиночная кавычка используется для ссылки на символ `make-symbolic-link`, а не на его значение как переменной.

- Сделать то же самое, только для режима Lisp.

```
(define-key lisp-mode-map "\C-xl" 'make-symbolic-link)
```

- Переопределить все ключи, которые сейчас запускают `next-line` в режиме Fundamental, чтобы они вместо этого запускали `forward-line`.

```
(substitute-key-definition 'next-line 'forward-line
                           global-map)
```

- Сделать `C-x C-v` неопределенным.

  ```
  (global-unset-key "\C-x\C-v")
  ```

  Одна из причин для удаления определения ключа состоит в том, чтобы вы могли сделать его префиксом. Просто определение `C-x C-v` *что-угодно* сделает `C-x C-v` префиксом, но сначала нужно лишить `C-x C-v` его обычного непрефиксного определения.
- Присвоить '`$`' синтаксическую категорию пунктуации в режиме Text. Обратите внимание на использование знаковой константы для '`$`'.

  ```
  (modify-syntax-entry ?\$ "." text-mode-syntax-table)
  ```
- Разрешить использование команды `narrow-to-region` без подтверждения.

  ```
  (put 'narrow-to-region 'disabled nil)
  ```

### 31.7.3 Инициализация терминала

Для каждого типа терминала может быть библиотека, загружаемая в Emacs, когда он запускается на этом типе терминала. Для типа терминала с именем *тип-терм* эта библиотека называется '`term/`*тип-терм*', и она находится, как обычно, путем поиска в каталогах `load-path`, при этом пробуются окончания '`.elc`' и '`.el`'. Обычно она появляется в подкаталоге '`term`' из каталога, где хранится большинство библиотек Emacs.

Обычное назначение специфичной для терминала библиотеки — отображение escape-последовательностей, используемых функциональными клавишами терминала, в более осмысленные имена с помощью функции `function-key-map`. Пример того, как это делается, смотрите в файле '`term/lk201.el`'. Многие функциональные клавиши отображаются автоматически в соответсвтии с базой данных Termcap; в специфичной для терминала библиотеке нужно описать только те функциональные клавиши, которые на указаны в Termcap.

Когда тип терминала содержит дефис, при выборе имени библиотеки имеет значение только часть перед первым дефисом. Таким образом, типы терминалов '`aaa-48`' и '`aaa-30-rv`' оба используют библиотеку '`term/aaa`'. Код библиотеки может применять `(getenv "TERM")` для выяснения полного имени типа терминала.

Имя библиотеки конструируется конкатенацией значения переменной `term-file-prefix` и типа терминала. Ваш файл '`.emacs`' может предотвратить загрузку специфичной для терминала библиотеки, устанавливая `term-file-prefix` равной `nil`.

В конце инициализации, когда считаны и ваш файл '`.emacs`', и любая специфичная для терминала библиотека, Emacs запускает ловушку `term-setup-hook`. Добавьте к этой ловушке функции, если вы хотите перекрыть часть специфичной для терминала библиотеки или определить инициализацию для терминалов, к которым нет библиотеки. См. Раздел 31.2.3 [Ловушки], с. 349.

### 31.7.4 Как Emacs находит файл инициализации

Обычно для поиска файла '`.emacs`' Emacs использует переменную среды `HOME`; знак '`~`' в имени файла обозначает именно это. Но если вы сделали `su`, Emacs пытается найти ваш собственный '`.emacs`', а не того пользователя, за кого вы себя сейчас выдаете. Идея в том, чтобы вы получали свои собственные настройки редактора, даже если работаете как привелигированный пользователь.

Более точно, Emacs сначала определяет, файл инициализации какого пользователя нужно использовать. Он получает имя пользователя из переменных среды `LOGNAME` и `USER`; если ни одна из них не существует, берется эффективный ID пользователя. Если имя пользователя соответствует реальному пользовательскому ID, то Emacs использует `HOME`; иначе, он находит начальный каталог в системной базе данных о пользователях.

# 32 Решение частых проблем

Если вы наберете не ту команду Emacs, которую имели в виду, то последствия этого могут быть непонятны для вас. Эта глава рассказывает о том, что вы можете сделать для того, чтобы отменить вашу ошибку и выйти из непонятной ситуации. Здесь также рассматриваются сбои в Emacs и аварийные завершения.

## 32.1 Выход и аварийное завершение

`C-g`
`C-`⟨BREAK⟩ `(MS-DOS)`
: Выход. Отменить выполняемую или частично набранную команду.

`C-]`
: Прервать самый глубокий уровень рекурсивного редактирования и отменить команду, которая его запустила (`abort-recursive-edit`).

⟨ESC⟩ ⟨ESC⟩ ⟨ESC⟩
: Выход или прерывание, в зависимости от того, что подходит в этот момент (`keyboard-escape-quit`).

`M-x top-level`
: Прервать все работающие в данный момент уровни рекурсивного редактирования.

`C-x u`
: Отменить ранее сделанное изменение в содержимом буфера (`undo`).

Существует два способа отмены команд, выполнение которых не закончилось: *выход* с помощью `C-g` и *прерывание* с помощью `C-]` или `M-x top-level`. Выход отменяет частично набранную команду или команду, которая уже выполняется. Прерывание выходит из одного уровня рекурсивного редактирования и отменяет команду, которая запустила это рекурсивное редактирование. (См. Раздел 30.12 [Рекурсивное редактирование], с. 338.)

Выход с помощью `C-g` используется для того, чтобы освободиться от частично набранной команды или числового аргумента, который вам не нужен. Она также относительно безопасно останавливает выполнение уже работающей команды, так что вы можете использовать ее, если случайно зададите команду, требующей много времени. В частности, выход из уничтожения безопасен; либо ваш текст *полностью* останется в буфере, либо *полностью* попадет в список уничтожений (или будет и там, и там). Выход из наращиваемого поиска выполняет специальные действия, документированные отдельно; в общем случае, чтобы выйти из поиска, может потребоваться два последовательных `C-g` (см. Раздел 12.1 [Наращиваемый поиск], с. 87).

В MS-DOS клавишей выхода, подобной `C-g`, служит `C-`⟨BREAK⟩. Так сделано по той причине, что в MS-DOS невозможно зарегистрировать нажатие `C-g` во время работы команды между интеракциями с пользователем. Напротив, зарегистрировать `C-`⟨BREAK⟩ можно *всегда*. См. Раздел C.1 [Клавиатура и мышь в MS-DOS], с. 403.

Работа `C-g` заключается в установке переменной `quit-flag` в значение `t`, как только `C-g` был набран; Emacs Lisp часто проверяет эту переменную и выходит, если она не равна `nil`. `C-g` действительно выполняется как команда, только если вы набирали ее в то время, когда Emacs ожидает ввода.

Если вы выйдите с помощью `C-g` второй раз до того, как первая `C-g` была опознана, то вы активизируете средство “аварийного выхода” и возвращаетесь в оболочку. См. Раздел 32.2.8 [Аварийный выход], с. 374.

Бывает много случаев, когда вы не сможете выйти. Когда Emacs ожидает, пока операционная система завершит какое-нибудь действие, выход невозможен, если только не предприняты особые меры для некоторых системных вызовов из Emacs, где случается

ожидание. Мы предприняли эти меры для тех системных вызовов, из которых пользователи, вероятно, захотят выходить, но может случиться, что вы найдете еще один. В одном очень распространенном случае, при ожидании ввода или вывода файла с использованием NFS, сам Emacs знает, как выйти, но большинство реализаций NFS просто не позволяют пользовательским программам прекратить ожидание NFS, когда NFS-сервер завис.

Прерывание с помощью `C-]` (`abort-recursive-edit`) используется для того, чтобы выйти из одного уровня рекурсивного редактирования и отменить команду, которая его запустила. Выход с помощью `C-g` не делает этого и не может этого делать, потому что он используется для отмены частично набранной команды *внутри* текущего уровня рекурсивного редактирования. Обе эти операции полезны. Например, если вы находитесь в рекурсивном редактировании и набираете `C-u 8`, чтобы ввести числовой аргумент, то вы можете отменить этот аргумент с помощью `C-g` и остаться в рекурсивном редактировании.

Команда ⟨ESC⟩ ⟨ESC⟩ ⟨ESC⟩ (`keyboard-escape-quit`) может совершить и выход, и прерывание. Этот ключ был определен, потому что во многих программах для PC для “выхода” используется ⟨ESC⟩. Он может отменить числовой аргумент, очистить выделенную область или выйти из замены с подтверждением, как `C-g`. Он также может прервать работу минибуфера или одного уровня рекурсивного редактирования, как `C-]`. Он также может отменить разбиение фрейма на несколько окон, как `C-x 1`. Единственное, что он не может сделать — остановить работающую команду. Причина этого в том, что этот ключ выполняется как обыкновенная команда, и Emacs не замечает ее, пока не будет готов принять новую команду.

Команда `M-x top-level` эквивалентна “достаточному количеству” команд `C-]`, чтобы прервать все уровни рекурсивного редактирования, в которых вы находитесь. `C-]` дает вам возможность покинуть один уровень за один раз, а `M-x top-level` покидает все уровни сразу. Как `C-]`, так и `M-x top-level` похожи на все другие команды и не похожи на `C-g` тем, что они действуют, только когда Emacs готов для приема команды. `C-]` — обыкновенный ключ и имеет смысл только благодаря привязке в таблице ключей. См. Раздел 30.12 [Рекурсивное редактирование], с. 338.

`C-x u` (`undo`), строго говоря, не является способом отмены команды, но вы можете представлять ее как отмену команды, уже окончившей выполнение. См. Раздел 4.4 [Отмена], с. 37.

## 32.2 Что делать с неполадками в Emacs

Этот раздел описывает различные условия, при которых Emacs отказывается работать правильно, и рассказывает, как распознать и исправить проблемы.

### 32.2.1 Если ⟨DEL⟩ не удаляет

Если вы обнаружили, что ⟨DEL⟩ выдает справку, как `Control-h`, а не удаляет знак, это значит, что ваш терминал посылает неверный код для ⟨DEL⟩. Вы можете справиться с этой проблемой, изменив таблицу преобразования клавиатуры (см. Раздел 31.5 [Перевод клавиатуры], с. 365).

### 32.2.2 Уровни рекурсивного редактирования

Уровни рекурсивного редактирования являются важным и полезным средством Emacs, но тем пользователям, кто их не понимает, они могут показаться сбоем в работе Emacs.

Если в строке режима вокруг круглых скобок есть квадратные скобки ‘`[...]`’, которые содержат имена главного и второстепенных режимов, то это означает, что вы вошли в уровень рекурсивного редактирования. Если вы не хотели этого или не понимаете, что это значит, то вы должны просто покинуть этот уровень рекурсивного редактирования. Чтобы

это сделать, наберите `M-x top-level`. Это называется возвратом к верхнему уровню. См. Раздел 30.12 [Рекурсивное редактирование], с. 338.

### 32.2.3 Мусор на экране

Если данные на экране выглядят неправильно, то первое, что надо сделать, — посмотреть, действительно ли текст неправилен. Наберите `C-l`, чтобы весь экран полностью перерисовался. Если он после этого появится в правильном виде, то неисправность была в предыдущем обновлении экрана. (Если нет, смотрите Раздел 32.2.4 [Мусор в тексте], с. 373.)

Проблемы с обновлением изображения часто возникают из-за неправильной записи termcap для используемого вами терминала. Файл '`etc/TERMS`' в дистрибутиве Emacs содержит исправления для известных проблем такого рода. '`INSTALL`' содержит в одном из своих разделов общие рекомендации по этим проблемам. Весьма вероятно, что просто недостаточно наполнение некоторых операций отображения. Чтобы проверить, не этого ли сорта у вас проблема, попробуйте запустить Emacs на другом терминале, сделанном другим производителем. Если неполадки обычно случаются на определенном виде терминалов, а на других нет, то это происходит, вероятно, из-за неправильной записи в termcap, хотя это также может быть результатом ошибки в Emacs, которая появляется на терминалах с какими-то особенностями или которым не хватает специфических средств.

### 32.2.4 Мусор в тексте

Если `C-l` показывает, что текст неправильный, попробуйте отменить в нем изменения, используя `C-x u` до тех пор, пока она не вернет текст к состоянию, которое вы считаете правильным. Кроме того, попробуйте `C-h l`, чтобы найти команду, набор которой привел к замеченным результатам.

Если кажется, что в начале или конце буфера пропал большой кусок текста, проверьте наличие слова '`Narrow`' в строке режима. Если оно там есть, то текст все еще присутствует в буфере, но он помечен как выходящий за границы доступности. Чтобы сделать его вновь видимым, наберите `C-x n w`. См. Раздел 30.8 [Сужение], с. 335.

### 32.2.5 Самопроизвольный вход в наращиваемый поиск

Если Emacs самопроизвольно показывает '`I-search:`' внизу экрана, то это означает, что терминал посылает `C-s` и `C-q` из-за плохо разработанного протокола управления потоком данных типа xon/xoff.

Если это с вами случилось, лучшим выходом будет постараться установить терминал в такой режим, где он не будет использовать управление потоком данных, или давая ему достаточное заполнение, чтобы он никогда не посылал `C-s`. (Один из способов увеличить величину заполнения — присвоить переменной `baud-rate` большее значение. Значение этой переменной обозначает скорость вывода терминала, выраженную в обычных единицах, бодах.)

Если вам не удалось выключить управление потоком, тогда лучше всего будет сказать Emacs справиться с ним самому. Чтобы сделать это, вызовите функцию `enable-flow-control`.

Как правило, встречаются определенные типы терминалов, на которых вы вынуждены использовать управление потоком. Используя `enable-flow-control-on`, вы можете удобно запросить использование управления потоком только для этих типов терминалов. Например, если вы обнаружили, что должны использовать управление потоком на терминалах VT-100 и H19, поместите в ваш файл '`.emacs`' следующее:

```
(enable-flow-control-on "vt100" "h19")
```

Когда задействовано управление потоком, вы должны набирать `C-\`, чтобы получить результат `C-s`, и `C-^`, чтобы получить результат `C-q`. (Эти псевдонимы работают посредством преобразований клавиатуры; смотрите Раздел 31.5 [Перевод клавиатуры], с. 365.)

### 32.2.6 Исчерпание памяти

Если вы получили сообщение '`Virtual memory exceeded`'[1], сохраните измененные буферы командой `C-x s`. Этот способ сохранения буферов требует минимальное количество дополнительной памяти. Emacs хранит резерв памяти, которая становится доступна, когда возникает эта ошибка; его должно быть достаточно, чтобы `C-x s` могла завершить свою работу.

Как только вы сохранили измененные буферы, вы можете выйти из этого задания Emacs и начать другое или воспользоваться командой `M-x kill-some-buffers`, чтобы освободить пространство для текущего задания Emacs. Если вы уничтожите буферы, содержащие значительный объем текста, вы сможете безопасно продолжать редактирование. Emacs заново резервирует память автоматически, когда видит, что доступно достаточно свободного места, на случай, если память опять закончится.

Не используйте `M-x buffer-menu`, чтобы сохранить или уничтожить буферы, когда вы исчерпали память, потому что меню буферов само требует заметного количества памяти, и резерва может не хватить.

### 32.2.7 Восстановление после краха

Если в Emacs или компьютере произошел фатальный сбой, вы можете восстановить файлы, которые вы редактировали в момент краха, из их автоматически сохраненных версий. Чтобы сделать это, запустите Emacs снова и наберите команду `M-x recover-session`.

Сначала эта команда покажет буфер, перечисляющий файлы прерванных сеансов, каждый со своей датой. Вы должны выбрать сеанс, который вы хотите восстановить. Обычно это самый последний. Переместите точку к выбранному файлу и введите `C-c C-c`.

Затем `recover-session` спросит вас о каждом файле, который вы редактировали во время того сеанса; она спрашивает, нужно ли восстанавливать этот файл. Если вы отвечаете `y` для какого-нибудь файла, она показывает даты этого файла и его автоматически сохраненной версии и снова спрашивает, нужно ли его восстанавливать. Этот второй запрос вы должны подтверждать вводом `yes`. Если вы это сделаете, Emacs обращается к этому файлу, но берет текст из самосохраненного файла.

Когда `recover-session` завершает работу, выбранные вами файлы находятся в буферах Emacs. Теперь вы должны их сохранить. Только это — их запись — обновляет сами файлы.

### 32.2.8 Аварийный выход

Так как в свое время были ошибки, вызывавшие зацикливание Emacs без проверки `quit-flag`, было введено специальное средство, которое заставляет Emacs немедленно прерваться, если вы наберете второй `C-g`, когда флаг уже установлен. Таким образом, вы всегда можете покинуть GNU Emacs. Обычно Emacs опознает и очищает `quit-flag` (и выходит!) достаточно быстро, чтобы этого не случалось. (В MS-DOS и совместимых системах набирайте дважды `C-`⟨BREAK⟩.)

Когда вы возвращаетесь в Emacs после прерывания, вызванного множественными `C-g`, он задает два вопроса, перед тем как вернуться к тому, чем он был занят:

[1] Виртуальная память исчерпана. *(Прим. переводчика)*

```
Auto-save? (y or n)
Abort (and dump core)? (y or n)
```

Отвечайте на каждый из них `y` или `n` и последующим ⟨RET⟩.

Ответ `y` на '`Auto-save?`' вызывает немедленную автоматическую запись всех модифицированных буферов, в которых включено автоматическое сохранение.

Ответ `y` на '`Abort (and dump core)?`' приводит к выполнению недопустимой инструкции и сбросу дампа памяти. Это нужно, чтобы дать возможность специалисту понять, почему Emacs не выполнил выход в первый раз. После сброса дампа памяти выполнение не продолжается. Если вы ответите `n`, выполнение продолжится. Если повезет, GNU Emacs в конечном счете проверит `quit-flag` и выйдет обычным образом. Если нет, и вы наберете другую `C-g`, он прервется снова.

Если Emacs на самом деле не завис, а просто медленно работает, вы можете вызвать действие двойного `C-g`, не желая этого в действительности. Тогда просто возобновите работу Emacs и ответьте `n` на оба вопроса, и вы попадете в прежнее состояние. Вероятно, затребованный вами выход скоро произойдет.

Средство двойного `C-g` отключено, когда Emacs запускается в системе X Windows, поскольку программа управления окнами всегда дает вам возможность уничтожить Emacs или создать другое окно и запустить другую программу.

В MS-DOS и совместимых системах аварийный выход иногда невозможен, даже если вы нажимаете `C-`⟨BREAK⟩ дважды, когда зависает какой-то системный вызов (MS-DOS или BIOS), или когда Emacs попал в очень короткий бесконечный цикл (в коде на Си, **не** на Лиспе).

### 32.2.9 Помощь при полном разочаровании

Если использование Emacs (или что-нибудь еще) страшно вас расстраивает и никакие методы, описанные выше, не решают проблему, то Emacs все еще может вам помочь.

Во-первых, если ваш Emacs не отвечает на команды, наберите `C-g C-g`, чтобы выйти из него и запустить новый.

Во-вторых, наберите `M-x doctor` ⟨RET⟩.

Доктор поможет вам, и вы почувствуете себя лучше. Каждый раз, когда вы что-нибудь говорите доктору, вы должны оканчивать это набором ⟨RET⟩ ⟨RET⟩. Это позволит доктору узнать, что вы закончили.

## 32.3 Описание ошибок в Emacs

Иногда вы будете сталкиваться с ошибками в Emacs. Хотя мы не можем обещать, что можем или будем исправлять ошибки, мы можем даже не согласиться, что это ошибка, но мы все равно хотим услышать о проблемах, с которыми вы столкнулись. Часто мы соглашаемся, что это ошибки, и хотим их исправить.

Чтобы дать нам возможность исправить ошибку, вы должны описать ее. Чтобы сделать это эффективно, вы должны знать когда и как это делать.

### 32.3.1 Когда это ошибка

Если Emacs выполняет недопустимую инструкцию или прерывается с сообщением операционной системы об ошибке, которая указывает на неполадку в программе (в противоположность чему-нибудь вроде "нет места на диске"), то это определенно ошибка в Emacs.

Если Emacs обновляет изображение так, что оно не соответствует тому, что находится в буфере, то это определенно ошибка. Если похоже, что команда выполняет ошибочные

действия, но неполадка исправляется сама, если вы наберете `C-l`, то это случай неправильного обновления изображения.

Бесконечное ожидание завершения команды может быть ошибкой, но вы должны точно определить, что это действительно происходит по вине Emacs. Некоторые команды просто требуют много времени. Наберите `C-g` (`C-`⟨BREAK⟩ в MS-DOS) и затем `C-h l`, чтобы увидеть, получил ли Emacs тот ввод, какой вы хотели набрать. Если ввод был таким, о котором вы *знаете*, что он должен обрабатываться быстро, сообщайте об ошибке. Если вы не знаете, должна ли эта команда требовать много времени, выясните это, либо просмотрев руководство, либо попросив помощи.

Если команда, с которой вы хорошо знакомы, выдает сообщение об ошибке там, где ее обычное определение должно быть правильным, то это, вероятнее всего, ошибка.

Если команда выполняет неправильные действия, то это ошибка. Но чтобы быть уверенным, вы должны знать точно, что она должна была сделать. Если вы не очень хорошо знакомы с этой командой или не знаете наверняка, как она должна действовать, может оказаться, что в действительности она работает правильно. Прежде чем делать поспешный вывод, покажите вашу проблему тому, кто знает точно.

Наконец, предполагаемое определение команды может не быть лучшим способом редактирования. Это очень важная проблема, но это также вопрос точки зрения. Кроме того, можно легко прийти к такому заключению из-за незнания некоторых существующих возможностей. Скорее всего, лучше не жаловаться на неполадку до тех пор, пока вы не проверили как обычно документацию, не почувствовали, что разобрались до конца и знаете наверняка, что то, что вы хотите, недоступно. Если после внимательного прочтения руководства вы не уверены том, что должна делать команда, посмотрите предметный указатель и глоссарий для любого термина, который может быть неясен.

Если и после подробного прочтения руководства вы не поняли, что должна делать эта команда, то это указывает на ошибку в руководстве, о которой нужно сообщить. Задача руководства — прояснить все для неэкспертов в Emacs, включая вас. Сообщать об ошибках в документации так же важно, так и об ошибках в программе.

Если диалоговая строка документации функции или переменной не согласуется с руководством, то одно из них неправильно; опишите это как ошибку.

### 32.3.2 Понимание отчетов об ошибках

Когда вы решите, что действительно нашли ошибку, то важно сообщить о ней и сделать это таким образом, чтобы описание было полезным. Что особенно полезно, так это точное описание команд, которые вы набирали, начиная с команды оболочки для запуска Emacs и вплоть до возникновения ошибки.

Самый важный принцип при сообщении об ошибках — сообщать *факты*. Гипотезы и словесные описания не могут заменить простые подробные данные. Всегда легче описывать факты, но многие предпочитают постараться дать толкования и описать их вместо фактов. Если объяснения основывается на догадках о том, каким образом реализован Emacs, они будут бесполезны. Как правило, не располагая фактами, мы не будем иметь настоящей информации об ошибке.

Предположим например, что вы набираете `C-x C-f /glorp/baz.ugh` ⟨RET⟩, обращаясь к файлу, который (как вы знаете) оказывается довольно большим, и Emacs печатает ‘`Сегодня я себя прекрасно чувствую`’. Наилучший способ описать эту ошибку — привести предложение, подобное предыдущему, так как это дает все факты.

Не считайте, что неполадка возникла из-за размера файла, и не говорите: “Когда я обращаюсь к большому файлу, Emacs печатает ‘`Сегодня я себя прекрасно чувствую`’”. Это именно то, что мы называем “объяснением на догадках”. Настолько же возможно, что ошибка произошла из-за того, что в имени файла имеется ‘`z`’. Если это так, то когда мы получили бы ваше описание, мы пытались бы решить проблему с “большим файлом”,

вероятно, без ‘z’ в его имени и не нашли бы никакой ошибки. Мы никак не могли бы догадаться, что должны были попробовать обратиться к файлу с буквой ‘z’ в имени.

С другой стороны, ошибка могла возникнуть из-за того, что файл начинается точно с 25 пробелов. Поэтому вы должны убедится в том, что проинформировали нас о точном содержании любого файла, который потребуется, чтобы воспроизвести ошибку. Что если ошибка встречается только тогда, когда вы до того набрали команду `C-x C-a`? Вот почему мы просим вас давать точную последовательность знаков, которые вы набрали со времени запуска Emacs.

Вы не должны даже говорить “обратиться к файлу” вместо `C-x C-f`, если не *уверены*, что нет различий в том, какая команда обращения используется. Аналогично, лучше сказать “после того как я набираю ⟨RET⟩ `A B C` ⟨RET⟩ `C-p`”, чем говорить “если у меня есть три буквы на строке”, если вы ввели текст именно таким способом.

Поэтому, пожалуйста, не предполагайте объяснения, когда сообщаете об ошибке. Если в действительности вы хотите *исследовать* эту проблему в отладчике и сообщить объяснения, которые уже есть нечто большее, чем догадки, это будет нам полезно — но пожалуйста, сообщите также и факты.

### 32.3.3 Форма отчета об ошибках

Лучший способ сообщить об ошибке — отправить электронное письмо сопроводителям Emacs по адресу `bug-gnu-emacs@gnu.org`. (Если вы хотите предложить изменение или улучшение, используйте этот же адрес.)

Если вы хотите читать отчеты об ошибках, вы можете найти их в группе новостей ‘`gnu.emacs.bug`’; помните однако, что как наблюдатель вы не должны критиковать ничего из того, что вы там увидите. Цель сообщений об ошибках — давать информацию сопроводителям Emacs. Наблюдатели приветствуются до тех пор, пока они не вмешиваются в это. В частности, некоторые отчеты содержат большие объемы данных; наблюдатели не должны на это жаловаться.

Пожалуйста, не посылайте сообщения об ошибках через сетевые новости; почта более надежна тем, что дает ваш точный адрес, который может понадобиться, чтобы попросить у вас больше информации.

Если вы не можете послать электронную почту, оправьте отчет об ошибке на бумаге или на машинночитаемом носителе по следующему адресу:

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

Мы не обещаем исправить ошибку; но если ошибка серьезная, или ужасная, или ее легко исправить, то вероятнее всего мы захотим это сделать.

Удобный способ послать отчет об ошибке в Emacs — использовать команду `M-x report-emacs-bug`. Она подготавливает буфер сообщения (см. Глава 26 [Посылка почты], с. 267) и автоматически вставляет *некоторую* важную информацию. Однако, она не может предоставить *всю* необходимую информацию; вы все равно должны прочитать приведенные ниже рекомендации и следовать им, тогда вы можете ввести остальные нужные сведения вручную перед отправкой сообщения.

Чтобы сопроводители могли исследовать ошибку, ваш отчет должен включать все следующие сведения:

- Номер версии Emacs. Без этого мы не сможем узнать, есть ли смысл искать эту ошибку в текущей версии GNU Emacs.

  Вы можете получить номер версии, набрав `M-x emacs-version` ⟨RET⟩. Если эта команда не работает, то у вас, вероятно, стоит не GNU Emacs, а что-то другое, поэтому вам лучше сообщать об ошибке по другому адресу.

- Тип используемой вами машины и название и номер версии операционной системы. `M-x emacs-version` ⟨RET⟩ предоставляет также и эту информацию. Скопируйте ее вывод из буфера '`*Messages*`', чтобы привести его полно и точно.
- Операнды, заданные команде `configure` во время установки Emacs.
- Полный перечень любых изменений, которые вы внесли в исходный текст Emacs. (У нас может не оказаться времени на исследование ошибки, если она не происходит в неизмененном Emacs. Но если вы внесли модификации и не сказали нам, вы просите нас пойти туда-не-знаю-куда принести то-не-знаю-что.

  Говорите об этих изменениях точно. Объяснения на английском недостаточно — пришлите контекстную заплату для них.

  Добавление своих файлов или перенос на другую машину тоже являются изменением исходного текста.
- Подробности о любых других отклонениях от стандартной процедуры установки GNU Emacs.
- Полный текст любых файлов, необходимых для воспроизведения ошибки.

  Если вы можете указать нам способ вызвать ошибку не обращаясь ни к каким файлам, пожалуйста, сделайте это. Это сделает отладку намного проще. Если для этого действительно требуются файлы, убедитесь, что вы приняли меры, чтобы мы могли увидеть их точное содержание. Например, часто может иметь значение, имеются ли пробелы в концах строк или перевод строки после последней строки в буфере (никого не должно волновать, оборвана ли последняя строка, но попробуйте рассказать это ошибке).
- Точные команды, которые мы должны напечатать, чтобы воспроизвести эту ошибку.

  Простой способ записать точно ввод, полученный Emacs, — написать файл сопровождения. Для этого выполните лисповское выражение

  ```
  (open-dribble-file "~/dribble")
  ```

  используя `M-:` или из буфера '`*scratch*`' сразу после старта Emacs. С этого момента Emacs копирует весь ваш ввод в указанный файл сопровождения, пока процесс Emacs не будет уничтожен.
- Для возможных ошибок изображения, тип терминала (значение переменной среды `TERM`), полная запись termcap для терминала из '`/etc/termcap`' (так как этот файл не идентичен для всех машин) и вывод, который Emacs в действительности посылал на терминал.

  Чтобы собрать этот вывод, выполните лисповское выражение

  ```
  (open-termscript "~/termscript")
  ```

  используя `M-:` или из буфера '`*scratch*`' сразу после старта Emacs. С этого момента Emacs копирует весь терминальный вывод также и в заданный файл терминального протокола до тех пор, пока процесс Emacs не будет уничтожен. Если ошибка произошла в процессе запуска Emacs, поместите это выражение в ваш файл '`.emacs`', таким образом, файл терминального протокола будет открываться, когда Emacs отобразит экран в первый раз.

  Следует предупредить: часто трудно, а иногда и невозможно исправить ошибку, зависящую от терминала, без доступа к терминалу того типа, который эту ошибку порождает.
- Описание поведения, которое вы наблюдали и сочли неправильным. Например, "Процесс Emacs получает фатальный сигнал" или "Получается вот такой текст, что, как я думаю, неправильно".[2]

[2] Писать отчеты нужно, конечно, на английском. Если вы не знаете английского сами, найдите того, кто вам поможет. *(Прим. переводчика)*

Конечно, если ошибка проявляется в том, что Emacs получает фатальный сигнал, ее нельзя не заметить. Но если ошибка — это неправильный текст, сопроводитель может не заметить, где же тут проблема. Зачем оставлять такую возможность?

Даже если проблема в том, что возникает фатальный сигнал, вы тем не менее должны сказать это явно. Предположим, произошло что-то странное, например, ваша копия исходного текста не синхронизирована или вы натолкнулись на ошибку в библиотеке Си в вашей системе. (Такое случалось!) Ваша копия может получить фатальный сбой, а наша может не получить. Если вы *скажете*, что ожидается крах, тогда, если наш Emacs не получит фатальный сбой, мы узнали бы, что ошибки не происходит. Если бы вы не сказали, то мы не узнали бы, происходит ли ошибка, — мы не смогли бы сделать никаких выводов из наших наблюдений.

- Если проявление неполадки — это сообщение Emacs об ошибке, то важно описать точный текст сообщения и след, показывающий, каким образом Лисп-программа в Emacs пришла к ошибке.

  Чтобы точно получить текст сообщения об ошибке, скопируйте его из буфера '`*Messages*`' в ваш отчет. Скопируйте его полностью, а не только часть.

  Чтобы получить след для ошибки, выполните лисповское выражение `(setq debug-on-error t)` перед тем, как произойдет ошибка (то есть вы должны выполнить это выражение, а потом повторить ситуацию, приводящую к ошибке). При возникновении ошибки это вызывает запуск отладчика Лиспа, который покажет вам след. Скопируйте текст следа отладчика в описание ошибки.

  Такое использование отладчика возможно только в том случае, если вы знаете, как вызвать ошибку снова. Если вы не можете воспроизвести ее, по крайней мере скопируйте полное сообщение об ошибке.

- Проверьте все программы, которые вы загрузили в среду Лиспа, включая ваш файл '`.emacs`', не устанавливают ли они какие-либо переменные, которые могут влиять на функционирование Emacs. Также посмотрите, случается ли ошибка в только что запущенном Emacs без загрузки файла '`.emacs`' (запустите Emacs с ключом `-q`, это предотвратит загрузку файла инициализации). Если в этом случае ошибка *не* встретится, вы должны сообщить точное содержание всех программ, которые вы должны загрузить в среду Лиспа для того, чтобы заставить ошибку проявиться.

- Если оказалось, что ошибка зависит от файла инициализации или других Лисп-программ, которые не являются частью стандартной системы Emacs, тогда вы должны удостовериться, что это не ошибка этих программ, обращаясь сперва к тем, кто их сопровождает. После того, как они подтвердят, что используют Emacs так, как ими предполагалось, они должны сообщить об этой ошибке.

- Если вы хотите сослаться на какое-то место в исходном тексте GNU Emacs, покажите эту строку кода и несколько строк контекста. Не давайте только номер строки.

  Номера строк в рабочих исходных текстах не совпадают с номерами в ваших файлах. Выяснение того, какой код находится в указанной строке в вашей версии, создаст дополнительную работу сопроводителям, и мы не сможем быть точно уверены.

- Дополнительная информация из отладчика Си, такого как GDB, может позволить нам найти проблему, возникающую на машине, которая нам недоступна. Если вы не умеете пользоваться GDB, пожалуйста, прочитайте руководство по нему — оно не слишком длинное, а GDB прост в применении. Вы можете найти дистрибутив GDB с интерактивным руководством в большинстве тех же мест, где вы можете получить дистрибутив Emacs. Чтобы запустить Emacs под GDB, вам нужно перейти в подкаталог '`src`', в котором был скомпилирован Emacs, и выполнить '`gdb emacs`'. Важно, чтобы каталог '`src`' был текущим, чтобы GDB считал в нем файл '`.gdbinit`'.

  Однако, при сборе дополнительных сведений вам придется подумать, если вы хотите, чтобы они показали, что же именно вызывает ошибку.

Например, многие посылают только след, но сам по себе он не очень полезен. Простой след с аргументами зачастую мало говорит о том, что происходит внутри GNU Emacs, потому что большинство перечисленных в нем аргументов являются указателями на лисповские объекты. Численные значения этих указателей не имеют никакой значимости; все, что играет роль — это содержимое объектов, на которые они указывают (и большая часть их содержимого — тоже указатели).

Чтобы предоставить полезную информацию, вам нужно показать значения лисповских объектов в лисповской записи. Сделайте это для каждой переменной, являющейся лисповским объектом, в нескольких нижних фреймах стека. Посмотрите в исходный текст, чтобы узнать, какие переменные являются лисповскими объектами, потому что отладчик считает их целыми числами.

Чтобы показать значение переменной в лисповской записи, сначала напечатайте ее значение, а потом используйте определенную пользователем команду GDB `pr`, которая напечатает лисповский объект в синтаксисе Лиспа. (Если вам приходится использовать другой отладчик, вызовите функцию `debug_print` с этим объектом в качестве аргумента.) Команда `pr` определена в файле '`.gdbinit`' и работает, только если вы отлаживаете запущенный процесс (не дамп памяти).

Чтобы ошибка в Лиспе останавливала Emacs и возвращала в GDB, установите контрольную точку в `Fsignal`.

Чтобы получить краткий список выполняющихся в данный момент лисповских функций, наберите команду GDB `xbacktrace`.

Если вы хотите исследовать аргументы лисповских функций, переходите вверх по стеку и каждый раз, когда попадаете во фрейм функции `Ffuncall`, вводите такие команды GDB:

```
p *args
pr
```

Чтобы напечатать первый аргумент, полученный этой функцией, используйте эти команды:

```
p args[1]
pr
```

Другие аргументы вы можете напечатать подобным же образом. Аргумент `nargs` функции `Ffuncall` говорит, сколько аргументов получила `Ffuncall`; это включает саму лисповскую функцию и ее аргументы.

Файл '`.gdbinit`' определяет другие команды, полезные для просмотра типов и содержимого лисповских объектов. Их имена начинаются с '`x`'. Эти команды работают на более низком уровне, чем `pr`, и менее удобны, но они могут работать, даже когда `pr` не работает, например при отладке дампа памяти, или когда Emacs получил фатальный сигнал.

- Если симптом ошибки состоит в том, что Emacs не отвечает, не предполагайте, что он "завис" — он может быть в бесконечном цикле. Чтобы выяснить, что же именно происходит, воспроизведите проблему под GDB и остановите Emacs, когда он не отвечает. (Если Emacs непосредственно использует X Windows, вы можете его остановить, набрав в GDB команду `C-z`.) Затем попробуйте выполнить один шаг с помощью '`step`'. Если Emacs завис, команда '`step`' не вернется. Если Emacs зациклился, '`step`' вернется.

  Если это покажет, что Emacs завис на системном вызове, остановите его снова и проверьте аргументы этого вызова. В вашем отчете укажите точно, где в исходном тексте находится этот системный вызов, и каковы его аргументы.

  Если Emacs попал в бесконечный цикл, пожалуйста, определите, где этот цикл начинается и где завершается. Простейший способ сделать это — использовать команду GDB '`finish`'. При каждом ее использовании Emacs продолжает выполнение, пока не

выйдет из одного фрейма стека. Продолжайте набирать ‘`finish`’, пока она не перестанет возвращаться — это будет означать, что бесконечный цикл находится в том фрейме стека, который вы только что пытались завершить.

Остановите Emacs снова и вводите последовательно ‘`finish`’, пока вы не *вернетесь* в тот же фрейм. Тогда используйте ‘`next`’, чтобы пройти сквозь этот фрейм. С помощью пошагового выполнения вы увидите, где цикл начинается и где заканчивается. Также, пожалуйста, проверьте используемые в цикле данные и попытайтесь выяснить, почему цикл не завершается, когда нужно. Включите все эти сведения в ваш отчет об ошибке.

Вот некоторые вещи, которые необязательны в отчете об ошибке:

- Исчерпывающее описание условий возникновения и не возникновения ошибки — это не нужно для воспроизводимых ошибок.

  Нередко люди, сталкивающиеся с ошибкой, тратят много времени на выяснение того, какие изменения во входном файле заставят ошибку исчезнуть, а какие не затронут ее.

  Это зачастую требует много времени и не столь полезно, потому что мы будем искать ошибку, запуская один пример в отладчике с контрольными точками, а не чистой дедукцией из серии примеров. Вы также можете сэкономить время, не выискивая дополнительные примеры.

  Разумеется, если вы можете найти более простой пример для сообщения *вместо* изначального, это будет удобнее. Ошибки в выводе будет легче обнаружить, работа в отладчике займет меньше времени, и так далее.

  Однако, упрощение не жизненно важно; если вы не можете это сделать, или у вас нет времени пытаться, пожалуйста, сообщите об ошибке с первоначальным тестовым примером.

- След системных вызовов работы Emacs.

  Следы системных вызовов очень полезны для некоторых особых видов отладки, но чаще всего они дают мало полезной информации. Поэтому странно, что многие, как кажется, считают, что *именно* оправка следа системных вызовов — это и есть отчет об фатальном сбое. Возможно, эта привычка сформировалась из опыта отладки программ, к которым нет исходного текста, или в которых нет отладочных символов.

  В большинстве программ след вызовов подпрограмм гораздо, гораздо информативнее следа системных вызовов. Даже в Emacs простой след подпрограмм обычно более информативен, хотя чтобы дать все сведения, вы должны дополнить след, показав значения переменных и напечатав их как лисповские объекты с помощью `pr` (смотрите выше).

- Заплата для ошибки.

  Заплата для ошибки полезна, если это хорошая заплата. Но не опускайте в отчете об ошибке другую необходимую информацию, такую как тестовый пример, в предположении, что заплаты достаточно. Мы можем увидеть недочеты в вашей заплате и решить проблему другим способом или не понять ее совсем. И если мы не можем понять, какую ошибку предполагает исправить ваша заплата, или почему она должна быть улучшением, мы не должны ее устанавливать.

- Предположения о том, чем является эта ошибка, или от чего она зависит.

  Такие предположения обычно неверны. Даже эксперты не могут правильно догадываться о таких вещах, не применив сначала отладчик для выяснения фактов.

### 32.3.4 Отправка заплат для GNU Emacs

Если хотите написать исправления ошибок или улучшения для GNU Emacs, это очень нам поможет. Когда вы посылаете ваши изменения, пожалуйста, следуйте этим рекомендациям, чтобы сопроводителям было легче их использовать. Если вы не следуете им, ваши

изменения тем не менее могут быть полезны, но использование их потребует дополнительной работы. Сопровождение GNU Emacs — это в лучших обстоятельствах много работы, и нам будет трудно справляться, если вы не сделаете все от вас зависящее, чтобы нам помочь.

- Посылайте вместе с вашими изменениями объяснение, какую проблему они решают, или какое улучшение они приносят. Для исправления ошибки просто включите описание этой ошибки и объясните, как это изменение ее исправляет.

  (Ссылка на сообщение об ошибке не так хороша, как включение самого сообщения, потому что тогда нам придется искать, и возможно, мы его уже удалили, если ошибка уже исправлена.)

- Всегда включайте хорошее описание проблемы, которую, как вы думаете, вы решили. Мы должны убедиться, что изменение правильно, перед тем как его устанавливать. Даже если оно правильно, у нас могут быть трудности с его пониманием, если сами мы не можем воспроизвести эту ошибку.

- Включайте все необходимые комментарии, которые в будущем помогут понять цель этого изменения другим людям, читающим исходный текст.

- Не перемешивайте изменения, сделанные по разным причинам. Посылайте их *раздельно*.

  Если вы делаете два изменения по разным причинам, мы можем не захотеть устанавливать их оба. Возможно, мы захотим установить только одно. Если вы пошлете их перемешанными в одном наборе заплат, мы должны будем делать дополнительную работу, чтобы их распутать — чтобы понять, какие части изменения служат каким целям. Если у нас нет на это времени, мы можем проигнорировать ваши изменения целиком.

  Если вы посылаете каждое изменение, как только вы его написали, с отдельным объяснением, то два этих изменения не перепутаются, и мы сможем подробно рассмотреть каждое, не затрачивая дополнительных усилий на их разделение.

- Посылайте каждое изменение, как только вы его закончили. Иногда люди считают, что помогают нам, накапливая много изменений и посылая их все вместе. Как объяснено выше, это наихудшее, что вы можете сделать.

  Поскольку вы должны посылать каждое изменение отдельно, вы также могли бы посылать их немедленно. Это даст нам возможность установить ваше изменение сразу же, если это важно.

- Для создания заплат используйте ‘`diff -c`’. Заплаты без контекста трудно установить надежно. Более того, их трудно изучать; мы должны всегда изучать заплату, чтобы решить, нужно ли ее устанавливать. Объединенный формат лучше бесконтекстного, но его не так легко читать, как формат ‘`-c`’.

  Если у вас есть GNU diff, используйте при создании заплат в коде на Си команду ‘`diff -c -F'^[_a-zA-Z0-9$]+ *('`’. Это покажет каждое имя функции, в которой находится изменение.

- Избегайте любой неоднозначности в том, что является старой версией, а что новой. Пожалуйста, задавайте старую версию первым аргументом diff, а новую версию вторым. И пожалуйста, давайте той или иной версии такое имя, которое показывало бы, старый это файл или ваш измененный.

- Пишите для ваших изменений журнальные записи. Это избавит нас от дополнительной работы по их написанию и поможет разъяснить ваши изменения, чтобы мы могли их понять.

  Цель журнала изменений — показать людям, где искать измененные места. Поэтому вы должны точно указывать, какие функции вы изменили; в больших функциях также часто полезно указывать, где именно в этой функции было сделано изменение.

С другой стороны, когда вы показали людям, где найти изменение, вам не нужно объяснять его цель в журнале изменений. Таким образом, если вы добавили новую функцию, все, что вы должны сказать, — это что она новая. Если вы чувствуете, что цель этого требует разъяснения, то это, скорее всего, так и есть, но помещайте разъяснение в комментариях в коде. Там оно будет более полезно.

Пожалуйста, прочитайте файлы ‘`ChangeLog`’ в каталогах ‘`src`’ и ‘`lisp`’, чтобы понять, какого сорта сведения в них нужно писать, и освоить используемый нами стиль. Если вы хотите, чтобы в строке заголовка, показывающей автора изменения, появилось ваше имя, пошлите нам этот заголовок. См. Раздел 22.12 [Change Log], с. 224.

- Когда вы пишете исправление, помните, что мы не можем установить изменение, которое могло бы нарушить работу на других системах. Пожалуйста, подумайте об эффекте, который может иметь ваше изменение, если его скомпилировать на системе другого типа.

  Иногда люди присылают исправления, которые *могут* быть улучшением в общем — но убедиться в этом трудно. Установить такие изменения трудно, потому что мы должны изучить их очень внимательно. Конечно, хорошее пояснение рассуждений, благодаря которым вы пришли к осознанию правильности такого изменения, может помочь убедить нас.

  Самые безопасные изменения — это изменения в конфигурационных файлах для конкретной машины. Они безопасны, потому что не могут повлечь за собой новых ошибок на других машинах.

  Пожалуйста, помогите нам справляться с нагрузкой, разрабатывая заплату таким образом, чтобы было ясно, что ее установка безопасна.

## 32.4 Содействие в разработке Emacs

Если вы хотели бы помочь в тестировании выпусков Emacs, чтобы убедиться, что они работают правильно, или если вы хотели бы работать над улучшением Emacs, пожалуйста, свяжитесь с сопроводителями по адресу `bug-gnu-emacs@gnu.org`. Испытатель должен быть готов как к исследованию ошибок, так и к их описанию. Если вы желаете улучшить Emacs, пожалуйста, спросите о предлагающихся проектах или предложите свои идеи.

Если вы уже написали расширение, пожалуйста, сообщите нам об этом. Если вы еще не начали работу, будет полезным связаться с `bug-gnu-emacs@gnu.org` до того, как вы начнете; тогда мы сможем посоветовать, как сделать так, чтобы ваши расширения лучше стыковались с остальной частью Emacs.

## 32.5 Как получить помощь по GNU Emacs

Если вы нуждаетесь в помощи по установке, использованию или модификации GNU Emacs, у вас есть два способа получить ее:

- Послать сообщение в список рассылки `help-gnu-emacs@gnu.org` или опубликовать ваш запрос с группе новостей `gnu.emacs.help`. (Этот список рассылки и группа новостей сообщаются между собой, поэтому безразлично, что именно вы будете использовать.)
- Посмотреть в каталоге услуг, нет ли кого-нибудь, кто сможет вам помочь за плату. Этот каталог находится в файле с именем ‘`etc/SERVICE`’ в дистрибутиве Emacs.

# Приложение A Аргументы командной строки

GNU Emacs обрабатывает аргументы командной строки, которые запрашивают различные действия при вызове Emacs. Они существуют для совместимости с другими редакторами и для сложных процедур. Мы не рекомендуем использовать их для обычного редактирования.

Аргументы, начинающиеся со знака '`-`', называются *ключами*. Остальные аргументы задают файлы, к которым нужно обратиться. Emacs обращается к указанным файлам во время запуска. Имя файла, заданное в командной строке последним, становится текущим буфером; другие файлы также присутствуют, но в других буферах. Как обычно, особый аргумент '`-`' говорит, что все последующие аргументы являются именами файлов, а не ключами, даже если начинаются на '`-`'.

Командные ключи Emacs могут задавать многие вещи, например размер и положение X-окна, используемого Emacs, его цвета и так далее. Некоторые ключи поддерживают продвинутое использование, это, например, запуск лисповских функций для файлов в пакетном режиме. Разделы в этой главе описывают доступные ключи, расположенные в соответствии с их назначением.

Есть два способа записи ключей: короткие формы, начинающиеся с одного знака '`-`', и длинные формы, начинающиеся с '`-`'. Например, '`-d`' — это короткая форма, а '`-display`' — соответствующая длинная форма.

Длинные формы с '`-`' легче запомнить, но дольше печатать. Однако, вы не обязаны писать имя ключа полностью, достаточно любого однозначного сокращения. Когда длинный ключ принимает аргумент, вы можете использовать для разделения имени ключа и аргумента либо пробел, либо знак равенства. Таким образом, вы можете написать как '`-display sugar-bombs:0.0`', так и '`-display=sugar-bombs:0.0`'. Мы рекомендуем использовать знак равенства, так как он более четко показывает взаимосвязь, и в приведенных ниже таблицах всегда используется знак равенства.

Большинство ключей указывают, как инициализировать Emacs, или устанавливают параметры для всего сеанса Emacs. Мы называем их *ключами запуска*. Немногие ключи указывают, что нужно сделать: например, загрузить библиотеки, вызвать функции или выйти из Emacs. Такие ключи называют *ключами действия*. Их и имена файлов вместе называют *аргументами действия*. Emacs обрабатывает все аргументы действия в том порядке, в котором они были записаны.

## A.1 Аргументы действия

Вот таблица аргументов и ключей действия:

'*файл*' Обратиться к *файлу* с помощью `find-file`. См. Раздел 14.2 [Обращение], с. 106.

'+*номер-строки файл*'
Обратиться к *файлу* с помощью `find-file`, а затем перейти в нем к строке с номером *номер-строки*.

‘`-l` *файл*’
‘`-load=`*файл*’
: Загрузить лисповскую библиотеку с именем *файл* с помощью функции `load`. См. Раздел 23.7 [Библиотеки Лиспа], с. 253. Библиотека может находиться либо в текущем каталоге, либо в пути поиска библиотек Emacs, как он задан переменной `EMACSLOADPATH` (см. Раздел A.5.1 [Общие переменные], с. 388).

‘`-f` *функция*’
‘`-funcall=`*функция*’
: Вызвать лисповскую *функцию* без аргументов.

‘`-eval` *выражение*’
: Вычислить лисповское *выражение*.

‘`-insert=`*файл*’
: Вставить содержимое *файла* в текущий буфер. Это похоже на действие `M-x insert-file`. См. Раздел 14.10 [Разнообразные действия с файлами], с. 132.

‘`-kill`’ Прекратить работу Emacs без подтверждения.

Файл инициализации может получить доступ к значениям аргументов действия через список в переменной `command-line-args`. Файл инициализации может перекрыть обычную обработку аргументов действия или определить новые путем чтения и установки этой переменной.

## A.2 Ключи запуска

Ключи запуска задают параметры для данного сеанса Emacs. В этом разделе описаны более общие ключи запуска; некоторые другие ключи, относящиеся к X Windows, вводятся в следующих разделах.

Некоторые ключи запуска влияют на процесс загрузки файлов инициализации. Обычно Emacs загружает ‘`site-start.el`’, если он существует, затем ваш собственный файл инициализации ‘`~/.emacs`’, если он существует, и наконец, ‘`default.el`’, если он существует; определенные ключи запрещают загрузку некоторых из этих файлов или заменяют их другими файлами.

‘`-t` *устройство*’
‘`-terminal=`*устройство*’
: Использовать *устройство* в качестве терминала для ввода и вывода.

‘`-d` *дисплей*’
‘`-display=`*дисплей*’
: Использовать систему X Windows и открыть начальный фрейм Emacs на дисплее с заданным именем.

‘`-nw`’
‘`-no-windows`’
: Не общаться непосредственно с X Windows, не обращая внимания на переменную среды `DISPLAY`, даже если она установлена.

‘-batch’
‘-batch’ Запустить Emacs *пакетном режиме*, что означает, что редактируемый текст не отображается, а стандартные команды терминала для прерывания, такие как C-z и C-c, продолжают иметь свое обычное значение. Emacs в пакетном режиме выводит в стандартный поток ошибок stderr только то, что в нормальном режиме выводилось бы программами в эхо-область.

Пакетный режим используется для запуска программ, написанных на языке Emacs Lisp, из сценариев командного интерпретатора, Make-файлов и так далее. Обычно при этом также используются ключи ‘-l’ или ‘-f’, чтобы запустить Лисп-программу для пакетной обработки.

Ключ ‘-batch’ подразумевает ‘-q’ (не загружать файл инициализации). Он также заставляет Emacs прекратить работу после того, как обработаны все командные ключи. Кроме того, самосохранение производится только в тех буферах, в которых оно было затребовано явно.

‘-q’
‘-no-init-file’
Не загружать ваш файл инициализации Emacs, ‘~/.emacs’, а также ‘default.el’.

‘-no-site-file’
Не загружать файл ‘site-start.el’. Ключи ‘-q’, ‘-u’ и ‘-batch’ не влияют на загрузку этого файла — только этот ключ блокирует ее.

‘-u *пользователь*’
‘-user=*пользователь*’
Загрузить файл инициализации Emacs, принадлежащий *пользователю*, ‘~*пользователь*/.emacs’, вместо вашего собственного.

‘-debug-init’
Включить отладчик Emacs Lisp для обработки ошибок в файле инициализации.

‘-unibyte’
Делать практически все с использованием однобайтных буферов и строк. Все буферы и строки будут однобайтными, если вы (или Лисп-программа) явно не запросите многобайтный буфер или строку. Установка переменной среды EMACS_UNIBYTE имеет тот же эффект.

‘-multibyte’
Подавить действие переменной EMACS_UNIBYTE, чтобы Emacs по умолчанию использовал многобайтные знаки, как обычно.

## A.3 Пример аргументов командной строки

Здесь приведен пример использования Emacs с аргументами и ключами. Он предполагает, что у вас есть программа на Лиспе, называемая ‘hack-c.el’, которая будучи загруженной выполняет некоторые полезные действия над текущим буфером, предположительно программой на Си.

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

Это говорит Emacs обратиться к файлу ‘foo.c’, загрузить ‘hack-c.el’ (которая производит изменения в файле, к которому вы обратились), сохранить ‘foo.c’ (заметьте, что save-buffer — это функция, которая привязана к C-x C-s), а затем вернуться в оболочку (из-за ‘-batch’). Ключ ‘-batch’ также гарантирует, что не будет проблем с перенаправлением вывода в файл ‘log’, так как Emacs не будет предполагать, что он должен работать с терминалом.

## A.4 Возврат в Emacs с аргументами

Вы можете задать для Emacs аргументы действия, когда вы возвращаетесь в него после приостановки. Чтобы подготовиться к этому, поместите в ваш файл '`.emacs`' следующий код (см. Раздел 31.2.3 [Ловушки], с. 349):

```
(add-hook 'suspend-hook 'resume-suspend-hook)
(add-hook 'suspend-resume-hook 'resume-process-args)
```

Для дальнейшей подготовки вы должны выполнить сценарий оболочки, '`emacs.csh`' (если вы используете в качестве оболочки csh) или '`emacs.bash`' (если вы пользуетесь bash). Эти сценарии определяют псевдоним с именем `edit`, который будет возвращать в Emacs, передавая ему новые аргументы командной строки, например файлы для редактирования.

Когда вы возвращаетесь в Emacs, правильно работают только аргументы действия. Аргументы запуска не распознаются — так или иначе, их слишком поздно исполнять.

Обратите внимание, возврат в Emacs (с аргументами или без) должен производиться из оболочки, являющейся родительским процессом задания Emacs. Именно поэтому `edit` — это псевдоним, а не программа или сценарий. Невозможно реализовать команду возврата, которую можно было бы запускать из подзадач этой оболочки; например, невозможно определить команду, которую можно было бы сделать значением переменной `EDITOR`. Следовательно, команда возврата не эквивалентна по возможностям серверу Emacs (см. Раздел 30.3 [Сервер Emacs], с. 330).

Эти псевдонимы используют сервер Emacs, если оказалось, что он уже запущен. Однако, они не могут определить это абсолютно точно. Они могут полагать, что сервер все еще запущен, Emacs, потому что файл '`/tmp/.esrv...`' все так же существует, тогда как в действительности вы уничтожили тот Emacs. Если такое происходит, найдите этот файл и удалите его.

## A.5 Переменные среды

В этом приложении описано, как Emacs использует переменные среды. Переменная среды — это строка, передаваемая операционной системой в Emacs, а все множество переменных среды называется средой. Имена переменных среды регистрозависимы, и для них принято использовать только заглавные буквы.

Так как переменные среды исходят от операционной системы, общего способа для их установки нет; он зависит от используемой вами операционной системы и в особенности от оболочки. Например, так можно установить переменную среды `ORGANIZATION` в значение '`не особая`' с использованием bash:

```
export ORGANIZATION="не особая"
```

а так это можно сделать в csh или tcsh:

```
setenv ORGANIZATION "не особая"
```

Когда Emacs настроен для использования оконной системы X, он наследует большое число переменных среды из библиотеки X. Для дальнейшей информации смотрите документацию по X Windows.

### A.5.1 Общие переменные

`AUTHORCOPY`
: Имя файла, используемое для архива статей, посланных с помощью пакета GNUS.

`CDPATH`
: Используется командой `cd` для поиска указанного вами каталога, если вы задали относительное имя.

`DOMAINNAME`
: Имя домена Internet, где находится машина, на которой запускается Emacs. Используется пакетом GNUS.

`EMACS_UNIBYTE`
: Определение этой переменной среды указывает Emacs производить почти все процедуры над однобайтными буферами и строками. Это эквивалентно использованию в командной строке ключа '`-unibyte`' при каждом вызове. См. Раздел A.2 [Ключи запуска], с. 386.

`EMACSDATA`
: Используется для начальной установки переменной `data-directory`, необходимой для нахождения не зависящих от архитектуры компьютера файлов, поставляемых с Emacs. Установка этой переменной перекрывает значение, заданное в файле '`paths.h`' во время сборки Emacs.

`EMACSLOADPATH`
: Разделенный двоеточиями перечень каталогов, из которых будут загружаться файлы Emacs Lisp. Установка этой переменной перекрывает значение, заданное в файле '`paths.h`' во время сборки Emacs.

`EMACSLOCKDIR`
: Каталог, в котором Emacs будет помещать файлы блокировки; эти файлы используются для защиты от одновременного редактирования одних и тех же файлов разными пользователями. Установка этой переменной перекрывает значение, заданное в файле '`paths.h`' во время сборки Emacs.

`EMACSPATH`
: Каталог, где находятся бинарные файлы, необходимые только Emacs. Установка этой переменной перекрывает значение, заданное в файле '`paths.h`' во время сборки Emacs.

`ESHELL` Используется для режима shell, перекрывает переменную среды `SHELL`.

`HISTFILE` Имя файла, в котором сохраняются команды оболочки от одного входа в систему до другого. По умолчанию эта переменная равна '`~/.history`', если вы используете в качестве оболочки (t)csh, '`~/.bash_history`', если вы пользуетесь bash, '`~/.sh_history`', если вы пользуетесь ksh, и '`~/.history`' в остальных случаях.

`HOME` Положение файлов пользователя в дереве каталогов; используется для раскрытия имен файлов, начинающихся с тильды ('`~`'). В MS-DOS это по умолчанию каталог, откуда Emacs был запущен, исключая завершающий '`/bin`', если он присутствует.

`HOSTNAME` Имя машины, на которой запущен Emacs.

`INCPATH` Разделенный двоеточиями перечень каталогов. Используется пакетом `complete` для поиска файлов.

`INFOPATH` Разделенный двоеточиями перечень каталогов, содержащих Info-файлы. Установка этой переменной перекрывает значение, заданное в файле '`paths.el`' во время сборки Emacs.

`LANG`
`LC_ALL`
`LC_CTYPE` Предпочитаемый пользователем регион. Имя региона, содержащее '`8859-`*n*', '`8859_`*n*' или '`8859`*n*', где *n* находится между 1 и 4, автоматически задает при запуске Emacs языковую среду '`Latin-`*n*'. Если *n* равно 9, это задает '`Latin-5`'.

`LOGNAME` Имя пользователя. Смотрите также `USER`.

`MAIL` Имя системного входного почтового ящика для этого пользователя.

`MAILRC` Имя файла почтовых псевдонимов. По умолчанию это '`~/.mailrc`'.

`MH` Имя файла установок системы mh. По умолчанию это '`~/.mh_profile`'.

`NAME` Действительное имя пользователя.

`NNTPSERVER`
Имя сервера новостей. Используется пакетами mh и GNUS.

`ORGANIZATION`
Название организации, к которой вы принадлежите. Используется пакетом GNUS для установки поля 'Organization:' в заголовках ваших сообщений.

`PATH` Разделенный двоеточиями перечень каталогов, в которых находятся исполняемые файлы. (В MS-DOS этот перечень разделяется точками с запятой.) Это значение используется для установки переменной Emacs Lisp `exec-path`; лучше использовать именно эту переменную.

`PWD` Если эта переменная установлена, то она задает каталог по умолчанию при запуске Emacs.

`REPLYTO` Если эта переменная установлена, то она задает первоначальное значение переменной `mail-default-reply-to`. См. Раздел 26.2 [Заголовки сообщений], с. 268.

`SAVEDIR` Имя каталога, в котором по умолчанию сохраняются статьи новостей. Используется пакетом GNUS.

`SHELL` Имя интерпретатора, используемого для разбора и выполнения программ, запускаемых из Emacs.

`TERM` Название терминала, на котором запущен Emacs. Вы обязаны установить эту переменную, если Emacs запускается не в пакетном режиме. В MS-DOS по умолчанию принимается значение '`internal`', что означает встроенную эмуляцию терминала, которая управляет собственным дисплеем данной машины.

`TERMCAP` Имя файла библиотеки termcap, описывающей, каким образом можно программировать терминал, заданный переменной `TERM`. По умолчанию это '`/etc/termcap`'.

`TMPDIR` Используется пакетом Emerge в качестве префикса для временных файлов.

`TZ` Задает текущий часовой пояс и, возможно, информацию о сезонном переводе времени. В MS-DOS значение по умолчанию основывается на коде страны; подробности смотрите в файле '`msdos.c`'.

`USER` Имя пользователя. Смотрите также `LOGNAME`. В MS-DOS, значением по умолчанию считается '`root`'.

`VERSION_CONTROL`
Используется для первоначальной установки переменной `version-control` (см. Раздел 14.3.1.1 [Имена резервных файлов], с. 110).

### A.5.2 Другие различные переменные

Следующие переменные используются лишь в определенных конфигурациях:

`COMSPEC` В MS-DOS, имя используемого командного интерпретатора. Эта переменная задает значение по умолчанию для переменной среды `SHELL`.

`NAME` В MS-DOS значение этой переменной равно значению переменной `USER`.

`TEMP`
`TMP` В MS-DOS это задает имя каталога для хранения временных файлов.

`EMACSTEST`
В MS-DOS это задает файл протокола операций со внутренним эмулятором терминала. Это бывает полезно для составления сообщений об ошибках.

`EMACSCOLORS`
Используется в системах MS-DOS для раннего установления экранных цветов, чтобы при запуске Emacs экран не промелькивал в цветах по умолчанию. Значением этой переменной должен быть состоящий из двух знаков код цвета букв (первый знак) и цвета фона (второй знак) для начертания по умолчанию. Каждый знак должен быть шестнадцатиричным кодом желаемого цвета на стандартном текстовом дисплее PC.

Дисплей PC обычно поддерживает только восемь цветов фона. Однако, Emacs переключает дисплей DOS в такой режим, где для фона можно использовать все 16 цветов, так что на самом деле работают все четыре бита цвета фона.

`WINDOW_GFX`
Используется при инициализации оконной системы Sun.

## A.6 Указание имени дисплея

Переменная среды `DISPLAY` сообщает всем X-клиентам, включая Emacs, где нужно отображать их окна. Ее значение создается автоматически при обычных обстоятельствах, когда вы включили X-сервер и запускаете задачи локально. Но иногда вам может понадобиться указать дисплей самим; например, если вы заходите на удаленную систему и хотите запускать программы-клиенты удаленно, но с отображением на вашем локальном экране.

В случае с Emacs, основная причина, по которой люди изменяют дисплей по умолчанию, состоит в том, чтобы позволить им зайти на другую систему и запустить в ней Emacs, но отображать его окна на локальном терминале. Ситуация, когда вам может понадобиться зайти на другую систему, возникает, к примеру, когда там находятся файлы, которые вы собрались редактировать, или исполняемый файл Emacs.

Переменная среды `DISPLAY` имеет следующий синтаксис: '*машина*`:`*дисплей*`.`*экран*', где *машина* — это имя машины, где запущен сервер системы X Windows, *дисплей* — это произвольно присваиваемое число, отличающее ваш сервер (X-терминал) от других серверов на той же машине, а *экран* — это редко используемое поле, позволяющее X-серверу контролировать несколько терминальных экранов. Точка и поле *экран* необязательны. Если поле *экран* включают, оно обычно равно нулю.

Для примера, если ваша машина называется '`glasperle`', и ваш сервер — первый (и, возможно, единственный) сервер в конфигурации, то переменная `DISPLAY` равна '`glasperle:0.0`'.

Вы можете указать имя дисплея явно при запуске Emacs, либо изменив переменную `DISPLAY`, либо с помощью ключа '`-d` *дисплей*' или '`-display=`*дисплей*'. Вот пример:

```
emacs -display=glasperle:0 &
```

Вы можете подавить прямое использование X Windows с помощью ключа '`-nw`'. Это ключ запуска. Он велит Emacs отображать, используя на управляющем терминале обычные знаки ASCII.

Иногда установки защиты доступа не позволяют программе с удаленной системы выводить на ваш локальный дисплей. В таком случае, попытка запуска Emacs даст подобное сообщение:

```
Xlib:  connection to "glasperle:0.0" refused by server
```

Вы, вероятно, сможете справиться с этой проблемой, предоставив при помощи команды `xhost` на вашей локальной системе разрешение на доступ для удаленной машины.

## A.7 Ключи для задания шрифта

По умолчанию Emacs отображает текст шрифтом с именем '`9x15`', в котором каждый знак имеет ширину девять и высоту пятнадцать пикселей. Вы можете указать в командной строке другой шрифт с помощью ключа '`-fn` *имя*'.

'`-fn` *имя*' Использовать в качестве шрифта по умолчанию шрифт с указанным *именем*.

'`-font=`*имя*'
'`-font`' — это синоним '`-fn`'.

В X Windows каждый шрифт обладает длинным именем, состоящим из одиннадцати слов или чисел, разделенных дефисами. У некоторых шрифтов кроме этого есть и более короткие псевдонимы — '`9x15`' один из таких псевдонимов. Emacs допускает оба вида имен. Вы можете использовать шаблоны имен шрифтов; тогда Emacs позволит X Windows выбрать один из шрифтов, соответствующих шаблону. Вот пример, в котором задается шрифт, чьим псевдонимом, как оказывается, является '`6x13`':

```
emacs -fn "-misc-fixed-medium-r-semicondensed-13-*-*-*-c-60-iso8859-1" &
```

Вы также можете указать шрифт в вашем файле '`.Xdefaults`':

```
emacs.font: -misc-fixed-medium-r-semicondensed-13-*-*-*-c-60-iso8859-1
```

Полное имя шрифта имеет следующий формат:

*-производитель-семейство-насыщенность-наклон-тип_ширины-стиль...*
*...-пиксели-высота-гориз-верт-пропорциональность-ширина-кодировка*

*семейство* Название семейства шрифтов — например, '`courier`'.

*насыщенность*
Обычно это '`bold`', '`medium`' или '`light`'. В именах некоторых шрифтов здесь могут появляться и другие слова.

*наклон* Это '`r`' (романский), '`i`' (курсив), '`o`' (наклонный), '`ri`' (обратный курсив) или '`ot`' (другое).

*тип_ширины*
Обычно это '`condensed`', '`extended`', '`semicondensed`' или '`normal`'. В именах некоторых шрифтов здесь могут появляться и другие слова.

*стиль* Это необязательное имя дополнительного стиля. Обычно оно пусто — в большинстве длинных имен шрифтов на этом месте стоят два дефиса подряд.

*пиксели* Высота шрифта в пикселях.

*высота* Это высота шрифта на экране, измеряемая в десятых долях принтерной точки — приблизительно 1/720 дюйма. Другими словами, это высота шрифта в точках, умноженная на десять. Для данного вертикального разрешения *высота* и *пиксели* пропорциональны; поэтому обычно указывают лишь одно из двух, оставляя второе как '`*`'.

*гориз* Горизонтальное разрешение экрана, для которого предназначен этот шрифт, в точках на дюйм.

*верт* Вертикальное разрешение экрана, для которого предназначен этот шрифт, в точках на дюйм. Обычно разрешение шрифтов, установленных в вашей системе соответствует разрешению вашего экрана; следовательно, как правило, для этого поля и поля *гориз* вы пишете '`*`'.

*пропорциональность*
: Это может быть ‘`m`’ (monospace), ‘`p`’ (proportional) или ‘`c`’ (character cell). Emacs может использовать шрифты ‘`m`’ и ‘`c`’.

*ширина*
: Это средняя ширина знака, в пикселях, умноженных на десять.

*кодировка*
: Изображаемый этим шрифтом набор знаков. Обычно вам нужно использовать ‘`iso8859-5`’.

Используйте только равноширинные шрифты — то есть такие, в которых все знаки имеют одинаковую ширину; Emacs пока не умеет правильно отображать шрифты переменной ширины. Любой шрифт с буквой ‘`m`’ или ‘`c`’ в поле длинного имени *пропорциональность* — равноширинный. Вот как использовать программу `xlsfonts` для получения списка всех доступных на вашей системе равноширинных шрифтов:

```
xlsfonts -fn '*x*' | egrep "^[0-9]+x[0-9]+"
xlsfonts -fn '*-*-*-*-*-*-*-*-*-*-*-*-m*'
xlsfonts -fn '*-*-*-*-*-*-*-*-*-*-*-*-c*'
```

Чтобы просмотреть, как выглядит конкретный шрифт, используйте команду `xfd`. Например,

```
xfd -fn 6x13
```

покажет все знаки шрифта ‘`6x13`’.

Во время работы Emacs вы можете поменять шрифт текущего фрейма (см. Раздел 17.10 [Параметры фрейма], с. 153) или определенного сорта текста (см. Раздел 17.13 [Начертания], с. 155).

## A.8 Параметры для задания цветов

Если у вас есть цветной дисплей, вы можете указать, какие цвета должны использоваться для различных частей экрана Emacs. Чтобы узнать, какие цвета доступны в вашей системе, посмотрите файл ‘`/usr/lib/X11/rgb.txt`’. Если вы не указали цвета, по умолчанию фон будет белым, а все остальное — черным. На монохромном дисплее, текст будет черным, фон будет белым, а рамка — серой, если это поддерживается дисплеем.

Вот список ключей для задания цветов:

‘`-fg` *цвет*’
‘`-foreground-color=`*цвет*’
: Задает цвет текста.

‘`-bg` *цвет*’
‘`-background-color=`*цвет*’
: Задает цвет фона.

‘`-bd` *цвет*’
‘`-border-color=`*цвет*’
: Задает цвет рамки в X-окна.

‘`-cr` *цвет*’
‘`-cursor-color=`*цвет*’
: Задает цвет курсора, показывающего положение точки.

‘`-ms` *цвет*’
‘`-mouse-color=`*цвет*’
: Задает цвет указателя мыши, когда он находится во фрейме Emacs.

‘`-r`’
‘`-reverse-video`’
: Инверсное изображение — обменять цвета текста и фона.

Например, чтобы получить коралловый указатель мыши и синевато-серый текстовый курсор, введите:

```
emacs -ms coral -cr 'slate blue' &
```

Вы можете инвертировать цвета текста и фона с помощью ключа '`-r`' или через ресурс X Windows '`reverseVideo`'.

## A.9 Параметры геометрии окна

Ключ '`-geometry`' управляет положением начального фрейма Emacs. Формат для указания геометрии окна такой:

'`-g` *ширинахвысота*`{+-}`*сдвиг-по-x*`{+-}`*сдвиг-по-y*'
: Задает размеры окна, *ширину* и *высоту* (измеряемые в знаковых столбцах и строках), и положение *сдвиг-по-x* и *сдвиг-по-y* (измеряемые в пикселях).

'`-geometry=`*ширинахвысота*`{+-}`*сдвиг-по-x*`{+-}`*сдвиг-по-y*'
: Это другой способ написать то же самое.

`{+-}` означает плюс или минус. Знак плюс перед *сдвиг-по-x* говорит, что это расстояние от левого края экрана; знак минус говорит, что это расстояние от правого края. Знак плюс перед *сдвиг-по-y* обозначает, что это расстояние от верхнего края экрана, а знак минус — что это расстояние от нижнего края. Сами значения *сдвиг-по-x* и *сдвиг-по-y* могут быть положительными или отрицательными, но это не меняет их смысл, только направление.

Emacs интерпретирует геометрию в тех же единицах, что и `xterm`. *ширина* и *высота* измеряются в знаках, так что фрейм с крупным шрифтом будет больше, чем фрейм с мелким шрифтом. *Сдвиг-по-x* и *сдвиг-по-y* измеряются в пикселях.

Так как последние две строки фрейма занимают строка режима и эхо-область, высота начального текстового окна на 2 меньше высоты, в указанной вами геометрии. В версиях Emacs, не использующих X-toolkit, полоска меню также занимает одну строку из указанного числа.

Вы не обязаны писать все поля при задании геометрии.

Если вы опустите и *сдвиг-по-x*, и *сдвиг-по-y*, программа управления окнами сама решит, где размещать фрейм Emacs, возможно, позволяя вам поместить его с помощью мыши. Например, '`164x55`' задает окно шириной 164 столбцов, достаточной для двух расположенных рядом окон обычной ширины, и высотой 55 строк.

По умолчанию ширина равна 80 столбцам, а высота — 40 строкам. Вы можете не задавать ширину или высоту, или и то, и другое. Если описание геометрии начинается целым числом, Emacs интерпретирует его как ширину. Если вы начнете описание с '`x`', за которым идет число, Emacs воспримет это как высоту. Таким образом, '`81`' задает только ширину; '`x45`' задает только высоту.

Если вы написали в начале '`+`' или '`-`', это начинает описание сдвига, что означает, что оба размера опущены. Таким образом, '`-3`' задает только *сдвиг-по-x*. (Если вы задали лишь один сдвиг, это всегда *сдвиг-по-x*.) '`+3-3`' задает и *сдвиг-по-x*, и *сдвиг-по-y*, помещая фрейм возле нижнего левого края экрана.

Вы можете указать значения по умолчанию для любого из этих полей в файле '`.Xdefaults`', ключ '`-geometry`' тогда перекрывает эти установки.

## A.10 Внутренние и внешние рамки

Фрейм Emacs имеет внутреннюю и внешнюю рамки. Внутренняя рамка — это дополнительная полоска цвета фона по четырем сторонам фрейма. Внутреннюю рамку добавляет

сам Emacs. Внешняя рамка добавляется программой управления окнами за пределами внутренней рамки; она может содержать различные кнопки, на которые вы можете щелкнуть для перемещения или минимизирования этого окна.

‘`-ib` *ширина*’
‘`-internal-border=`*ширина*’
: Задает *ширину* внутренней рамки.

‘`-bw` *ширина*’
‘`-border-width=`*ширина*’
: Задает *ширину* внешней рамки.

Когда вы задаете размер фрейма, ширина рамок не учитывается. Позиция фрейма отсчитывается от внешней границы внешней рамки.

Для задания внутренней рамки шириной *n* пикселей используйте ключ ‘`-ib` *n*’. По умолчанию это 1. Для задания внешней рамки шириной *n* пикселей используйте ключ ‘`-bw` *n*’ (хотя программа управления окнами может и не учесть заданное вами число). По умолчанию ширина внешней рамки равна двум.

## A.11 Заголовки фреймов

Фрейм Emacs может иметь заголовок или не иметь его. Заголовок фрейма, если задан, появляется в оформлении окна и пиктограммах как имя этого фрейма. Если заголовок фрейма Emacs не задан, по умолчанию он будет составлен из имени исполняемой программы и имени вашей машины (если есть только один фрейм), или им будет имя буфера в выбранном окне (если есть несколько фреймов).

Вы можете указать заголовок начального фрейма Emacs с помощью ключа командной строки:

‘`-title` *заголовок*’
‘`-title=`*заголовок*’
‘`-T` *заголовок*’
: Задает *заголовок* начального фрейма Emacs.

Ключ ‘`-name`’ (см. Раздел A.13 [Ресурсы X], с. 396) также задает заголовок начального фрейма Emacs.

## A.12 Пиктограммы

Большинство программ управления окнами позволяют пользователю “минимизировать” фрейм, убирая его из виду и оставляя на его месте небольшое отличительное окно-“пиктограмму”. Если щелкнуть на окне-пиктограмме, снова появится сам фрейм. Если у вас одновременно запущено несколько клиентов, вы можете избежать загромождения экрана, минимизировав большинство из них.

‘`-i`’
‘`-icon-type`’
: Использовать в качестве пиктограммы Emacs портрет гну.

‘`-iconic`’
‘`-iconic`’ Запустить Emacs в минимизированном виде.

Ключ ‘`-i`’ или ‘`-icon-type`’ говорит Emacs использовать пиктограмму, содержащую рисунок с изображением гну GNU. Если этот ключ не задан, Emacs предоставляет выбор пиктограммы программе управления окнами — обычно это просто небольшой прямоугольник с заголовком фрейма.

Ключ ‘`-iconic`’ велит Emacs запускаться как пиктограмма, а не открывая фрейм обычным способом. В таком случае пиктограмма только показывает, что Emacs запустился; обычный текстовый фрейм не появляется, пока вы не деминимизируете его.

## A.13 X-ресурсы

Запущенные в системе X Windows программы организуют свои пользовательские параметры в иерархию классов и ресурсов. Вы можете задать для этих параметров значения по умолчанию в вашем файле X-ресурсов, обычно называемом ‘`~/.Xdefaults`’.

Каждая строка в этом файле задает значение одного параметра или набора связанных параметров для одной или нескольких программ (возможно, даже для всех программ).

Программы определяют именованные ресурсы с конкретным смыслом. Они также определяют, как ресурсы группируются в именованные классы. Например, ресурс ‘`internalBorder`’ в Emacs контролирует ширину внутренней рамки, а ресурс ‘`borderWidth`’ — ширину внешней рамки. Оба этих ресурса являются частью класса ‘`BorderWidth`’. В этих именах важен правильный регистр букв.

В файле ‘`~/.Xdefaults`’ вы можете задать значение для одного ресурса в каждой строке, следующим образом:

```
emacs.borderWidth: 2
```

Или вы можете использовать имя класса, чтобы задать то же значение для всех ресурсов этого класса. Вот пример:

```
emacs.BorderWidth: 2
```

Если вы задали значение для класса, оно становится значением по умолчанию для всех ресурсов этого класса. Вы также можете указать значения для отдельных ресурсов; для этих конкретных ресурсов они перекроют значение, общее для класса. Таким образом, следующий пример задает 2 как ширину всех рамок по умолчанию, но заменяет это значение на 4 для внешней рамки:

```
emacs.Borderwidth: 2
emacs.borderwidth: 4
```

Порядок, в котором эти строки появляются в этом файле, не играет роли. Также, ключи командной строки всегда обладают приоритетом перед файлом X-ресурсов.

Строка ‘`emacs`’ в примере выше также является именем ресурса. На самом деле, она представляет собой имя исполняемого файла, который вы использовали для запуска Emacs. Если Emacs установлен под другим именем, он ищет ресурсы с этим именем, а не с ‘`emacs`’.

‘`-name` *имя*’
‘`-name=`*имя*’
: Использовать *имя* в качестве имени ресурса (и заголовка) для начального фрейма Emacs. Этот ключ не влияет на остальные фреймы, но программы на Лиспе могут указать при создании фрейма его имя.

  Если вы не задали этот ключ, по умолчанию в качестве имени ресурса используется имя исполняемого файла Emacs.

‘`-xrm` *значение-ресурса*’
‘`-xrm=`*значение-ресурса*’
: Устанавливает значения X-ресурсов для данного задания Emacs (смотрите ниже).

Для согласованности, ‘`-name`’ также задает имя для использования со значениями других ресурсов, не принадлежащих какому-то определенному фрейму.

Ресурсы, именующие любой запущенный Emacs, также образуют класс; его имя — ‘`Emacs`’. Если вы напишете ‘`Emacs`’ вместо ‘`emacs`’, то такой ресурс будет относиться ко

всем фреймам во всех заданиях Emacs, вне зависимости от заголовков фреймов и имени исполняемого файла. Вот пример:

```
Emacs.BorderWidth: 2
Emacs.borderWidth: 4
```

Вы можете указать строку значений дополнительных ресурсов для Emacs с помощью ключа командной строки '`-xrm=`*ресурсы*'. Текст *ресурсы* должен иметь тот же формат, что вы используете в файле X-ресурсов. Чтобы задать спецификации нескольких ресурсов, разделите их переводом строки, как вы сделали бы в файле. Вы также можете использовать '`#include "`*имя-файла*`"`' для включения файла ресурсов. Значения ресурсов, заданные через '`-xrm`', имеет приоритет перед всеми остальными спецификациями.

Следующая таблица перечисляет имена ресурсов, обозначающих параметры для Emacs, каждый со своим классом:

`background` (class `Background`)
: Название цвета фона.

`bitmapIcon` (class `BitmapIcon`)
: Если это '`on`', использовать растровую пиктограмму (изображение гну), если это '`off`', предоставить право выбора пиктограммы программе управления окнами.

`borderColor` (class `BorderColor`)
: Название цвета внешней рамки.

`borderWidth` (class `BorderWidth`)
: Ширина внешней рамки в пикселях.

`cursorColor` (class `Foreground`)
: Название цвета для текстового курсора (точки).

`font` (class `Font`)
: Имя шрифта для текста (или имя набора шрифтов, см. Раздел 18.10 [Наборы шрифтов], с. 170).

`foreground` (class `Foreground`)
: Название цвета для текста.

`geometry` (class `Geometry`)
: Размер и положение окна. Будьте осторожны, не определяйте этот ресурс как '`emacs*geometry`', потому что это может повлиять на отдельные меню, а не только на сам фрейм Emacs.

  Если этот ресурс задает положение, то оно относится только начальному фрейму Emacs (или, в случае ресурса для определенного фрейма, только для этого фрейма). Однако, если здесь задан размер, то он относится ко всем фреймам.

`iconName` (class `Title`)
: Заголовок для отображения на пиктограмме.

`internalBorder` (class `BorderWidth`)
: Ширина внутренней рамки в пикселях.

`menuBar` (class `MenuBar`)
: Если '`on`', показывать во фреймах полоски меню; если '`off`', не показывать.

`minibuffer` (class `Minibuffer`)
: Если '`none`', не создавать в этом фрейме минибуфер. Вместо этого будет использоваться отдельный фрейм минибуфера.

`paneFont` (class `Font`)
: Имя шрифта для заголовков поля меню, в версиях Emacs, не использующих библиотеку виджетов.

`pointerColor` (class `Foreground`)
: Цвет указателя мыши.

`reverseVideo` (class `ReverseVideo`)
: Если ‘`on`’, обменять цвета фона и текст, если ‘`off`’, использовать цвета, как они заданы.

`verticalScrollBars` (class `ScrollBars`)
: Показывать во фреймах полоски прокрутки, если это ‘`on`’; показывать, если это ‘`off`’.

`selectionFont` (class `Font`)
: Имя шрифта для пунктов всплывающих меню в версиях Emacs, не использующих библиотеку виджетов. (Для версий, использующих библиотеку виджетов, смотрите Раздел A.14 [X-ресурсы для Lucid], с. 398, а также Раздел A.15 [X-ресурсы для Motif], с. 399.)

`title` (class `Title`)
: Название для отображения в заголовке начального фрейма Emacs.

Это ресурсы для изменения внешнего вида определенных начертаний (см. Раздел 17.13 [Начертания], с. 155):

*начертание*`.attributeFont`
: Шрифт для заданного *начертания*.

*начертание*`.attributeForeground`
: Цвет текста для заданного *начертания*.

*начертание*`.attributeBackground`
: Цвет фона для заданного *начертания*.

*начертание*`.attributeUnderline`
: Флаг подчеркивания для заданного *начертания*. Если ‘`on`’ или ‘`true`’, то подчеркивать.

## A.14 X-ресурсы для меню Lucid

Если установленный на вашей системе Emacs был собран с использованием библиотеки с виджетами меню Lucid, то полоска меню — это отдельный виджет, и у него свои ресурсы. Имена этих ресурсов содержат строку ‘`pane.menubar`’ (как всегда, после строки вызова Emacs или ‘`Emacs`’, что обозначает любой вызов). Задавайте их так:

```
Emacs.pane.menubar.ресурс:   значение
```

Например, чтобы задать для пунктов меню шрифт ‘`8x16`’, напишите так:

```
Emacs.pane.menubar.font: 8x16
```

Ресурсы для всплывающих меню, а не полоски меню, начинаются с ‘`menu*`’ подобным же образом. Например, чтобы задать для пунктов всплывающих меню шрифт ‘`8x16`’, напишите следующее:

```
Emacs.menu*.font:  8x16
```

Для диалогов вместо ‘`menu`’ пишите ‘`dialog`’:

```
Emacs.dialog*.font:  8x16
```

Опыт показывает, что на некоторых системах вам может понадобиться добавить перед ‘`pane.menubar`’ или ‘`menu*`’ строку ‘`shell.`’. На некоторых других системах вы должны не добавлять ‘`shell.`’.

Вот перечень ресурсов для полоски меню и всплывающих меню:

`font` Шрифт текста пунктов меню.

`foreground`
Цвет текста.

`background`
Цвет фона.

`buttonForeground`
В полоске меню, цвет фона для выделенного пункта.

`horizontalSpacing`
Горизонтальный промежуток между пунктами. По умолчанию это 3.

`verticalSpacing`
Вертикальный промежуток между пунктами. По умолчанию это 1.

`arrowSpacing`
Горизонтальный промежуток между стрелкой (обозначающей подменю) и связанным текстом. По умолчанию это 10.

`shadowThickness`
Толщина линии затенения вокруг виджета.

## A.15 X-ресурсы для меню Motif

Если установленный на вашей системе Emacs был собран с использованием библиотеки виджетов Motif, то полоска меню — это отдельный виджет, и у него свои ресурсы. Имена этих ресурсов содержат строку '`pane.menubar`' (как всегда, после строки вызова Emacs или '`Emacs`', что обозначает любой вызов). Задавайте их так:

`Emacs.pane.menubar.`*подвиджет*`.`*ресурс*`:` *значение*

Каждая отдельная строка в полоске меню является подвиджетом; имя подвиджета такое же, что и текст пункта. Например, слово '`Files`' в полоске меню — это часть подвиджета с именем '`emacs.pane.menubar.Files`'. Вероятнее всего вы захотите задать одни и те же ресурсы для всей полоски меню. Чтобы сделать это, используйте '`*`' вместо имени конкретного подвиджета. Например, чтобы задать для пунктов меню шрифт '`8x16`', напишите следующее:

```
Emacs.pane.menubar.*.fontList:  8x16
```

Это также задает значение этого ресурса для подменю.

Каждый пункт в подменю полоски меню также имеет собственное имя для X-ресурсов; например, подменю '`Files`' имеет пункт, называемый '`Save Buffer`'. Описание ресурса для пункта подменю выглядит так:

`Emacs.pane.menubar.popup_*.`*меню*`.`*пункт*`.`*ресурс*`:` *значение*

Например, задать шрифт для пункта '`Save Buffer`' можно следующим образом:

```
Emacs.pane.menubar.popup_*.Files.Save Buffer.fontList: 8x16
```

Для пункта в подменю второго уровня, таком как '`Check Message`' в подменю '`Spell`' в меню '`Edit`', ресурс соответствует такому шаблону:

`Emacs.pane.menubar.popup_*.popup_*.`*меню*`.`*ресурс*`:` *значение*

Например,

`Emacs.pane.menubar.popup_*.popup_*.Spell.Check Message:` *значение*

Невозможно определить ресурс для всех пунктов полоски меню, не определяя их также и для подменю. Поэтому, если вы хотите, чтобы пункты подменю выглядели не так, как сама полоска меню, вы должны попросить об этом в два этапа. Сначала задайте ресурс для всех их; затем переопределите это значение только для подменю. Вот пример:

```
Emacs.pane.menubar.*.fontList:  8x16
Emacs.pane.menubar.popup_*.fontList: 8x16
```

Для всплывающих меню используйте ‘`menu*`’, а не ‘`pane.menubar`’. Например, чтобы задать для пунктов всплывающих меню шрифт ‘`8x16`’, напишите вот это:

```
Emacs.menu*.fontList:  8x16
```

Вот перечень ресурсов для полосок меню и всплывающих меню:

`armColor` Цвет для отображения кнопки со стрелкой.

`fontList` Используемый шрифт.

`marginBottom`
`marginHeight`
`marginLeft`
`marginRight`
`marginTop`
`marginWidth`
: Количество пространства, которое следует оставить вокруг пункта внутри рамки.

`borderWidth`
: Ширина рамки вокруг пункта меню с каждой стороны.

`shadowThickness`
: Ширина рамки затенения.

`bottomShadowColor`
: Цвет рамки затенения снизу и справа.

`topShadowColor`
: Цвет рамки затенения слева и сверху.

# Приложение B Антиновости для Emacs 19

Для тех пользователей, кто живет назад во времени, здесь представлены сведения об устарении Emacs до версии 19. Мы надеемся, что вам понравятся большие упрощения, появляющиеся из-за отсутствия определенных особенностей Emacs 20.

- Поддержка многобайтных знаков и преобразование последовательности конец-строки полностью убраны. (Некоторые пользователи считают это грандиозным продвижением.) Коды знаков ограничены промежутком от 0 до 255, а файлы, импортированные на Unix-подобные системы, могут содержать в конце каждой строки знак ^M, чтобы вы не забывали о контроле над файлами типа MS-DOG.
- Наборы шрифтов, системы кодирования и методы ввода также уничтожены.
- Строка режима по умолчанию показывает строку '`Emacs`', на случай, если вы забыли, каким редактором пользуетесь.
- Полосы прокрутки всегда находятся с правой стороны окна. Это ясно отделяет их от текста в окне.
- Функция `M-x customize` заменена очень простой функцией `M-x edit-options`. Она показывает вам *все* пользовательские переменные с самого начала, так что вам не нужно отыскивать нужные. Кроме того, для установки пользовательской переменной она предоставляет совсем немного команд, таких как `s` и `x`.
- Клавиша ⟨DELETE⟩ не делает в Emacs 19 ничего особенного, когда вы используете ее после выделения области с помощью мыши. В такой ситуации она делает то же, что и всегда: удаляет один предшествующий знак.
- `C-x C-w` теперь не изменяет основной режим в соответствии с новым именем файла. Если вы хотите изменить основной режим, воспользуйтесь `M-x normal-mode`.
- В режиме Transient Mark область подсвечивается в каждом окне.
- Режим Outline не использует свойства перекрытия; вместо этого, он прячет строку, преобразуя предшествующий перевод в строки в код 015. Однако, когда вы сохраняете файл, знак 015 волшебным образом появляется в нем как перевод строки.
- Теперь есть хитрый способ, которым вы можете активизировать минибуферы рекурсивно, даже если `enable-recursive-minibuffers` равна `nil`. Все, что вам нужно сделать, — *переключить окно* на другое, не являющееся окном минибуфера, а затем использовать команду минибуфера. Таким методом вы можете произвести любое число уровней минибуферов, но `M-x top-level` выведет вас из всех.
- Мы убрали ограничение на длину списков истории минибуферов; теперь они содержат все аргументы минибуфера, использовавшиеся с начала сеанса.
- Динамическое раскрытие сокращений теперь преобразует регистр очень простым и прямолинейным способом. Если вы запросили сохранение регистра, все раскрытие происходит по образцу введенного вами сокращения.
- Команды `compose-mail` больше не существует; `C-x m` теперь непосредственно запускает `mail`.
- Нельзя отменить специальное значение некоторых знаков в именах файлов. Что вы видите, то и получаете: если имя выглядит как имя удаленного файла, то это имя удаленного файла.
- `M-x grep-find` удалена, потому что `grep` никогда и не терялся.
- Некоторые команды Dired переорганизованы: двухзнаковые последовательности заменены на быстрые однознаковые команды:
  - Для `dired-mark-executables`, напечатайте `*`.
  - Для `dired-mark-directories`, напечатайте `/`.
  - Для `dired-mark-symlinks`, напечатайте `@`.

- Для `dired-change-marks`, напечатайте `c`.
- Для `dired-unmark-all-files`, напечатайте `C-M-?`.
- Для `dired-unmark-all-marks`, напечатайте `C-M-?` ⟨RET⟩.

Но если вы хотите использовать команду `dired-flag-garbage-files`, `&`, вам нужно перестать жить в прошлом.

- В режиме C вы теперь можете задать предпочтительный стиль блоков комментариев. Например, если вы хотите использовать стиль

```
/*
blah
blah
*/
```

вы должны установить переменную `c-block-comments-indent-p` в значение `t`.

- Для настройки начертаний режима Font Lock используйте переменную `font-lock-face-attributes`. Для получения подробностей смотрите ее документацию.
- Для эффективности режим Font Lock теперь использует по умолчанию наименьший поддерживаемый уровень оформления для выбранного основного режима.
- Если вы уничтожаете буфер, все регистры, содержащие сохраненные позиции в этом буфере, будут указывать неизвестно куда.
- Функция `set-frame-font` переименована в `set-default-font`.
- Переменной `tex-main-file` не существует. Конечно, вы можете создать эту переменную, но это не даст ничего особенного.
- Переменная `scroll-preserve-screen-position` удалена; также исчезла и возможность, которой она управляла.
- Мы удалили функции `add-untranslated-filesystem` и `remove-untranslated-filesystem` и заменили их более простой, `using-unix-filesystems`.
- Чтобы следовать продолжающемуся уменьшению объема памяти компьютеров, из Emacs 19 исключены многие другие файлы и функции. Нет необходимости упоминать здесь их все. Если вы попытаетесь применить одну из них, вы получите сообщение об ошибке, говорящее, что она не определена или не имеет значения.

# Приложение C Emacs и MS-DOS

Этот раздел кратко описывает особенности использования Emacs под "операционной системой" MS-DOS (известной также как "MS-DOG"). Если вы соберете Emacs для MS-DOS, этот двоичный файл можно будет также запустить на Windows 3.X, Windows NT, Windows 9X или OS/2 как приложение DOS; сведения из данной главы относятся ко всем этим системам, если вы используете Emacs, который был скомпилирован для MS-DOS.

Заметьте, что возможно собрать Emacs специально для Windows NT или Windows 9X. Если вы сделаете так, большая часть главы будет неприменима к вашему случаю; тогда вы получите поведение, которое более близко к тому, что описано в остальной части руководства, и которое включает поддержку для длинных имен файлов, множественных фреймов, полосок прокрутки, меню для мыши и подпроцессов. Однако, раздел о текстовых и двоичных файлах все еще будет иметь силу. Также в конце этой главы есть два раздела, которые относятся исключительно к Windows NT и 9X.

## C.1 Клавиатура и мышь в MS-DOS

Раскладки клавиатуры PC используют в качестве клавиши ⟨META⟩ левую клавишу ⟨ALT⟩. Вы можете выбрать между двумя способами эмулирования клавиш ⟨SUPER⟩ и ⟨HYPER⟩: использовать либо правый ⟨CTRL⟩, либо правый ⟨ALT⟩, установив переменные `dos-hyper-key` и `dos-super-key`, соответственно, в значение 1 или 2. Если ни `dos-super-key`, ни `dos-hyper-key` не равна 1, то по умолчанию правый ⟨ALT⟩ также отображается в клавишу ⟨META⟩. Однако, если установлена `KEYB.COM`, программа для поддержки в MS-DOS многоязыковой клавиатуры, Emacs *не будет* отображать правый ⟨ALT⟩ в ⟨META⟩, так как он используется для получения таких знаков, как `~` и `@` в раскладках, отличных от раскладки US; в таком случае вы можете использовать в качестве клавиши ⟨META⟩ только левый ⟨ALT⟩.

Переменная `dos-keypad-mode` — это флаг, который управляет кодами, возвращаемыми клавишами дополнительной цифровой клавиатуры. Вы также можете определить клавишу ⟨ENTER⟩ с дополнительной клавиатуры как `C-j`, поместив в ваш файл '`_emacs`' следующую строку:

```
;; Делает так, чтобы Enter с дополнительной
;; клавиатуры действовал как C-j.
(define-key function-key-map [kp-enter] [?\C-j])
```

Клавиша, известная в Emacs как ⟨DEL⟩ (потому что таково ее назначение на большинстве рабочих станций), на PC известна как ⟨BS⟩ (забой). Поэтому специфичная для PC инициализация терминала изменяет отображение клавиши ⟨BS⟩ таким образом, чтобы она действовала как ⟨DEL⟩; клавиша ⟨DEL⟩ отображается в `C-d` по тем же причинам.

Emacs, собранный для MS-DOS, распознает в качестве ключа выхода комбинацию `C-`⟨BREAK⟩, точно так же, как `C-g`. Это делается, потому что Emacs не может зарегистрировать нажатие `C-g`, если только он не готов к принятию ввода. Как следствие, вы не можете использовать `C-g` для остановки запущенной команды (см. Раздел 32.1 [Выход], с. 371). Напротив, `C-`⟨BREAK⟩ *можно* уловить, как только вы ее набрали (как `C-g` на других системах), поэтому ее можно применять для остановки работающей команды и для аварийного выхода (см. Раздел 32.2.8 [Аварийный выход], с. 374).

Emacs поддерживает в MS-DOS мышь (только на терминале по умолчанию). Команды мыши работают как описано, включая те, что используют меню и полоску меню (см. Раздел 1.4 [Полоска меню], с. 26). Полоски прокрутки не работают в MS-DOS Emacs. Мыши на PC обычно имеют только две кнопки; они действуют как `Mouse-1` и `Mouse-2`, но если вы нажмете их вместе, то они подействуют как `Mouse-3`.

Собранный для MS-DOS и запущенный под Windows Emacs поддерживает операции с буфером обмена. Команды, которые помещают текст в список уничтожений или восстанавливают текст из него, сначала проверяют буфер обмена, так же, как в X Windows (см.

Раздел 17.1 [Команды мыши], с. 147). В MS-DOS Emacs под Windows поддерживаются только первичное выделение и буфер вырезок; вторичное выделение всегда выглядит как пустое.

Доступ к буферу обмена реализован в Windows так, что длина текста, который вы можете в него поместить, ограничена количеством доступной Emacs свободной памяти DOS. Обычно в буфере может уместиться до 620KB текста, но этот предел зависит от конфигурации системы и будет меньше, если вы запустите Emacs как подпроцесс другой программы. Если уничтоженный текст не умещается, Emacs печатает сообщение об этом и не помещает этот текст в буфер обмена.

Также, в буфер обмена Windows нельзя поместить знаки с нулевым кодом. Если уничтожаемый текст содержит нули, Emacs не будет записывать его в буфер обмена и напечатает об этом предупреждение в эхо-области.

Переменная `dos-display-scancodes`, если она не равна `nil`, велит Emacs показывать ASCII-значение скан-кодов клавиатуры при каждом нажатии клавиши; эта возможность служит дополнением к команде `view-lossage`, она полезна для отладки.

## C.2 Изображение в MS-DOS

В MS-DOS нельзя использовать вариации шрифтов, такие как курсив или жирный шрифт, но множественные начертания поддерживаются; каждое из них может определять цвета шрифта и фона. Следовательно, вы можете пользоваться всей функциональностью пакетов Emacs, которые используют шрифты (таких как `font-lock`, режим Enriched Text и других), определяя подходящие начертания как использующие различные цвета. Чтобы посмотреть, какие цвета и начертания доступны, и как они выглядят, запустите команду `list-colors-display` (см. Раздел 17.10 [Параметры фрейма], с. 153) и команду `list-faces-display` (см. Раздел 17.13 [Начертания], с. 155).

См. Раздел C.6 [MS-DOS и MULE], с. 408, где описано, как Emacs отображает глифы и знаки, которые не поддерживаются встроенными шрифтами.

Множественные фреймы (см. Глава 17 [Фреймы], с. 147) поддерживаются в MS-DOS, но они перекрываются, так что в одно время вы можете видеть только один из них. Этот единственный фрейм занимает весь экран. Когда вы запускаете Emacs в сеансе DOS из-под MS-Windows, вы можете сделать так, чтобы видимый фрейм занимал не весь экран, но Emacs все равно не может показать более одного фрейма одновременно.

Команда `mode4350` переключает между отображением 43 или 50 строк, в зависимости от вашей аппаратуры; команда `mode25` переключает к размеру экрана по умолчанию, 80x25.

По умолчанию Emacs знает только как установить размер окна для 80-ти столбцов на 25, 28, 35, 40, 43 или 50 строк. Однако, если у вашего видеоадаптера есть особые видеорежимы, которые переключаю изображение в другие размеры, вы можете сделать так, чтобы Emacs поддерживал и их тоже. Когда вы просите Emacs изменить размер фрейма на $n$ столбцов и $m$ строк, он проверяет, есть ли переменная с именем `screen-dimensions-`*nxm*, и если она есть, использует ее значение (которое должно быть целым числом) как видеорежим, в который нужно переключиться. (Emacs переключается в этот видеорежим, вызывая функцию BIOS `Set Video Mode` со значением `screen-dimensions-`*nxm* в регистре `AL`.) Например, представим, что ваш адаптер переходит к размерам 66x80, когда его перевели в видеорежим 85. Тогда вы можете сделать так, чтобы Emacs поддерживал этот размер экрана, поместив в ваш файл '`_emacs`' следующее:

```
(setq screen-dimensions-66x80 85)
```

Поскольку Emacs способен в MS-DOS установить размер фрейма только к конкретным поддерживаемым размерам, он не может следовать любому возможному запросу на его изменение. Когда запрошен неподдерживаемый размер, Emacs выбирает следующий,

больший, чем указанный. Например, если вы потребуете фрейм 36x80, вместо этого вы получите 40x80.

Переменные `screen-dimensions-`*nxm* используются, только если они точно совпадают с указанным размером; при поиске следующего большего размера они игнорируются. В примере выше, даже если ваш VGA-адаптер поддерживает размер 38x80, и вы определили переменную `screen-dimensions-38x80` с соответствующим значением, вы тем не менее получите экран 40x80, когда потребуете сделать размер фрейма равным 36x80. Если вы хотите получить в этом случае размеры 38x80, вы можете сделать это, установив переменную с именем `screen-dimensions-36x80` и тем же значением видеорежима, что и `screen-dimensions-38x80`.

Изменение размеров фрейма в MS-DOS приводит также к новым размерам и все остальные фреймы.

## C.3 Имена файлов в MS-DOS

MS-DOS обычно использует для разделения единиц имени файла обратную косую черту, '\', а не обычную косую черту, как другие системы. Emacs под MS-DOS позволяет использовать и простую, и обратную косую черту, а также знает о буквах дисков в именах файлов.

Имена файлов в MS-DOS регистронезависимы и ограничены восемью знаками плюс необязательная точка и еще три знака. Emacs достаточно осведомлен об этих ограничениях, чтобы справляться с именами файлов, предназначенных для других операционных систем. К примеру, начальные точки '.' в имени файла недопустимы в MS-DOS, поэтому Emacs автоматически переводит их в подчерки '_'; так что ваш файл инициализации по умолчанию (см. Раздел 31.7 [Файл инициализации], с. 366) называется в MS-DOS '`_emacs`'. Излишние знаки перед или после точки обычно игнорируются самой MS-DOS; поэтому, если вы обращаетесь к файлу '`LongFileName.EvenLongerExtension`', вы без предупреждений получите '`longfile.eve`', но в строке режима Emacs все так же будет показывать это длинное имя файла. Во всех остальных отношениях вы сами должны заботиться об указании допустимых под MS-DOS имен файлов; описанное выше прозрачное преобразование действует только на имена файлов внутри Emacs.

Упомянутые выше ограничения на имена файлов в MS-DOS делают конструирование имен резервных файлов почти невозможным (см. Раздел 14.3.1.1 [Имена резервных файлов], с. 110) без потери некоторых знаков из первоначального имени файла. Например, именем резервного файла для '`docs.txt`' будет '`docs.tx~`', даже если использован единственный резервный файл.

Если вы запускаете Emacs как приложение DOS под Windows 9X, вы можете включить поддержку длинных имен файлов. Если вы сделаете так, Emacs не станет усекать имена файлов или приводить их к нижнему регистру; напротив, он будет использовать имена файлов так, как вы их задали, без изменений. Чтобы задействовать поддержку длинных имен файлов, установите переменную среды `LFN` равной '`y`' перед запуском Emacs. К сожалению, Windows NT не позволяет программам DOS получать доступ к длинным именам файлов, так что собранный для MS-DOS Emacs будет видеть только их короткие псевдонимы в форме 8+3.

В MS-DOS нет понятия о начальном каталоге, поэтому в MS-DOS Emacs делает вид, что каталог, в котором он установлен — это значение переменной среды `HOME`. То есть, если исполняемый файл Emacs, '`emacs.exe`', находится в каталоге '`c:/utils/emacs/bin`', то Emacs ведет себя так, как если бы `HOME` была установлена равной '`c:/utils/emacs`'. В частности, именно здесь Emacs ищет файл инициализации, '`_emacs`'. Помня это, вы можете использовать в именах файлов знак '~' как псевдоним начального каталога, как вы делали бы это в Unix. Вы также можете установить переменную среды `HOME` до запуска Emacs; ее значение перекроет описанное выше поведение по умолчанию.

Emacs обрабатывает в MS-DOS имя каталога '`/dev`' особым образом, из-за некоторых средств в библиотеках эмулятора из DJGPP, которые создают видимость, что в этом каталоге перечислены имена устройств ввода/вывода. Мы рекомендуем избегать использования на любом диске настоящего каталога с именем '`/dev`'.

## C.4 Текстовые файлы и двоичные файлы

GNU Emacs использует для разделения строк знак перевода строки. Это соглашение относится к Unix, на которой GNU Emacs был разработан, и в системах GNU, так как они принимают Unix за модель.

MS-DOS и MS-Windows обычно используют для разделения строк последовательность возврат каретки-прогон строки. (Прогон строки — это тот же знак, что и перевод строки.) Следовательно, для удобного редактирования в Emacs необходимо преобразование этих последовательностей конец-строки (EOL). И обычно Emacs делает это: во время считывания файла он превращает возврат каретки-перевод строки в перевод строки, а во время записи — перевод строки в возврат каретки-перевод строки. Это преобразование реализуется тем же механизмом, что управляет преобразованием кодов знаков из разных языков (см. Раздел 18.7 [Системы кодирования], с. 165).

Одним из следствий этого особого преобразования формата большинства файлов является то, что сообщаемая Emacs позиция знака (см. Раздел 4.9 [Информация о позиции], с. 40) не соответствует информации о размере файла, известной операционной системе.

Некоторые виды файлов не должны преобразовываться, так как их содержимое не является на самом деле текстом. Следовательно, Emacs различает в MS-DOS некоторые файлы как *двоичные* и считывает и записывает их буквально. (Это различение — не часть MS-DOS; оно делается только в Emacs.) Сюда включаются исполняемые файлы, сжатые архивы и так далее. Emacs решает, двоичный это файл или нет, по его имени: шаблоны имен, указывающих на двоичные файлы, определяются переменной `file-name-buffer-file-type-alist`. Обратите внимание, если имя файла соответствует одному из шаблонов для двоичных файлов в `file-name-buffer-file-type-alist`, Emacs использует систему кодирования `no-conversion` (см. Раздел 18.7 [Системы кодирования], с. 165), которая выключает *все* преобразования кодирования, а не только EOL.

Кроме того, если Emacs узнает из содержимого файла, что в нем в качестве разделителя строк используется перевод строки, а не возврат каретки-прогон строки, он не делает превращения при чтении или записи этого файла. Таким образом, вы можете редактировать в MS-DOS файлы с систем Unix или GNU без специальных усилий, и в них останутся знаки EOL в стиле Unix.

Вы можете обратиться к файлу и указать, нужно ли воспринимать его как текстовый или же как двоичный, с помощью команд `find-file-text` и `find-file-binary`. Преобразование EOL — это часть более общего механизма преобразования системы кодирования, поэтому другой способ указать это же — воспользоваться командами для задания системы кодирования (см. Раздел 18.9 [Задание кодирования], с. 168). Например, `C-x` ⟨RET⟩ `c undecided-unix` ⟨RET⟩ `C-x C-f foobar.txt` обратится к файлу '`foobar.txt`' без превращения EOL.

В строке режима показывается, использовалось ли для текущего буфера преобразование EOL. Обычно после буквы системы кодирования в начале строки режима появляется двоеточие. Если в этом буфере используется преобразование последовательности конец-строки MS-DOS, этот знак заменяется на обратную косую черту.

Если вы используете NFS или Samba для доступа к файловым системам, находящимся на компьютерах с Unix или GNU, Emacs не должен производить преобразование EOL для любых файлов в этих файловых системах — даже если вы создаете новый файл. Чтобы затребовать это, обозначьте их как *нетранслируемые* файловые системы, вызвав функцию

`add-untranslated-filesystem`. Она принимает один аргумент: имя файловой системы, включая букву диска, и, возможно, каталог. Например,

```
(add-untranslated-filesystem "Z:")
```

обозначает как нетранслируемую файловую систему диск Z, а

```
(add-untranslated-filesystem "Z:\\foo")
```

обозначает как нетранслируемую файловую систему каталог '`\foo`' на диске Z.

Чаще всего `add-untranslated-filesystem` пишется в файле '`_emacs`' или же в файле '`site-start.el`', чтобы все пользователи вашей системы получили от нее пользу.

Чтобы отменить действие `add-untranslated-filesystem`, используйте функцию `remove-untranslated-filesystem`. Эта функция принимает один аргумент, который должен быть такой же строкой, что была ранее передана `add-untranslated-filesystem`.

## C.5 Печать в MS-DOS

Команды печати, такие как `lpr-buffer` (см. Раздел 30.4 [Распечатка], с. 331) и `ps-print-buffer` (см. Раздел 30.5 [Postscript], с. 332), могут работать в MS-DOS и MS-Windows, посылая вывод на порты принтера, если недоступна команда в стиле Unix `lpr`. Такое поведение контролируется теми же переменными, что управляют печатью через `lpr` в Unix (см. Раздел 30.4 [Распечатка], с. 331, см. Раздел 30.6 [Управление печатью в Postscript], с. 332), но MS-DOS и MS-Windows значения этих переменных по умолчанию отличаются от их значений по умолчанию в Unix.

Если вы хотите использовать ваш локальный принтер, печатающий обычным для DOS образом, то установите лисповскую переменную `lpr-command` в значение `""` (это значение по умолчанию), а `printer-name` — равной имени порта принтера — например, `"PRN"`, обычного порта локального принтера (это значение по умолчанию), или `"LPT2"`, или `"COM1"`, для последовательного принтера. Вы также можете установить `printer-name` равной имени файла, в этом случае "печатаемый" вывод на самом деле добавляется в конец этого файла. Если вы установите `printer-name` в значение `"NUL"`, печатаемый вывод забывается (посылается в системное нулевое устройство) без предупреждений.

В MS-Windows, когда установлено сетевое программное обеспечение, вы можете также использовать принтер, предоставляемый другой машиной, установив `printer-name` равной UNC-имени разделяемого ресурса для этого принтера — например, `"//joes_pc/hp4si"`. (Здесь не имеет значения, пишете ли вы простые или обратные косые черты.) Чтобы выяснить имена разделяемых принтеров, запустите в командной подсказке DOS команду '`net view`' для получения перечня серверов и '`net view` *имя-сервера*', чтобы увидеть имена принтеров (и каталогов), предоставляемых этим сервером.

Если вы устанавливаете `printer-name` равной имени файла, лучше всего будет задать абсолютное имя. Emacs изменяет рабочий каталог в соответствии с каталогом по умолчанию текущего буфера, так что если имя файла в `printer-name` относительно, в результате у вас окажется несколько таких файлов, по одному в каталоге каждого буфера, откуда производилась печать.

Команды `print-buffer` и `print-region` вызывают программу `pr` или используют особые переключатели программы `lpr`, чтобы создать на каждой напечатанной странице заголовок. Обычно в MS-DOS и MS-Windows нет этих программ, поэтому по умолчанию переменная `lpr-headers-switches` устанавливается так, что запросы на печать заголовков страниц игнорируются. Поэтому `print-buffer` и `print-region` дают тот же вывод, что и `lpr-buffer` и `lpr-region`, соответственно. Если у вас есть подходящая программа `pr` (например, из GNU Textutils), установите `lpr-headers-switches` равной `nil`; тогда Emacs будет вызывать `pr` для создания заголовков страниц и печатать полученный вывод, как указано переменной `printer-name`.

Наконец, если у вас есть подобие `lpr`, вы можете установить переменную `lpr-command` в значение `"lpr"`. Тогда Emacs будет использовать для печати `lpr`, как и в других системах. (Если имя этой программы отличается от ‘`lpr`’, установите `lpr-command` так, чтобы она указывала, где ее можно найти.) Когда `lpr-command` не равна `""`, переменная `lpr-switches` несет свой обычный смысл. Если значением переменной `printer-name` является строка, то она используется как значение ключа ‘`-P`’ для `lpr`, как и в Unix.

Параллельный набор переменных, `ps-lpr-command`, `ps-lpr-switches` и `ps-printer-name` (см. Раздел 30.6 [Управление печатью в Postscript], с. 332), определяет, как нужно печатать PostScript-файлы. Эти переменные используются таким же способом, как соответствующие описанные выше переменные для печати не в PostScript. Таким образом, значение `ps-printer-name` используется в качестве имени устройства (или файла), на которое посылается вывод PostScript, так же, как `printer-name` используется для печати не в PostScript. (Это два различных набора переменных в случае, если у вас два принтера, подсоединенных к разным портам, и только один из них — PostScript-принтер.)

Значение переменной `ps-lpr-command` равно по умолчанию `""`, что направляет вывод PostScript на порт принтера, заданный в `ps-printer-name`. Но `ps-lpr-command` можно также установить равной имени программы, воспринимающей PostScript-файлы. Это значит, что если у вас не PostScript-принтер, вы можете задать в этой переменной имя программы-интерпретатора PostScript (такой как Ghostscript). Любые переключатели, которые необходимо передать интерпретатору, указываются в переменной `ps-lpr-switches`. (Если значением `ps-printer-name` является строка, она будет добавлена к списку переключателей в качестве значения ключа ‘`-P`’. Скорее всего, это будет полезно, только если вы пользуетесь `lpr`, поэтому, если вы печатаете через интерпретатор, вам следует сделать `ps-printer-name` не строкой, чтобы она проигнорировалась.)

Например, чтобы использовать Ghostscript для печати на принтере Epson, подсоединенному к порту ‘`LPT2`’, поместите в ваш файл ‘`_emacs`’ такой код:

```
(setq ps-printer-name t)  ; Ghostscript не принимает -P
(setq ps-lpr-command "c:/gs/gs386")
(setq ps-lpr-switches '("-q" "-dNOPAUSE"
"-sDEVICE=epson"
"-r240x72"
"-sOutputFile=LPT2"
"-Ic:/gs"))
```

(Это предполагает, что Ghostscript установлен в каталоге ‘`"c:/gs"`’.)

Для обратной совместимости, если переменная `dos-printer` (`dos-ps-printer`) имеет значение, то оно перекрывает значение `printer-name` (`ps-printer-name`); это относится только к MS-DOS и MS-Windows.

## C.6 Поддержка разных языков в MS-DOS

Emacs поддерживает в MS-DOS те же наборы знаков разных языков, что и в Unix, и на других платформах (см. Глава 18 [MULE], с. 161), включая системы кодирования для преобразования между различными наборами знаков. Однако, из-за несовместимости между MS-DOS/MS-Windows и Unix у этой поддержки есть несколько специфичных для DOS аспектов, о которых пользователи должны знать. Эти аспекты описаны в данном разделе.

`M-x dos-codepage-setup`
: Приводит экран Emacs и системы кодирования в соответствие с текущей кодовой страницей DOS.

`M-x codepage-setup`
: Создает систему кодирования для конкретной кодовой страницы DOS.

По своему дизайну MS-DOS поддерживает один набор из 256 знаков в каждый момент времени, но дает вам выбрать из множества наборов знаков. Эти альтернативные наборы знаков известны как *кодовые страницы DOS*. Каждая кодовая страница включает все 128 знаков ASCII, но остальные 128 (с кодами от 128 до 255) изменяются от одной кодовой страницы к другой. Каждой кодовой странице DOS присвоен трехзначный номер, как 850, 862, etc.

В противоположность X Windows, которая позволяет вам использовать несколько шрифтов одновременно, в MS-DOS нельзя использовать несколько кодовых страниц в течение одного сеанса. Вместо этого, MS-DOS загружает одну кодовую страницу во время запуска, и чтобы изменить кодовую страницу, вы должны перезагрузить MS-DOS[1]. Практически те же ограничения применяются к исполняемым файлам DOS в других системах, таких как MS-Windows.

Если вы вызываете Emacs в MS-DOS с ключом '`-unibyte`' (см. Раздел A.2 [Ключи запуска], с. 386), Emacs не делает никаких преобразований знаков, не входящих в ASCII. Напротив, он считывает и записывает такие знаки буквально и буквально посылает их восьмибитные коды на дисплей. Таким образом, однобайтный Emacs поддерживает в MS-DOS текущую кодовую страницу, какой бы она не была, но не может представить никакие другие знаки.

Для многобайтной работы на MS-DOS, Emacs должен знать, какие знаки может отображать выбранная кодовая страница DOS. Поэтому он вскоре после старта посылает системе запрос о номере выбранной кодовой страницы и сохраняет этот номер в переменной `dos-codepage`. Некоторые системы возвращают 437, значение по умолчанию, даже если в действительности кодовая страница другая. (Обычно это происходит, если вы используете кодовую страницу, встроенную в дисплей.) Вы можете велеть Emacs использовать другую кодовую страницу, устанавливая в вашем файле инициализации переменную `dos-codepage`.

Многобайтный Emacs поддерживает только некоторые кодовые страницы DOS, те, что показывают восточные виды письма, как японская кодовая страница 932, и те, которые кодируют один из наборов знаков ISO 8859.

Дальневосточные кодовые страницы могут непосредственно отображать один из наборов знаков MULE для этих стран, поэтому Emacs просто подготавливает к применению подходящую систему кодирования терминала, которая поддерживается этой кодовой страницей. Описываемые в остальной части главы особые средства относятся в основном к кодовым страницам, кодирующим наборы знаков ISO 8859.

Для кодовых страниц, соответствующих одному из наборов знаков ISO 8859, Emacs узнает имя этого набора, основываясь на номере кодовой страницы. Emacs автоматически создает систему кодирования для поддержки чтения и записи файлов в текущей кодовой странице и использует эту систему кодирования по умолчанию. Имя этой системы кодирования — `cp`*nnn*, где *nnn* — это номер кодовой страницы.[2]

Все системы кодирования `cp`*nnn* используют в качестве мнемоники для строки режима букву '`D`' (от "DOS"). Поскольку во время запуска и система кодирования терминала, и система кодирования для ввода/вывода файлов устанавливаются в подходящее значение `cp`*nnn*, строка режима в MS-DOS обычно начинается '`-DD\-`'. См. Раздел 1.3 [Строка режима], с. 25.

Так как кодовая страница также указывает и на то, какой вид письма вы используете, Emacs автоматически запускает `set-language-environment`, чтобы выбрать языковую среду для этой письменности (см. Раздел 18.3 [Языковые среды], с. 162).

[1] Обычно одна кодовая страница прошита в памяти дисплея, тогда как другие можно устанавливать, изменяя системные файлы конфигурации, а именно, '`CONFIG.SYS`', и перезагружаясь.

[2] Стандартные системы кодирования Emacs для ISO 8859 не подходят для этой цели, так как обычно кодовые страницы DOS не соответствуют стандартным кодам ISO. Например, буква '`ç`' ('`c`' с седилем) имеет в стандартном наборе символов Latin-1 код 231, но соответствующая кодовая страница 850 использует для этой буквы код 135.

Если буфер содержит знак, принадлежащий какому-то другому набору ISO 8859, не тому, что поддерживается выбранной кодовой страницей DOS, Emacs показывает его с помощью последовательности знаков ASCII. Например, если в текущей кодовой странице нет глифа для буквы ‘ò’ (маленькой ‘o’ с акцентом грав), она отображается как ‘{‘o}’, где фигурные скобки служат визуальным указанием на то, что это один знак. (Это может выглядеть странно с некоторыми знаками не из латиницы, а, например, из греческого или иврита; но это все же можно читать, если вы знаете язык.) Хотя знак может занимает на экране несколько столбцов, он на самом деле является одиночным знаком, и все команды Emacs понимают его как одиночный.

Не всем знакам из кодовых страниц DOS есть соответствия в ISO 8859 — некоторые используются для других целей, например для псевдографики. В Emacs нет внутреннего представления для этих знаков, поэтому когда вы считываете файл, содержащий такие знаки, они превращаются в определенные знаковые коды, задаваемые переменной `dos-unsupported-character-glyph`.

Кроме ISO 8859, Emacs поддерживает много других наборов знаков, но он не может отображать их в MS-DOS. Поэтому, если в буфере появляется один из таких многобайтных знаков, Emacs для MS-DOS показывает их, как указано в переменной `dos-unsupported-character-glyph`; по умолчанию этим глифом является пустой треугольник. Чтобы увидеть действительный код такого знака и набор, к которому он принадлежит, используйте команду `C-u C-x =`. См. Раздел 4.9 [Информация о позиции], с. 40.

По умолчанию Emacs определяет систему кодирования для поддержки текущей кодовой страницы. Чтобы определить систему кодирования для какой-то другой кодовой страницы (например, чтобы обратиться к файлу, написанному на машине с DOS в другой стране), используйте команду `M-x codepage-setup`. Она запрашивает с возможностью завершения трехзначный номер кодовой страницы, а затем создает для нее систему кодирования. Тогда для чтения и записи файла вы можете применять эту новую систему кодирования, но тогда вы должны явно задавать ее командам работы с файлами (см. Раздел 18.9 [Задание кодирования], с. 168).

Эти системы кодирования также полезны для обращения к файлам, использующим кодовые страницы DOS, когда Emacs запущен на других операционных системах.

## C.7 Подпроцессы в MS-DOS

Поскольку MS-DOS — это однозадачная “операционная система”, асинхронные подпроцессы недоступны. В частности, не работают режим Shell и его варианты. Также не работают большинство средств Emacs, использующих асинхронные подпроцессы, включая проверку правописания и GUD. Когда вы сомневаетесь, попробуйте и увидите; неработающие команды печатают сообщение об ошибке, говорящее, что асинхронные подпроцессы не поддерживаются.

Компиляция под Emacs с использованием `M-x compile`, поиск файлов с помощью `M-x grep` и показ различий между файлами с помощью `M-x diff` в действительности работают, запуская подчиненные процессы синхронно. Это означает, что вы не можете редактировать, пока подчиненный процесс не завершится.

Напротив, Emacs, скомпилированный как “родное” приложение Windows, **поддерживает** асинхронные подпроцессы. См. Раздел C.8 [Процессы под Windows], с. 411.

Команды печати, такие как `lpr-buffer` (см. Раздел 30.4 [Распечатка], с. 331) и `ps-print-buffer` (см. Раздел 30.5 [Postscript], с. 332), работают в MS-DOS, посылая вывод на один из портов принтера. См. Раздел C.5 [Печать в MS-DOS], с. 407.

Когда вы запустили синхронный процесс в MS-DOS, убедитесь, что он завершился и не пытается читать ввод с клавиатуры. Если программа не завершится сама, вы не сможете прервать ее вручную, потому что MS-DOS не предоставляет общего способа прервать процесс. Иногда в таких случаях может помочь нажатие `C-c` или `C-`⟨BREAK⟩.

Доступ к файлам на других машинах не поддерживается в MS-DOS. Другие ориентированные на работу с сетью команды, такие как отправка почты, хождение по Web, удаленный вход в систему и другие, также не работают, если в MS-DOS не встроен доступ к сети через сетевой редиректор.

Dired в MS-DOS использует пакет `ls-lisp`, где на других платформах применяется системная команда `ls`. Поэтому Dired поддерживает в MS-DOS лишь некоторые из возможных ключей, которые вы можете перечислить в переменной `dired-listing-switches`. Работают ключи '`-A`', '`-a`', '`-c`', '`-i`', '`-r`', '`-S`', '`-s`', '`-t`' и '`-u`'.

## C.8 Подпроцессы в Windows 95 и NT

Emacs, скомпилированный как "родное" приложение Windows (в противоположность версии для DOS) включает полную поддержку асинхронных подпроцессов. В версии для Windows синхронные и асинхронные подпроцессы работают нормально как в Windows 95, так и в Windows NT, до тех пор, пока вы запускаете только 32-битные приложения Windows. Однако, если вы запускаете в подпроцессе приложение DOS, вы можете столкнуться с некоторыми проблемами или вообще не суметь запустить это приложение; а если вы одновременно запустили два приложения DOS в двух подпроцессах, вам, возможно, придется перезагрузить систему.

Поскольку стандартный интерпретатор команд (и большинство утилит командной строки) в Windows 95 являются приложениями DOS, эти проблемы весьма важны. Но мы ничего не можем с этим поделать, их может исправить только Microsoft.

Если вы запустили только один подпроцесс с приложением DOS, он должен работать, как ожидается, если только он "ведет себя хорошо" и не пытается получить прямой доступ к дисплею и не делает необычных действий. Если у вас есть программа отслеживания загрузки CPU, то она покажет, что машина загружена на 100%, даже если приложение DOS ничего не делает, но это просто пережиток того способа, которым эти программы измеряют загрузку CPU.

Вы должны завершить приложение DOS до того, как запустите любое другое в отдельном подпроцессе. Emacs не может прервать или завершить подпроцесс DOS. Единственный способ остановить такой подпроцесс — дать его программе команду выхода.

Если вы попытаетесь одновременно запустить два приложения DOS в разных подпроцессах, второй будет приостановлен до завершения первого, даже если оба они асинхронные.

Если вы можете перейти в первый подпроцесс и велеть ему выйти, второй процесс должен нормально продолжить работу. Однако, если второй подпроцесс синхронный, зависнет сам Emacs до тех пор, пока не завершится первый подпроцесс. Если это не может случаться без пользовательского ввода, вам не останется ничего, кроме как перезагрузиться, если вы работаете на Windows 95. Если у вас Windows NT, вы можете использовать монитор процессов, чтобы уничтожить нужный экземпляр ntvdm (это уничтожит оба подпроцесса DOS).

Если вам приходится в такой ситуации перезагружать Windows 95, не используйте команду `Shutdown` в меню `Start`; это только подвесит систему. Вместо этого, нажмите `CTL-ALT-`⟨DEL⟩ и выберите затем `Shutdown`. Обычно это работает, хотя и может занять несколько минут.

## C.9 Использование системного меню в Windows

Emacs, собранный как "родное" приложение Windows, обычно выключает такое свойство Windows, что прижатие клавиши ⟨ALT⟩ вызывает меню. Так делается, потому что ⟨ALT⟩ работает в Emacs в качестве ⟨META⟩. Часто при использовании Emacs пользователи

временно нажимают клавишу ⟨META⟩, а потом меняют решение; если это будет вызывать меню Windows, то смысл последующих команд изменится. Многие находят это неприятным.

Вы можете снова задействовать обработку прижатия клавиши ⟨ALT⟩, принятую в Windows по умолчанию, установив `w32-pass-alt-to-system` в отличное от `nil` значение.

# Манифест GNU

Приведенный ниже Манифест GNU был написан Ричардом Столменом в начале проекта GNU, чтобы призвать к сотрудничеству и поддержке. За первые несколько лет он был немного изменен с учетом новых разработок, но сейчас кажется лучшим оставить его неизменным, каким его знает большинство.

С тех пор выяснились некоторые частые случаи недопонимания, которых можно избежать, употребляя другие слова. Добавленные в 1993 году сноски помогают прояснить эти места.

Для получения современной информации о доступных программах GNU, пожалуйста, смотрите последний выпуск Бюллетеня GNU. Этот перечень слишком длинный, чтобы приводить его здесь.

## Что такое GNU? GNU это не UNIX!

GNU, что означает Gnu's Not Unix, — это имя для полностью Unix-совместимой программной системы, которую я пишу, чтобы свободно передавать всем, кто сможет ею пользоваться.[1] Несколько добровольцев помогают мне. Крайне требуется вклад времени, денег, программ и оборудования.

На данный момент у нас уже есть текстовый редактор Emacs с языком Лисп для написания команд редактирования, отладчик, работающий на уровне исходного текста, совместимый с YACC генератор анализаторов, компоновщик и около 35-ти утилит. Оболочка (командный интерпретатор) почти завершена. Новый переносимый оптимизирующий компилятор Си скомпилировал сам себя и может быть выпущен в этом году. Начальное ядро существует, но для эмуляции Unix нужны еще многие возможности. Когда ядро и компилятор будут закончены, можно будет распространять систему GNU, пригодную для разработки программ. В качестве программы форматирования текста мы будем использовать TEX, но над nroff продолжается работа. Мы также будем использовать свободную, переносимую систему X Windows. После этого мы добавим переносимый Common Lisp, игру "Империя", электронную таблицу и сотни других вещей плюс диалоговую документацию. Мы надеемся в конце концов предоставить все полезное, что обычно поступает с системой Unix, и даже еще больше.

GNU сможет запускать программы Unix, но не будет идентична ей. Мы введем все удобные усовершенствования, основываясь на нашем опыте работы с другими операционными системами. В частности, мы планируем сделать поддержку длинных имен файлов, номеров версий файлов, защищенную от сбоев файловую систему, возможно, завершение имен файлов, поддержку терминально-независимого вывода и, в конце концов, возможно, основанную на Лиспе оконную систему, при помощи которой различные Лисп-программы и обычные программы Unix смогут совместно использовать экран. Как Си, так и Лисп будут доступны в качестве системных языков программирования. Мы постараемся обеспечить поддержку UUCP, MIT Chaosnet и протоколов связи Internet.

[1] Некоторые слова здесь были употреблены небрежно. Намерение состояло в том, чтобы никто не должен был платить за *разрешение* пользоваться системой GNU. Но эти слова не проясняют это, и люди часто интерпретируют их так, как будто они говорят, что копии GNU всегда должны распространяться бесплатно или за малую плату. Это никогда не было целью; далее манифест упоминает о возможности компаний предоставлять услуги по распространению за плату. Впоследствии я понял, что нужно точно различать "free" в смысле свободы и "free" в смысле цены. Свободное программное обеспечение — это программное обеспечение, которое пользователи вольны распространять и изменять. Некоторые пользователи могут получить копии бесплатно, тогда как другие платят за получение копий, и если эти деньги помогают улучшать программы, то чем больше цена, тем лучше. Важно то, что каждый, у кого есть копия, свободен сотрудничать с другими при ее использовании.

Изначально GNU нацелена на машины класса 68000/1600 с виртуальной памятью, так как на них ее легче всего запустить. Дополнительные усилия для запуска GNU на меньших машинах будут предоставлены тому, кто захочет использовать ее на них.

Чтобы избежать ужасной путаницы, пожалуйста, произносите ‘G’ в слове ‘GNU’, когда оно является именем данного проекта.

## Почему я должен написать GNU

Я считаю, что золотое правило требует: если мне нравится программа, то я должен поделиться ей с другими, кому она тоже нравится. Продавцы программного обеспечения хотят разделить пользователей и подчинить их себе, делая так, чтобы каждый из них соглашался не делиться с другими. Я отказываюсь нарушать солидарность с другими пользователями таким образом. Я не могу с чистой совестью подписать соглашение о нераскрытии или лицензионное соглашение по программному обеспечению. Во время моей работы в Лаборатории Искусственного Интеллекта я сопротивлялся этим тенденциям и другим препонам, но в конце концов они зашли слишком далеко: я не мог оставаться в институте, где за меня делаются такие вещи против моей воли.

Чтобы я мог продолжать использовать компьютеры с чистой совестью, я решил собрать вместе достаточное количество свободных программ и получить возможность обходиться без какого-либо несвободного программного продукта. Я ушел из Лаборатории ИИ, чтобы у МТИ не было никаких законных оснований мешать мне раздавать GNU.

## Почему GNU будет совместима с Unix

Unix не является моим идеалом системы, но она не так уж плоха. Ее основные черты, по-видимому, будут полезны, и я надеюсь восполнить пробелы Unix без того, чтобы разрушить ее основу. А система, совместимая с Unix, может быть удобна для освоения многим людям.

## Каким образом GNU станет доступна

GNU не является общественной собственностью. Каждому разрешено видоизменять и повторно распространять ее, но распространителю не разрешается препятствовать ее дальнейшему распространению. То есть, не разрешается присваивать модификации. Я хочу быть уверенным в том, что все версии GNU останутся свободными.

## Почему многие программисты хотят помочь

Я встретил много программистов, которые заинтересовались GNU и захотели помочь.

Многих программистов не устраивает коммерциализация системных программных продуктов. Она может дать им возможность заработать больше денег, но заставляет их чувствовать себя соперниками других программистов, а не товарищами. Основной дружеский акт среди программистов — совместное использование программ; типичные маркетинговые соглашения, используемые сегодня, по существу запрещают программистам относится друг к другу по-дружески. Покупатель программного продукта должен выбирать между дружбой и подчинением закону. Естественно, многие решат, что дружба важнее. Но те, кто верит в закон, чувствуют себя неловко, сделав какой-либо выбор. Они становятся циничными и думают, что программирование — это просто способ делать деньги.

Работая над GNU и используя ее, а не принадлежащие кому-либо программы, мы можем быть благожелательны ко всем и в то же время соблюсти закон. Кроме того, GNU служит вдохновляющим примером и знаменем, сплачивающим остальных вокруг нас для

совместного использования программ. Это может дать нам ощущение гармонии, которое невозможно, если мы используем несвободный программный продукт. Для почти половины программистов, с которыми я разговаривал, это неоценимое счастье, которое деньги заменить не могут.

## Каким образом вы можете внести свой вклад

Я прошу производителей компьютеров о пожертвованиях в виде денег или машин. Я прошу отдельных людей о вкладе программами или работой.

Одно из следствий того, что вы внесли свой вклад машинами, это то, что на них GNU будет запущена в кратчайшие сроки. Машины должны быть укомплектованными, готовыми к использованию системами, пригодными для использования в жилом помещении, и не должны нуждаться в слишком сложных системах охлаждения и питания.

Я нашел очень много программистов, готовых потратить часть своего рабочего времени для работы над GNU. Для большинства проектов такую распределенную работу с неполной занятостью очень трудно координировать; независимо написанные части, возможно, не заработают вместе. Но для частной задачи замещения Unix это проблема отсутствует. Полная система Unix содержит сотни утилит, каждая из которых документирована отдельно. Большинство интерфейсов фиксированы совместимостью с Unix. Если каждый сотрудник сможет написать совместимую замену для одной Unix-утилиты и сделает так, чтобы она работала вместо оригинала в системе Unix, тогда вместе эти утилиты будут работать надлежащим образом. Даже если позволить Мерфи создать несколько неожиданных проблем, объединение этих компонент будет осуществимой задачей. (Ядро требует более тесной взаимосвязи и будет разрабатываться небольшой компактной группой).

В случае получения мною денежных пожертвований, я буду в состоянии нанять нескольких человек на полный или неполный рабочий день. Оклад будет невысоким по программистским меркам, но я ищу таких людей, для которых укрепление духа общности так же важно, как и зарабатывание денег. Я рассматриваю это как способ дать увлеченным людям возможность отдать всю свою энергию работе над GNU, оберегая их от необходимости зарабатывать себе на жизнь другим путем.

## Почему все пользователи компьютеров получат выгоду

Как только GNU будет написана, каждый сможет получить хороший свободный программный продукт так же свободно, как воздух.[2]

Это означает гораздо больше, чем просто экономию каждому стоимости лицензии на использование Unix. Это означает, что будет устранена большая часть расточительного дублирования усилий по системному программированию. Эти усилия смогут пойти вместо этого на продвижение технологии.

Полные исходные коды системы будут доступны каждому. В результате тот пользователь, которому нужны изменения в системе, всегда сможет беспрепятственно сделать их сам или нанять программистов или компанию, которые взялись бы сделать их для него. Пользователи больше не будут во власти одного программиста или компании, которые владеют исходными кодами и находятся в исключительном положении в смысле внесения изменений.

Учебные заведения смогут обеспечить более мощную образовательную среду, поощряя всех студентов изучать и улучшать системные коды. Гарвардская компьютерная лаборатория раньше проводила следующую политику: никакая программа не могла быть установлена в системе, если все ее исходные коды не были предоставлены на всеобщее обозрение;

[2] Это еще одно место, где я не выявил тщательно разницу между двумя различными значениями слова “free”. Само по себе это предложение не является неправдой — вы можете получить копии программ GNU бесплатно от ваших друзей или по сети. Но оно предполагает ложную идею.

и это фактически поддерживалось отказом устанавливать определенные программы. Это меня очень вдохновило.

Наконец, будут ликвидированы расходы на рассмотрение того, кто владеет системным программным продуктом, и что он вправе или не вправе делать с ним.

Соглашения о том, что люди должны платить за пользование программой, включая лицензирование копий, всегда влекут за собой громадные затраты для общества из-за громоздких механизмов, необходимых для подсчета того, сколько (то есть, за какие программы) должен платить человек. И только полицейское государство может заставить каждого подчиниться им. Рассмотрим космическую станцию, где производство воздуха должно стоить очень дорого: взимание с каждого живого существа платы за литр воздуха может быть справедливо, но ношение противогаза со счетчиком и день и ночь невыносимо, даже если каждый в состоянии заплатить по счету за воздух. А телевизионные камеры повсюду, чтобы следить, не снимаете ли вы противогаз, возмутительны. Лучше уж содержать завод по производству воздуха на средства от поголовного налога и сбросить противогазы.

Копирование всей или части программы присуще программисту как дыхание и также плодотворно. И оно должно быть столь же свободным.

## Некоторые легко опровергаемые возражения против целей GNU

> “Никто не будет ее использовать, если она будет бесплатной, так как это означает, что пользователь не сможет положиться ни на какое сопровождение.”
>
> “Вы должны будете назначить цену за программу, чтобы оплатить обеспечение сопровождения.”

Если люди охотнее заплатят за GNU плюс обслуживание вместо получения GNU бесплатно и без обслуживания, то компания по обеспечению только обслуживания тех, кто получил GNU бесплатно, может стать прибыльной.[3]

Мы должны различать сопровождение в виде действительной работы по программированию и простую поддержку. Первое — это нечто, в чем нельзя положиться на продавца программного продукта. Если ваши проблемы не разделяются достаточным количеством людей, то продавец скажет вам, чтобы вы убирались.

Если ваша фирма нуждается в возможности положиться на сопровождение, то существует единственный выход — иметь все необходимые исходные коды и сервисные программы. Тогда вы можете нанять любого, кто сможет решить вашу проблему; вы не зависите ни от чьей милости. В случае Unix цена исходных кодов выводит это из рассмотрения для большинства компаний. С GNU это будет просто. Вполне возможно, что при этом не найдется достаточно компетентного человека, но вина за эту проблему не может возлагаться на соглашения по распространению. GNU не решает все мировые проблемы, а только некоторые из них.

Тем временем, пользователи, которые ничего не знают о компьютерах, нуждаются в поддержке: в выполнении для них того, что они могли бы с легкостью сделать сами, но не знают как.

Такие услуги могут предоставить компании, которые продают только поддержку и ремонтное обслуживание. Если правда, что пользователи скорее потратят деньги и получат продукт с обслуживанием, то они также будут готовы покупать обслуживание, получив продукт бесплатно. Компании по обслуживанию будут конкурировать в качестве и цене; пользователи не будут привязаны к какой-то одной из них. Тем временем те из нас, кто не нуждается в обслуживании, смогут пользоваться программой без его оплаты.

[3] Сейчас существуют несколько таких компаний.

"Вы не сможете заинтересовать многих людей, не используя рекламу, а чтобы окупить ее, вам придется назначить цену за программу."

"Бесполезно рекламировать программу, которую люди могут получить бесплатно."

Существуют различные виды бесплатной или очень дешевой рекламы, которая может быть использована для информирования большого числа пользователей о чем-то подобном GNU. Но может быть правда и то, что можно заинтересовать большее число пользователей микрокомпьютеров с помощью рекламных объявлений. Если это действительно так, то фирма, которая рекламирует услуги по копированию и пересылке GNU за плату, должна быть достаточно процветающей, чтобы платить за рекламу, и даже более того. Тогда только пользователи, которые извлекают пользу из рекламы, платят за нее.

С другой стороны, если многие получают GNU от своих друзей, и подобные фирмы не будут процветать, то это покажет, что реклама не так уж необходима для распространения GNU. Почему же сторонники свободного рынка не хотят позволить решить это ему самому?[4]

"Моей компания нужна собственная операционная система, чтобы получить преимущество в конкурентной борьбе."

GNU выведет программное обеспечение для операционных систем из сферы конкурентной борьбы. Вы не сможете получить преимущество в этой области, но и ваши конкуренты также не смогут получить преимущество над вами. Вы будете конкурировать с ними в других областях, совместно извлекая выгоду в этой области. Если ваш бизнес — это продажа операционных систем, то вам GNU не понравится, но это ваши трудности. Если ваш бизнес состоит в чем-нибудь еще, то GNU сможет оградить вас от втягивания в дорогостоящий бизнес по продаже операционных систем.

Я хотел бы увидеть дальнейшее развитие GNU, поддерживаемое дарами от многих производителей и пользователей, это сократит затраты для каждого.[5]

"Разве программисты не заслуживают вознаграждения за свое творчество?"

Если уж что-то и заслуживает вознаграждения, так это общественный вклад. Творчество может быть общественным вкладом, но только до тех пор, пока общество может свободно пользоваться его результатами. Если программисты заслуживают вознаграждения за создание прогрессивных программ, то также они заслуживают и наказания, если они ограничивают использование этих программ.

"Может ли программист просить вознаграждение за свое творчество?"

Нет ничего плохого в том, что кто-то требует плату за работу или стремится увеличить свой доход, до тех пор, пока он не пользуется разрушительными средствами. Но сегодня привычные средства в области программного обеспечения основываются именно на разрушении.

Выжимание денег из пользователей программы, ограничивая их возможности использовать ее, — это и есть разрушение, так как ограничения уменьшает объем и способы использования программы. Это сокращает объем прибыли, которую человечество извлекает из программы. Когда кто-то предумышленно вводит ограничения, то вредным последствием этого является преднамеренное разрушение.

---

[4] Фонд Свободного Программного Обеспечения получает большую часть средств за услуги по распространению, хотя он является организацией, существующей на пожертвования, а не коммерческой компанией. Если *никто* не выберет в качестве способа получить копии заказ от Фонда, он будет неспособен выполнять свои работу. Но это не означает, что собственнические ограничения оправданы, чтобы заставить каждого пользователя платить. Если малая часть всех пользователей заказывает копии в Фонде, этого достаточно, чтобы Фонд остался на плаву. Поэтому мы просим пользователей поддержать нас таким образом. Сделали ли вы свою часть?

[5] Группа компьютерных компаний недавно выдала средства на поддержку сопровождения компилятора GNU C Compiler.

Причина того, что добросовестные граждане не пользуются такими разрушительными средствами, чтобы стать богаче, состоит в том, что если каждый бы поступал так, то мы все вместе стали бы беднее от всеобщего разрушения. Это этика Канта, или Золотое правило. Так как мне не нравятся последствия, проистекающие из того, что кто-то припрятывает информацию, я вынужден считать, что тот, кто поступает так, неправ. В частности, желание быть вознагражденным за свое творчество не оправдывает лишение общества вообще всего творения или его части.

"Не будут ли программисты голодать?"

На это я могу ответить, что никого не заставляют быть программистом. Большинство из нас не смогут ухитриться зарабатывать деньги, стоя на улице и корча гримасы. Но в результате нас же никто не заставляет провести свою жизнь, стоя на улице, корча гримасы и голодая. Мы делаем что-то другое.

Но это неправильный ответ, так как он признает подразумеваемое в вопросе предположение: что без собственности на программное обеспечение программист не сможет получить ни цента. По общему мнению — все или ничего.

Действительная же причина, по которой программисты не будут голодать, состоит в том, что по-прежнему будет существовать возможность получать деньги за программирование, просто не так много, как сейчас.

Ограничение копирования не является единственной основой бизнеса в области программного обеспечения. Это самая распространенная основа, так как она приносит больше всего денег. Если бы это было запрещено или отвергнуто покупателями, то бизнес в области программного обеспечения перешел бы на другие организационные принципы, которые в данный момент используются не так часто. Всегда существует множество способов организации любого вида бизнеса.

Вероятнее всего, программирование с новыми организационными принципами не будет таким прибыльным, как сейчас. Но это не аргумент против перемен. Не считается же несправедливым жалование, которое получают клерки по продаже сейчас. Если бы программисты получали столько же, то это также не будет несправедливым. (На практике, они будут все так же получать значительно больше, чем клерки.)

"Разве люди не имеют права контролировать, как используется их творение?"

"Контроль над использованием чьих-то идей" на деле вводит контроль над жизнью других людей, и он обычно используется, чтобы сделать их жизнь более трудной.

Люди, которые внимательно изучали происхождение права на интеллектуальную собственность (например, юристы), говорят, что не существует внутренне присущего права на интеллектуальную собственность. Виды предполагаемых прав на интеллектуальную собственность, которые признаются правительством, были созданы особыми законодательными актами для специальных целей.

Например, патентная система была создана для поощрения того, чтобы изобретатели раскрывали детали своих изобретений. Это должно было помочь обществу, а не изобретателям. В то время семнадцатилетний период существования патента был коротким по сравнению с темпами повышения уровня развития. Так как патенты затрагивают только предпринимателей, для которых затраты усилий и денег на лицензионное соглашение малы по сравнению с расходами на запуск производства, то патенты зачастую не наносят ощутимого вреда. Они не мешают большинству людей, которые используют запатентованные продукты.

Идея авторских прав не существовала в древности, когда авторы нередко копировали других авторов со всеми подробностями в работах, которые не относились к художественной литературе. Такая практика была полезна, и только благодаря ей многие авторские работы сохранились хотя бы частично. Система авторских прав была создана специально для того, чтобы поощрять авторство. Сфера ее действия — это книги, которые могли

быть экономно размножены только при помощи печатного станка. Это приносило мало вреда и не мешало большинству людей, которые читали книги.

Все права интеллектуальной собственности — это просто лицензии, предоставленные обществом, так как считается, правильно или ошибочно, что все общество в целом будет извлекать выгоду, предоставив их. Но в каждом конкретном случае мы должны задаваться вопросом: действительно ли мы станем богаче, выдав такую лицензию? Какие именно действия человека мы лицензируем?

Сегодняшняя ситуация с программным обеспечением очень сильно отличается от положения с книгами сто лет назад. Факты, состоящие в том, что простейший способ копирования программы — это передача ее одним соседом другому, что программы имеют как исходные, так и объектные коды, различающиеся между собой, что программы используются для работы, а не для чтения и удовольствия, вместе создают ситуацию, в которой лицо, настаивающее на авторских правах, наносит ущерб обществу в целом как в материальном, так и в духовном плане, и в которой человек не должен поступать так, независимо от того, позволяет ли ему это закон или нет.

“Конкуренция заставляет все делать лучше и лучше.”

Наглядный пример конкуренции — это гонки: награждая победителя, мы заставляем других бежать быстрее. Когда капитализм работает именно так, он делает нужное дело. Но его защитники ошибаются, полагая, что он всегда так работает. Если бегуны забывают, за что именно предлагается награда, и стремятся выиграть любой ценой, то они могут найти другую стратегию, такую, например, как расталкивание других бегунов. Если бегуны для начала ударятся в кулачный бой, то все они придут к финишу позже.

Частный и засекреченный программный продукт — это моральный эквивалент бегунов в кулачном бою. Досадно говорить об этом, но единственный рефери, который у нас есть, по всей видимости не возражает против драк. Он просто регулирует их (“На каждые 10 ярдов, которые вы пробежите, вы можете сделать один удар”). В действительности же он должен прекращать их и наказывать бегунов даже за попытку драки.

“Не прекратят ли все программировать без денежного вознаграждения?”

На самом деле многие будут заниматься программированием совершенно без денежного вознаграждения. Программирование имеет непреодолимое очарование для некоторых людей, обычно для тех, кому это лучше всего удается. Нет недостатка в профессиональных музыкантах, которые упорно продолжают заниматься музыкой, даже несмотря на то, что они не надеются зарабатывать этим на жизнь.

В действительности этот вопрос, несмотря на то, что он часто задается, не подходит к ситуации. Оплата программирования не исчезнет, просто она станет меньше. Так что правильно ставить вопрос следующим образом: будет ли кто-нибудь программировать за уменьшенное денежное вознаграждение? Мой опыт подсказывает мне, что такие люди найдутся.

В течение более чем десяти лет многие из лучших программистов работали в Лаборатории Искусственного Интеллекта, получая гораздо меньше денег, чем они могли бы иметь где-то еще. Они получили массу неденежных вознаграждений: славу и уважение, например. Кроме того, творчество — это развлечение, само являющееся наградой.

В дальнейшем многие из них ушли, когда появился шанс делать столь же интересную работу за большие деньги.

Эти факты демонстрируют, что люди будут заниматься программированием не только ради обогащения. Но если кроме того появится шанс заработать много денег, то они будут надеяться на них и требовать их. Организации с низкой оплатой слабо конкурируют с организациями, где она высокая, но дела у них не должны быть плохи, если организации с высокой оплатой будут запрещены.

“Нам отчаянно нужны программисты. Если они потребуют, чтобы мы прекратили помогать нашим соседям, то мы должны будем подчиниться.”

Вы никогда не будете в таком отчаянном положении, чтобы вас можно было подчинить требованиям такого сорта. Запомните: миллион на защиту, но ни цента на дань!

"Программистам как-то же надо зарабатывать на жизнь."

На первый взгляд это правильно. Однако, существует много способов, которыми программист мог бы заработать на жизнь, не продавая права на использование программы. Сейчас этот путь привычен, так как он дает программистам и бизнесменам самые большие деньги, а не потому, что это единственный путь заработать на жизнь. Легко найти другие пути, если вы захотите найти их. Вот несколько примеров.

Производители, внедряя новые компьютеры, будут платить за перенос операционных систем на новое оборудование.

Программистов также можно было бы использовать при продаже услуг по обучению, сопровождению и текущему обслуживанию.

Люди с новыми идеями могли бы распространять свободное программное обеспечение, спрашивая пожертвования от удовлетворенных пользователей или продавая текущее обслуживание. Я встречал людей, которые уже успешно работают таким образом.

Пользователи со схожими потребностями могут образовывать пользовательские группы и платить членские взносы. Группа заключала бы контракт с программистскими компаниями для написания программ, которыми хотели бы пользоваться члены такой группы.

Все виды развития могут финансироваться с помощью налога на программный продукт:

> Предположим, что каждый, купивший компьютер, должен заплатить x процентов от его цены в качестве налога на программное обеспечение. Правительство передает эти деньги агентству, подобному Национальному научному фонду, чтобы оно потратило их на развитие программного обеспечения.
>
> Но если покупатель компьютера сделает пожертвование на развитие программного обеспечения сам, то он может получить кредит в уплату налога. Он может сделать денежное пожертвование на проект по своему выбору, причем выбор проекта часто объясняется тем, что он надеется воспользоваться его результатами. Он может получить кредит на любую сумму пожертвования вплоть до общей суммы налога, которую он должен заплатить.
>
> Полная ставка налога может определяться голосованием налогоплательщиков, при этом налог определяется пропорционально сумме, с которой он берется.
>
> Следствия:
>
> - Общество, использующее компьютеры, поддерживает развитие программного обеспечения.
> - Общество решает, какой уровень поддержки требуется.
> - Пользователи, которых волнует, на какой проект пойдет их доля, могут сами выбрать его.

В конце концов, создание свободных программ — это шаг по направлению к постдефицитному миру, где никто не должен будет работать очень напряженно, чтобы просто прожить. Люди смогут посвятить себя тому виду деятельности, который доставляет им удовольствие, например программированию, отработав требуемые десять часов в неделю на необходимых заданиях, таких как законотворчество, семейные советы, починка роботов и астероидные исследования. Не надо будет зарабатывать на жизнь программированием.

Мы уже имеем значительное сокращение объема работ, которые общество в целом должно делать для обеспечения реальной производительности, но только малая часть этого преобразуется в досуг для работников, так как требуется много деятельности в непроизводственной сфере для сопровождения производственной деятельности. Основная причина этого — бюрократия и соответственные усилия на борьбу с конкуренцией. Свободное

программное обеспечение значительно сократит эти расходы в области производства программных продуктов. Мы должны сделать это, чтобы повышение производительности преобразовывалось в сокращение работы для нас.

# Глоссарий

Абзац
: Абзац — это единица среднего размера в тексте на естественном языке. Существуют специальные команды для перемещения по абзацам и для действий с ними. См. Раздел 21.3 [Абзацы], с. 183.

Аргумент по умолчанию
: Значение аргумента по умолчанию — это значение, которое будет предполагаться, если вы не зададите его сами. Когда для чтения аргумента используется минибуфер, аргумент по умолчанию будет использоваться, если вы просто набираете ⟨RET⟩. См. Глава 5 [Минибуфер], с. 45.

Балансировка круглых скобок
: В Emacs можно сбалансировать скобки автоматически или вручную. Ручная балансировка производится командами, используемыми для перемещения по сбалансированным выражениям (см. Раздел 22.2 [Списки], с. 206). Автоматическая балансировка производится при помощи мигания или подсвечивания скобки, которая соответствует только что вставленной. (см. Раздел 22.6 [Парные скобки], с. 218).

Блокирование файлов
: Блокирование файлов нужно для выдачи предупреждения, когда вы начинаете изменять файл, который уже редактирует кто-то другой. См. Раздел 14.3.2 [Одновременное редактирование], с. 112.

Буфер
: Буфер — это основная единица редактирования; один буфер соответствует одному куску редактируемого текста. Вы можете иметь несколько буферов, но в каждый конкретный момент вы редактируете только один, ‘выбранный’ буфер, хотя когда вы используете несколько окон (см.), несколько буферов могут быть видимыми. Чаще всего буферы обращаются (см.) к какому-либо файлу. См. Глава 15 [Буферы], с. 135.

Буфер только для чтения
: Буфер только для чтения — это такой буфер, чей текст вам не разрешается изменять. Обычно Emacs создает буферы, доступные только для чтения, когда они содержат текст, который имеет особое значение для Emacs; например, буферы Dired. Обращение к защищенному от записи файлу также создает буфер только для чтения. См. Глава 15 [Буферы], с. 135.

Верхний уровень
: Верхний уровень — это нормальное состояние Emacs, в котором вы редактируете текст файла, к которому обратились. Вы бываете на верхнем уровне, если вы не находитесь на уровне рекурсивного редактирования (см.), или в минибуфере (см.) или в середине команды. Вы можете вернуться на верхний уровень при помощи прерывания (см.) или выхода (см.). См. Раздел 32.1 [Выход], с. 371.

Восстановление
: Восстановление означает вставку прежде уничтоженного текста. Это может использоваться для отмены ошибочного уничтожения или для копирования или перемещения текста. В некоторых других системах это называется “вставкой”. См. Раздел 9.2 [Восстановление], с. 71.

Вставка
: Вставка — это копирование текста в буфер, либо с клавиатуры, либо из другого места в Emacs.

Вторичное выделение
: Вторичное выделение — это одно конкретное выделение (см.) в X Windows; некоторые приложения в X Window могут использовать его для передачи тек-

ста другим приложениям. В Emacs есть особые команды мыши для работы с вторичным выделением. См. Раздел 17.2 [Вторичное выделение], с. 149.

Второстепенный режим
: Второстепенный режим — это необязательное средство Emacs, которое может быть включено или выключено независимо от всех других возможностей Emacs. Каждый второстепенный режим имеет команды включения и выключения. См. Раздел 31.1 [Второстепенные режимы], с. 341.

Входной почтовый ящик
: Входной почтовый ящик — это файл, в который операционная система доставляет почту. Rmail перемещает почту из файлов входных почтовых ящиков в Rmail-файлы (см.), в которых почта затем хранится постоянно или до тех пор, пока она не будет явно удалена. См. Раздел 27.5 [Входной файл Rmail], с. 278.

Выбор
: Выбор буфера делает его текущим (см.) буфером. См. Глава 15 [Буферы], с. 135.

Выделение
: Система X Windows позволяет приложению определить именованные выделения, значениями которых является какой-то текст. Программа также может считывать выделения, которые установили другие программы. Это основной способ обмена текстом между оконными приложениями. В Emacs есть команды для работы с первичным (см.) и вторичным выделением (см.).

Выход
: Выход — это отмена частично набранной или запущенной команды с помощью `C-g` (или `C-`⟨BREAK⟩ в MS-DOS). См. Раздел 32.1 [Выход], с. 371.

Выравнивание
: Выравниванием называют добавление дополнительных пробелов к строкам текста, чтобы сделать их точно определенной ширины. См. Раздел 21.5 [Заполнение], с. 185.

Вычеркивание
: Вычеркивание в Rmail-файле или буфере Dired — это операция, которая в действительности удаляет сообщения или файлы, которые вы ранее пометили для удаления.

Глобальный
: Глобальный означает “не зависящий от текущей среды, действующий на весь Emacs”. Это противоположность локальному (см.). Конкретные примеры использования термина ‘глобальный’ появятся ниже.

Глобальная переменная
: Глобальное значение переменной (см.) действует во всех буферах, которые не имеют своих собственных локальных значений для этой переменной. См. Раздел 31.2 [Переменные], с. 343.

Глобальная подстановка
: Глобальная подстановка означает замену каждого экземпляра какой-то строки другой строкой по всему большому куску текста. См. Раздел 12.7 [Замена], с. 95.

Глобальная таблица ключей
: Глобальная таблица ключей (см.) содержит привязки ключей, которые действуют всегда, кроме случаев, когда они перекрываются локальными привязками в локальной таблице ключей основного режима (см.). См. Раздел 31.4.1 [Таблицы ключей], с. 356.

Глобальное сокращение
: Глобальное определение сокращения (см.) действует во всех основных режимах, которые не имеют локальных (см.) определений для тех же самых сокращений. См. Глава 24 [Сокращения], с. 257.

Глобальный список пометок
: Глобальный список пометок записывает последовательность буферов, в которых вы недавно устанавливали метку. Во многих случаях вы можете использовать это для прохода по буферам, которые вы редактируете, или в которых вы находили теги. См. Раздел 8.6 [Глобальный список пометок], с. 67.

Графический знак
: Графические знаки — это такие знаки, для которых определено изображение, а не только имя. Все не-Meta-знаки (см.), за исключением Control-знаков (см.), являются графическими. Они включают буквы, цифры, знаки пунктуации и пробелы; сюда не входят ⟨RET⟩ или ⟨ESC⟩. В Emacs, ввод графического знака вставляет этот знак (в обычных режимах редактирования). См. Глава 4 [Основы редактирования], с. 35.

Заблокированная команда
: Заблокированная команда — это команда, которую вы не можете запустить без специального подтверждения. Обычной причиной блокирования команды, является то, что она может запутать начинающего пользователя. См. Раздел 31.4.11 [Блокирование команды], с. 364.

Завершение
: Завершение — это то, что делает Emacs, когда он автоматически раскрывает сокращенное имя в полное имя. Завершение выполняется для аргументов минибуфера (см.), когда набор возможного допустимого ввода известен; например, для имен команд, буферов и файлов. Завершение осуществляется, когда набираются ⟨TAB⟩, ⟨SPC⟩ или ⟨RET⟩. См. Раздел 5.3 [Завершение], с. 47.

Законченный ключ
: Законченный ключ — это последовательность ключей, которая полностью определяет одно действие, которое выполнит Emacs. Например, `X`, `C-f` и `C-x m` — законченные ключи. Законченные ключи получают смысл после привязки (см.) к командам (см.). Таким образом, `X` обычно привязана к команде, которая вставляет в буфер ‘`X`’. `C-x m` обычно привязывается к команде, которая начинает составление почтового сообщения. См. Раздел 2.2 [Ключи], с. 30.

Замена
: Смотрите ‘глобальная подстановка’.

Захват файла
: Emacs использует захват файлов, чтобы заметить, когда два разных пользователя начинают редактировать один файл одновременно. См. Раздел 14.3.2 [Захват файлов], с. 112.

Знак
: Знаки составляют содержимое буфера Emacs; смотрите Раздел 2.4 [Текстовые знаки], с. 31. Кроме того, последовательности ключей (см.), обычно составляются из знаков (однако они могут также включать и другие события ввода). См. Раздел 2.1 [Пользовательский ввод], с. 29.

Знак новой строки
: Смотрите ‘перевод строки’.

Измененный буфер
: Буфер (см.) является измененным, если его текст изменялся с момента последнего сохранения (или с момента его создания, если он никогда не записывался). См. Раздел 14.3 [Сохранение], с. 108.

Именованная метка
: Именованная метка — это регистр (см.), используемый для записи позиции в буфере, чтобы вы могли переместить точку к этой позиции. См. Глава 10 [Регистры], с. 77.

Имя команды
: Имя команды — это имя лисповского символа, являющегося командой (см. Раздел 2.3 [Команды], с. 31). Вы можете вызвать любую команду по имени, используя `M-x` (см. Глава 6 [M-x], с. 53).

Имя файла
: Имя файла — это название, по которому ссылаются на файл. Имена файлов бывают относительными и абсолютными; смысл относительного имени файла зависит от текущего каталога, но абсолютное имя ссылается на один и тот же файл вне зависимости от того, какой каталог является текущим. В системах GNU и Unix, абсолютное имя файла начинается с косой черты (корневого каталога), или с ‘`~/`’ или ‘`~`*пользователь*`/`’ (начальный каталог).

История выбранных буферов
: Emacs хранит историю выбранных буферов, которая записывает, как давно выбирался каждый буфер Emacs. Это используется для определения того, какой буфер выбрать. См. Глава 15 [Буферы], с. 135.

История минибуфера
: В истории минибуфера записан текст, который вы задавали ранее в качестве аргументов минибуфера, чтобы вы могли удобно использовать тот же самый текст еще раз. См. Раздел 5.4 [История минибуфера], с. 49.

Каталог
: Каталоги файлов — это именованные группы в файловой системе, в которые вы можете помещать отдельные файлы или подкаталоги. См. Раздел 14.8 [Каталоги], с. 131.

Каталог по умолчанию
: Когда вы задаете имя файла, не начинающееся с ‘`/`’ или ‘`~`’, то оно интерпретируется относительно каталога по умолчанию для текущего буфера. См. Раздел 5.1 [Каталог по умолчанию], с. 45.

Команда
: Команда — это лисповская функция, специально определенная так, чтобы она могла служить в Emacs привязкой к ключу. Когда вы набираете последовательность ключей (см.), его привязка (см.) ищется в соответствующих таблицах ключей (см.), чтобы определить, какая команда должна быть запущена. См. Раздел 2.3 [Команды], с. 31.

Комментарий
: Комментарий — это текст в программе, который предназначен только для людей, читающих программу, и который специально помечен таким образом, что он игнорируется во время загрузки программы и ее компиляции. Emacs предлагает специальные команды для создания, выравнивания и уничтожения комментариев. См. Раздел 22.7 [Комментарии], с. 219.

Компиляция
: Компиляция — это процесс создания исполняемой программы из исходных кодов. В Emacs существуют команды для компиляции файлов с кодами на языке Emacs Lisp (см. раздел “Byte Compilation” в *the Emacs Lisp Reference Manual*) и программ на Си и других языках (см. Раздел 23.1 [Компиляция], с. 247).

Компонент имени файла
: Один компонент имени файла ссылается на файл, находящийся непосредственно в конкретном каталоге. В системах GNU и Unix, имя файла — это последовательность компонентов, разделенных косыми чертами. Например, ‘`foo/bar`’

— это имя файла, содержащее два компонента, ‘`foo`’ и ‘`bar`’; оно ссылается на файл с именем ‘`bar`’ в каталоге с именем ‘`foo`’ в текущем каталоге.

Косвенный буфер
: Косвенный буфер — это буфер, который разделяет текст с другим буфером, называемым его базовым буфером. См. Раздел 15.6 [Косвенные буферы], с. 139.

Курсор
: Курсор — это прямоугольник на экране, который показывает позицию, именуемую точкой (см.), где происходит вставка и удаление текста. Курсор находится на знаке, который идет вслед за точкой, или под ним. Часто люди говорят ‘курсор’, когда, строго говоря, имеют в виду ‘точку’. См. Глава 4 [Основы редактирования], с. 35.

Локальный
: Локальный означает ‘действующий только в каком-то отдельном контексте’. Уместный вид контекста — это выполнение некоторой отдельной функции, отдельный буфер или отдельный основной режим. Этот термин противоположен термину ‘глобальный’ (см.). Конкретные применения термина ‘локальный’ в терминологии Emacs появятся ниже.

Локальная переменная
: Локальное значение переменной (см.) применяется только к одному буферу. См. Раздел 31.2.4 [Локальные переменные], с. 350.

Локальная таблица ключей
: Локальная таблица ключей используется в отдельном основном режиме. Привязка ключей (см.) в текущей локальной таблице ключей перекрывает глобальные привязки тех же самых последовательностей ключей. См. Раздел 31.4.1 [Таблицы ключей], с. 356.

Локальное сокращение
: Определение локального сокращения действует, только если выбирается какой-то определенный основной режим. В этом основном режиме игнорируется любое глобальное определение для того же самого сокращения. См. Глава 24 [Сокращения], с. 257.

Макро клавиатуры
: Макросы клавиатуры — это способ определения новых команд Emacs из последовательностей уже существующих без необходимости написания программы на Лиспе. См. Раздел 31.3 [Макросы клавиатуры], с. 353.

Метка
: Метка указывает на позицию в тексте. Она определяет один конец области (см.), точка находится в другом конце области. Многие команды оперируют с текстом целиком от точки до метки. Каждый буфер имеет свою метку. См. Глава 8 [Пометка], с. 63.

Метод ввода
: Метод ввода — это система ввода текстовых знаков, не входящих в ASCII, путем набора последовательности ASCII-знаков (см.). См. Раздел 18.4 [Методы ввода], с. 163.

Метод составления сообщений
: Метод составления сообщений — это программа, запускаемая в Emacs для редактирования и отправки почтового сообщения. Emacs позволяет вам выбирать из нескольких различных методов составления сообщений. См. Раздел 26.6 [Почтовые методы], с. 274.

Минибуфер
: Минибуфер — это окно, которое появляется, когда это необходимо, внутри эхообласти (см.), он используется для чтения аргументов команд. См. Глава 5 [Минибуфер], с. 45.

Многобайтный знак
: Многобайтный знак — это знак, который занимает в буфере несколько позиций. Emacs использует многобайтные знаки для представления текста, не выражаемого через ASCII, поскольку число не-ASCII-знаков гораздо больше 256. См. Раздел 18.1 [Введение в MULE], с. 161.

Модифицированный буфер
: Смотрите ‘измененный буфер’.

Набор знаков
: Emacs поддерживает много наборов знаков, каждый из которых представляет некий алфавит или письменность. См. Глава 18 [MULE], с. 161.

Нажатие на кнопку
: Нажатие на кнопку — это вид события ввода, генерируемый, когда вы нажимаете кнопку мыши. См. Раздел 31.4.10 [Кнопки мыши], с. 363.

Настройка
: Настройка — это произведение мелких изменений в работе Emacs. Чаще всего она делается при помощи установки переменных (см. Раздел 31.2 [Переменные], с. 343) или при помощи перепривязки последовательностей ключей (см. Раздел 31.4.1 [Таблицы ключей], с. 356).

Область
: Область — это текст между точкой (см.) и меткой (см.). Многие команды оперируют с текстом в области. См. Глава 8 [Пометка], с. 63.

Обращение
: Обращение к файлу означает загрузку его содержимого в буфер (см.), где его можно редактировать. См. Раздел 14.2 [Обращение], с. 106.

Ограничение
: Ограничение буфера — это количество текста в начале или конца буфера, которое временно является невидимым или недоступным. Придание буферу ненулевого ограничения называется сужением (см.). См. Раздел 30.8 [Сужение], с. 335.

Одновременное редактирование
: Одновременное редактирование означает, что сразу два пользователя модифицируют один и тот же файл. Одновременное редактирование, если оно не замечено вовремя, может привести к потере работы одного из пользователей. Emacs обнаруживает все случаи одновременного редактирования и предупреждает одного из пользователей, чтобы он разобрался. См. Раздел 14.3.2 [Одновременное редактирование], с. 112.

Окно
: Emacs делит фрейм (см.) на одно или несколько окон, каждое из которых в любой момент может показывать содержимое одного буфера (см.). См. Глава 1 [Экран], с. 23, для основной информации о том, как Emacs использует экран. См. Глава 16 [Окна], с. 141, для информации о командах управления окнами.

Определение функции
: Определение функции — это список в программе на верхнем уровне скобок. Они получили такое название, так большинство подобных списков в программах на Лиспе — это вызовы лисповской функции `defun`.[1] См. Раздел 22.4 [Определения функций], с. 208.

Основной режим
: Основные режимы Emacs — это взаимоисключающие наборы параметров, каждый из которых настраивает Emacs для редактирования текста определенно-

[1] От английского ‘define function’. *(Прим. переводчика)*

го вида. В идеале, каждый язык программирования имеет свой собственный основной режим. См. Глава 19 [Основные режимы], с. 175.

Отмена
: Отмена означает выполнение вашего предыдущего редактирования в обратном направлении, при этом возвращается текст, существовавший ранее в этом сеансе редактирования. См. Раздел 4.4 [Отмена], с. 37.

Отмена особого смысла знака
: Отмена особого смысла знака означает лишение знака присущего ему специального значения. В Emacs обычно это делается с помощью `C-q`. Что именно составляет специальное значение зависит от контекста и от условий. Например, “обыкновенный” знак, рассматриваемый как команда Emacs, вставляет сам себя; следовательно, в этом контексте специальный знак — это любой знак, который обычно не вставляет себя (например, ⟨DEL⟩), а отмена особого смысла делает его самовставляющимся, как если бы он не был специальным знаком. Отмена особого смысла допускается не во всех контекстах. См. Глава 4 [Основы редактирования], с. 35.

Отмена особого смысла в имени файла
: Отмена особого смысла знаков в имени файла лишает такие конструкции как ‘`$`’, ‘`~`’ и ‘`:`’ их специального значения. См. Раздел 14.13 [Буквальные имена файлов], с. 134.

Отступ
: Отступ означает пустое пространство в начале строки. В большинстве языков программирования сложились традиции по использованию отступов, нацеленные на то, чтобы сделать понятной структуру программы, и в Emacs есть специальные команды для настройки отступов. См. Глава 20 [Отступы], с. 177.

Ошибка
: Ошибка появляется, когда команда Emacs не может выполниться при существующих условиях. Когда появляется ошибка, выполнение команды прекращается (если только команда не была запрограммирована как-то по-другому), и Emacs уведомляет об ошибке, печатая сообщение (см.). Все ранее набранное сбрасывается. После этого Emacs готов считывать другую команду редактирования.

Перестановка
: Перестановка двух единиц текста означает помещение каждого из них на место, ранее занимаемое другим. Существуют команды Emacs для перестановки двух смежных знаков, слов, s-выражений (см.) или строк. (см. Раздел 13.2 [Перестановка], с. 101).

Первичное выделение
: Первичное выделение — это одно конкретное выделение (см.) в X Windows; именно это выделение используется большинством приложений в X Windows для передачи текста другим приложениям.

  Команды уничтожения устанавливают первичное выделение, а команды восстановления используют первичное выделение, когда это уместно. См. Раздел 9.1 [Уничтожение], с. 69.

Первичный Rmail-файл
: Ваш первичный Rmail-файл — это файл с именем ‘`RMAIL`’ в вашем начальном каталоге, куда Rmail записывает получаемую вами почту, если вы не задали имя другого файла. См. Глава 27 [Rmail], с. 275.

Перевод строки
: Знаки Control-J в буфере ограничивают строки текста и называются поэтому переводами строки. См. Раздел 2.4 [Текстовые знаки], с. 31.

Переменная
: Переменная — это объект в Лиспе, который может хранить произвольное значение. Emacs использует некоторые переменные для своих внутренних целей и имеет другие (известные как ‘пользовательские параметры’ (см.)), для которых вы сами можете установить значения, чтобы управлять работой Emacs. Используемые в Emacs переменные, которыми вы, вероятно, заинтересуетесь, перечислены в указателе переменных в данном руководстве. См. Раздел 31.2 [Переменные], с. 343, для более подробной информации о переменных.

Перемещение текста
: Перемещение текста означает стирание его из одного места и вставку в другое. Обычно это делается при помощи уничтожения (см.) и последующего восстановления (см.). См. Раздел 9.1 [Уничтожение], с. 69.

Перерисовка экрана
: Перерисовка экрана — это процесс корректировки изображения на экране согласно изменениям, произведенным в редактируемом тексте. См. Глава 1 [Экран], с. 23.

Печатный знак ASCII
: Печатные знаки ASCII включают латинские буквы, арабские цифры, пробел и следующие знаки пунктуации: ‘`!@#$%^& *()_-+=|\~‘ {}[]:;"’ <>,.?/`’.

Подсветка
: Подсветка текста означает отображение его с другим цветом шрифта и/или фона, чтобы он выделялся из остального текста в буфере.

Подсказка
: Подсказка — это текст, напечатанный для запроса ввода от пользователя. Вывод подсказки называется запросом. Подсказки Emacs всегда появляются в эхо-области (см.). Один из видов запроса происходит, когда для считывания аргумента используется минибуфер (см. Глава 5 [Минибуфер], с. 45); эхо, появляющееся, когда вы останавливаетесь в середине набора многознаковой последовательности ключей — это также один из видов подсказки (см. Раздел 1.2 [Эхо-область], с. 24).

Подстановка строк
: Смотрите ‘глобальная подстановка’.

Поиск
: Поиск означает перемещение точки к следующему месту, где встретилось заданное регулярное выражение или заданная строка. См. Глава 12 [Поиск], с. 87.

Поиск слов
: Поиск слов — это поиск последовательности слов без учета пунктуации между ними. См. Раздел 12.3 [Поиск слов], с. 90.

Полоска меню
: Полоска меню — это линия наверху фрейма Emacs. Она содержит слова, на которых вы можете щелкнуть мышью, чтобы получить меню. Полоска меню поддерживается только в X Windows. См. Раздел 1.4 [Полоска меню], с. 26.

Полоска прокрутки
: Полоска прокрутки — это высокий узкий прямоугольник, находящийся с края окна. Вы можете использовать в нем команды мыши для прокрутки этого окна. Полоски прокрутки поддерживаются только в X Windows. См. Раздел 17.11 [Полоски прокрутки], с. 154.

Пользовательский параметр
: Пользовательский параметр — это переменная (см.), которая существует для того, чтобы вы могли настраивать Emacs, присваивая ей новое значение. См. Раздел 31.2 [Переменные], с. 343.

Последовательность ключей
: Последовательность ключей (или коротко, ключ) — это последовательность событий ввода (см.), которая имеет смысл как одна единица. Если последовательности ключей достаточно, чтобы определить одно действие, то это законченный ключ (см.), если нет, то это префиксный ключ (см.). См. Раздел 2.2 [Ключи], с. 30.

Почта
: Почта — это сообщения, посылаемые одним пользователем другому через компьютерную систему, чтобы получатель прочитал их, когда ему будет удобно. В Emacs есть команды для составления и отправки почты, а также для чтения и редактирования почты, которую вы получили. См. Глава 26 [Посылка почты], с. 267. См. Глава 27 [Rmail], с. 275, о том, как читать почту.

Предложения
: В Emacs есть команды для перемещения по предложениям и для их уничтожения. См. Раздел 21.2 [Предложения], с. 182.

Преобразование регистра
: Преобразование регистра означает перевод текста из верхнего регистра в нижний или наоборот. См. Раздел 21.6 [Регистр], с. 189, чтобы узнать о командах преобразования регистра.

Прерывание
: Прерывание означает выход из рекурсивного редактирования (см.). Для этого используются команды `C-]` и `M-x top-level`. См. Раздел 32.1 [Выход], с. 371.

Префикс заполнения
: Префикс заполнения — это цепочка знаков, которая ожидается в начале каждой строки, когда произведено заполнение. Она не рассматривается как часть заполняемого текста. См. Раздел 21.5 [Заполнение], с. 185.

Префиксный аргумент
: Смотрите ‘числовой аргумент’.

Префиксный ключ
: Префиксный ключ — это последовательность ключей (см.), чья единственная функция — это введение множества более длинных ключей. `C-x` — это пример префиксного ключа; любая двухзнаковая последовательность, начинающаяся с `C-x`, также является допустимой последовательностью ключей. См. Раздел 2.2 [Ключи], с. 30.

Привязка
: Последовательность ключей прибретает свое значение в Emacs при получении привязки, которая является командой (см.), то есть Лисп-функцией, которая запускается, когда пользователь вводит эту последовательность. См. Раздел 2.3 [Команды], с. 31. Настройка часто подразумевает перепривязку знака к другой командной функции. Привязки всех последовательностей ключей записываются в таблице ключей (см.). См. Раздел 31.4.1 [Таблицы ключей], с. 356.

Привязывать
: Привязать ключ значит изменить его привязку (см.). См. Раздел 31.4.5 [Перепривязка], с. 359.

Пробельные знаки
: Пробельные знаки — это любая последовательность форматирующих знаков (пробелов, табуляций, переводов строки и забоев).

Проведение мыши
: Проведение мыши — это вид события ввода, генерируемый, когда вы нажимаете кнопку мыши, перемещаете мышь и затем отпускаете кнопку. См. Раздел 31.4.10 [Кнопки мыши], с. 363.

Прокрутка
: Прокрутка означает сдвигание текста в окне Emacs таким образом, чтобы можно было увидеть другую часть буфера. См. Глава 11 [Изображение], с. 81.

Прямоугольник
: Прямоугольник состоит из текста в заданном диапазоне столбцов и в заданном диапазоне строк. Обычно вы задаете прямоугольник помещая точку в одном его углу, а метку в другом. См. Раздел 9.4 [Прямоугольники], с. 74.

Пустые строки
: Пустые строки — это строки, которые содержат только пробельные знаки. Emacs имеет несколько команд для работы с пустыми строками в буфере.

Путь поиска
: Путь поиска — это список имен каталогов, которые нужно использовать для поиска файлов, предназначенных для определенных целей. Например, переменная `load-path` содержит путь поиска для нахождения файлов с лисповскими библиотеками. См. Раздел 23.7 [Библиотеки Лиспа], с. 253.

Разбор
: Мы говорим, что определенные команды Emacs производят разбор слов или выражений в редактируемом тексте. В действительности, все, что они умеют — это находить другой конец слова или выражения. См. Раздел 31.6 [Синтаксис], с. 366.

Распечатка
: То же, что и твердая копия.

Расширение
: Расширение — это уничтожение любого ограничения (см.) в текущем буфере; это противоположность сужению (см.). См. Раздел 30.8 [Сужение], с. 335.

Регистры
: Регистрами называют именованные гнезда, в которые можно записывать прямоугольники или позиции в тексте или буфере для последующего использования. См. Глава 10 [Регистры], с. 77.

Регулярное выражение
: Регулярное выражение — это образец, который может соответствовать различным строкам или кускам текста; например, ‘`l[0-9]+`’ соответствует ‘`l`’, за которой следует одна или несколько цифр. См. Раздел 12.5 [Регулярные выражения], с. 91.

Режим Overwrite
: Режим Overwrite — второстепенный режим. Когда он включен, обыкновенные знаки текста заменяют существующий текст после точки, а не выталкивают его вправо. См. Раздел 31.1 [Второстепенные режимы], с. 341.

Резервная копия
: Резервная копия файла сохраняет содержимое файла, которое тот имел перед текущим сеансом редактирования. Emacs автоматически создает резервную копию файла, чтобы помочь вам проследить или отменить изменения, в которых вы позже раскаялись. См. Раздел 14.3.1 [Резерв], с. 110.

Рекурсивное редактирование
: Смотрите ‘уровень рекурсивного редактирования’.

Самовставляющийся знак
: Знак является самовставляющимся, если печать этого знака вставляет его в буфер. Обычные печатные и пробельные знаки являются в Emacs самовставляющимися, за исключением случаев в некоторых основных режимах.

Самодокументирование
: Самодокументирование — это средство Emacs, которое может сказать вам, что делает каждая команда, или дать список всех команд, относящихся к заданной вами теме. Самодокументирование запрашивается с помощью клавиши запроса справки, `C-h`. См. Глава 7 [Справка], с. 55.

Самосохранение
: Самосохранение — это когда Emacs автоматически сохраняет содержимое буфера в файле с особым именем. Таким образом, если из-за ошибки системы или пользователя буфер будет потерян, его информация не пропадает. См. Раздел 14.5 [Самосохранение], с. 114.

Синтаксическая таблица
: Синтаксическая таблица сообщает Emacs, какие знаки являются частью слова, какие уравновешивают друг как скобки и так далее. См. Раздел 31.6 [Синтаксис], с. 366.

Синтаксический разбор
: Смотрите ‘разбор’.

Система кодирования
: Система кодирования — это кодировка для представления текстовых знаков в файлах или потоках информации. Emacs умеет преобразовывать при чтении или записи текст во многих системах кодирования к нужной. См. Раздел 18.7 [Системы кодирования], с. 165.

Событие ввода
: Событие ввода представлет внутри Emacs одно действие, предпринятое пользователем за терминалом. События ввода включают набор знаков, нажатие функциональных клавиш, нажатие или отпускание кнопок мыши и переключение между фреймами Emacs. См. Раздел 2.1 [Пользовательский ввод], с. 29.

Сокращение
: Сокращение — это текстовая строка, которая раскрывается в другую текстовую строку, когда появляется в буфере. Например, вы могли бы определить несколько букв как сокращение для длинной фразы, которую вы хотите вставлять часто. См. Глава 24 [Сокращения], с. 257.

Сообщение
: Смотрите слово ‘почта’.

Сообщение об ошибке
: Сообщение об ошибке — это одиночная строка вывода, отображаемая Emacs, когда пользователь просит выполнить что-нибудь невозможное (например, уничтожить текст вперед, когда точка находится в конце буфера). Они появляются в эхо области и сопровождаются звуковым сигналом.

Сохранение
: Сохранение буфера означает копирование его текста в файл, к которому обращался этот буфер. Таким способом текст в файле действительно изменяется в процессе редактирования в Emacs. См. Раздел 14.3 [Сохранение], с. 108.

Список
: Список — это, приблизительно, текстовая строка, начинающаяся с открывающей круглой скобки и заканчивающаяся соответствующей закрывающей скобкой. В режиме C и в других режимах, отличных от режима Lisp, группы,

окруженные другими видами парных разделителей, соответствующих языку программирования, например фигурными скобками, также являются списками. См. Раздел 22.2 [Списки], с. 206.

Список меток
: Список меток используется для хранения нескольких последних позиций метки на случай, если вы захотите к ним вернуться. В каждом буфере свой собственный список меток; кроме того, есть единый глобальный список меток (см.). См. Раздел 8.5 [Список пометок], с. 66.

Список уничтожений
: Список уничтожений — это место, куда записывается весь недавно уничтоженный вами текст. Вы можете вновь вставить любой из уничтоженных фрагментов текста, пока он находится в списке. Это называется восстановлением (см.). См. Раздел 9.2 [Восстановление], с. 71.

Страница
: Страница — это единица текста, ограниченная знаками перевода страницы (ASCII control-L, код 014), идущими в начале строки. Предусмотрено несколько команд Emacs для перемещения через страницы и для действий с ними. См. Раздел 21.4 [Страницы], с. 184.

Строка продолжения
: Если строка текста длиннее, чем ширина окна, то при отображении она занимает больше одной экранной строки. Тогда мы говорим, что строка текста продолжается, а все используемые для этого экранные строки после первой называются строками продолжения. См. Глава 4 [Основы редактирования], с. 35.

Строка режима
: Строка режима — это строка внизу каждого окна (см.), которая выдает информацию о состоянии буфера, показанного в этом окне. См. Раздел 1.3 [Строка режима], с. 25.

Строка
: Строка — это один из видов лисповских объектов, который содержит последовательность знаков. Многие переменные Emacs предназначены для хранения строк в качестве значений. Синтаксис Лиспа для строк состоит из последовательности знаков этой строки с ‘"’ в начале и ‘"’ в конце. Когда знак ‘"’ является частью строки, его нужно записывать в виде ‘\"’, а знак ‘\’, являющийся частью строки, должен быть записан как ‘\\’. Все другие знаки, включая перевод строки, можно вставлять просто записывая их в строке; однако, также допускаются управляющие последовательности, принятые в Си, такие как ‘\n’ для перевода строки или ‘\241’, использующая восьмеричный код.

Сужение
: Сужение означает создание ограничений (см.), позволяющих редактирование текущего буфера только для части текста. Часть текста, оказавшаяся снаружи, недоступна для пользователя до тех пор, пока границы не будут вновь раздвинуты, но эта недоступная часть текста все еще находится там, и записывая файл, вы записываете его полностью. См. Раздел 30.8 [Сужение], с. 335.

Счетчик повторений
: Смотрите ‘числовой аргумент’.

Таблица ключей
: Таблица ключей — это структура данных, которая записывает привязки (см.) ключей к командам, которые они запускают. Например, глобальная таблица ключей привязывает знак `C-n` к командной функции `next-line`. См. Раздел 31.4.1 [Таблицы ключей], с. 356.

Таблица ключей второстепенного режима
: Таблица ключей, принадлежащая второстепенному режиму и активная, когда включен этот режим. Таблицы ключей второстепенных режимов имеют преимущество перед локальными в буфере таблицами ключей, так же, как локальные таблицы ключей имеют преимущество перед глобальной. См. Раздел 31.4.1 [Таблицы ключей], с. 356.

Таблица перевода клавиатуры
: Таблица перевода клавиатуры — это массив, который переводит коды знаков, приходящие с терминала, в коды знаков, которые будут составлять последовательности ключей. См. Раздел 31.5 [Перевод клавиатуры], с. 365.

Таблица тегов
: Таблица тегов — это файл, который служит в качестве указателя определений функций в одном или нескольких файлах. См. Раздел 22.13 [Теги], с. 224.

Твердая копия
: Твердая копия означает напечатанный вывод. В Emacs есть команды для создания твердой копии текста из буферов Emacs. См. Раздел 30.4 [Распечатка], с. 331.

Текст
: Есть два значения (см. Глава 21 [Текст], с. 181):

  - Данные, состоящие из последовательности знаков, в противоположность двоичным числам, изображениям, графическим командам, исполняемым программам и тому подобному. Содержимое буфера Emacs — это всегда текст в этом смысле.
  - Данные, написанные на естественном языке, в противоположность программам, или следующие стилистическими условностями естественного языка.

Текущая строка
: Строка, в которой находится точка (см. Раздел 1.1 [Точка], с. 23).

Текущее определение функции
: Определение функции (см.), в котором находится точка. Если точка находится между двумя определениями, то текущее определение — это то, которое следует после точки. См. Раздел 22.4 [Определения функций], с. 208.

Текущий абзац
: Абзац, в котором находится точка. Если она находится между абзацами, то текущий абзаца — это тот, который следует за точкой. См. Раздел 21.3 [Абзацы], с. 183.

Текущий буфер
: Текущий буфер в Emacs — это буфер, на который действуют большинство команд редактирования. Вы можете выбрать любой буфер Emacs в качестве текущего. См. Глава 15 [Буферы], с. 135.

Точка
: Точка — это место в буфере, в котором происходит вставка или удаление. Считается, что точка находится между двумя знаками, а не на каком-то одном из них. Курсор (см.) терминала показывает положение точки. См. Глава 4 [Основы], с. 35.

Удаление
: Удаление означает стирание текста без копирования его в список уничтожений (см.). Альтернатива этому — уничтожение (см.). См. Раздел 9.1 [Уничтожение], с. 69.

Удаление окон
: Удаление окна означает устранение его с экрана. Другие окна расширяются, чтобы использовать освободившееся пространство. Удаленное окно нельзя

вернуть, но реальный текст из-за этого не пропадает. См. Глава 16 [Окна], с. 141.

Удаление сообщений
: Удаление сообщения обозначает его пометку для исключения из вашего почтового файла. Оно может быть отменено до того момента, пока вы не вычеркните (см.) удаленные файлы из Rmail-файла. См. Раздел 27.4 [Удаление сообщений], с. 277.

Удаление файлов
: Удаление файла означает стирание его из файловой системы. См. Раздел 14.10 [Файлы Разное], с. 132.

Уничтожение
: Уничтожение означает стирание текста и запоминание его в списке уничтожений, чтобы вы могли восстановить (см.) его позднее. Некоторые другие системы называют это “вырезанием”. Большинство Emacs команд для стирания текста делают именно уничтожение, как противоположность удалению (см.). См. Раздел 9.1 [Уничтожение], с. 69.

Уничтожение заданий
: Уничтожение задания (такого как вызов Emacs) означает прекращение его существования. Любые данные внутри этого задания, если они не были записаны в файл, пропадают. См. Раздел 32.1 [Выход], с. 371.

Управление версиями
: Системы управления версиями отслеживают несколько версий файла с исходными кодами. Они предоставляют более мощную альтернативу хранению резервных копий файлов (см.). См. Раздел 14.7 [Управление версиями], с. 116.

Управляющий знак
: Управляющий знак — это знак, который вы вводите, прижав клавишу ⟨CTRL⟩. У некоторых управляющих знаков есть также собственные клавиши, так что вы можете ввести их, не используя ⟨CTRL⟩. Например, ⟨RET⟩, ⟨TAB⟩, ⟨ESC⟩ и ⟨DEL⟩ все являются управляющими знаками. См. Раздел 2.1 [Пользовательский ввод], с. 29.

Управляющий знак ASCII
: Управляющий знак ASCII — это Control-версия заглавной буквы или Control-версия одного из этих знаков: ‘`@[\]^_?`’.

Уровень рекурсивного редактирования
: Уровень рекурсивного редактирования — это состояние, в котором часть исполнения команды вызывает запрос к пользователю для редактирования какого-нибудь текста. Это может быть тот же самый текст, к которому применялась команда, а может быть и другой. Строка режима показывает наличие рекурсивного редактирования при помощи квадратных скобок (‘`[`’ или ‘`]`’). См. Раздел 30.12 [Рекурсивное редактирование], с. 338.

Усечение
: Усечение текстовых строк на дисплее означает отбрасывание в строке любого текста, который не вмещается в ширину окна. Смотрите также ‘строка продолжение’. См. Глава 4 [Основы редактирования], с. 35.

Файл сопровождения
: Файл, в который Emacs записывает все знаки, набранные пользователем на клавиатуре. Файлы сопровождения используются для регистрации ввода при отладке ошибок Emacs. Emacs не создает файл сопровождения, пока вы не потребуете этого. См. Раздел 32.3 [Ошибки в Emacs], с. 375.

Файл терминального протокола
: Файл терминального протокола содержит запись всех знаков, посылаемых Emacs на терминал. Он используется для отслеживания неполадок при перерисовке экрана в Emacs. Emacs не создает файл терминального протокола, если вы не потребуете этого. См. Раздел 32.3 [Ошибки в Emacs], с. 375.

Форматированный текст
: Форматированный текст — это текст, отображаемый с информацией форматирования во время редактирования. Информация форматирования включает шрифты, цвета и заданные поля. См. Раздел 21.11 [Форматированный текст], с. 198.

Фрейм
: Фрейм — это прямоугольный блок окон Emacs. При старте в Emacs есть один фрейм, но вы можете создать новые. Вы можете поделить каждый фрейм на окна Emacs (см.). Если вы пользуетесь X Windows, все фреймы могут быть видимы одновременно. См. Глава 17 [Фреймы], с. 147.

Функциональная клавиша
: Функциональная клавиша — это клавиша, которая посылает некий ввод, но не соответствует никакому знаку. См. Раздел 31.4.7 [Функциональные клавиши], с. 361.

Числовой аргумент
: Числовой аргумент — это число, заданное перед командой, чтобы изменить ее действие. Часто числовой аргумент служит в качестве счетчика повторов. См. Раздел 4.10 [Аргументы], с. 42.

Щелчок
: Щелчок — это вид события ввода, генерируемый, когда вы нажимаете кнопку мыши и отпускаете ее, не двигая мышь. См. Раздел 31.4.10 [Кнопки мыши], с. 363.

Электрик
: Мы говорим, что знак является электрик-знаком, если обычно он бывает самовставляющимся (см.), но текущий основной режим (см.) переопределяет для каких-то дополнительных действий. Например, несколько основных режимов для языков программирования определяют некоторые знаки-разделители для создания в этой строке отступа или вставки еще одного или более перевода строки, помимо вставки самого этого знака.

Эхо
: Эхо — это подтверждение получения команд с помощью их показа (в эхо-области). Emacs никогда не повторяет однознаковые последовательности ключей; более длинные последовательности ключей повторяются, только если вы сделаете паузу, пока набираете их.

Эхо-область
: Эхо-область — это нижняя строка экрана, используемая для вывода эхо аргументов команд, для задавания вопросов и для печати кратких сообщений (включая сообщения об ошибках). Эти сообщения сохраняются в буфере ‘`*Messages*`’, чтобы вы потом могли их просмотреть. См. Раздел 1.2 [Эхо-область], с. 24.

Языковая среда
: Ваш выбор языковой среды определяет принимаемые по умолчанию метод ввода (см.) и систему кодирования (см.). См. Раздел 18.3 [Языковые среды], с. 162. Эти значения по умолчанию важны, если вы редактируете не-ASCII текст. (см. Глава 18 [MULE], с. 161).

Alt
: Alt — это название бита-модификатора, который может присутствовать в знаке, введенном с клавиатуры. Чтобы сделать знак Alt-знаком, наберите его, прижав клавишу ⟨ALT⟩. Таким знакам даются имена, начинающиеся с `Alt-`

(для краткости обычно пишут `A-`). (Обратите внимание, на многих терминалах есть клавиша, помеченная как ⟨ALT⟩, которая на самом деле работает как ⟨META⟩.) См. Раздел 2.1 [Пользовательский ввод], с. 29.

ASCII-знак
: ASCII-знак — это либо управляющий знак ASCII, либо печатный знак ASCII. См. Раздел 2.1 [Пользовательский ввод], с. 29.

Auto Fill
: Режим Auto Fill — это второстепенный режим, в котором вставляемый вами текст автоматически разбивается на строки фиксированной длины. См. Раздел 21.5 [Заполнение], с. 185.

`C-`
: `C-` в имени знака — это сокращение для Control. См. Раздел 2.1 [Пользовательский ввод], с. 29.

`C-M-`
: `C-M-` в имени знака — это сокращение для Control-Meta. См. Раздел 2.1 [Пользовательский ввод], с. 29.

Copyleft
: Copyleft — это уведомление, дающее обществу официальное разрешение на повторное распространение программы или иного произведения искусства. Copylefts используются программистами левых партий для поддержки свободы и сотрудничества, так же как copyrights используются программистами правых партий для повышения власти над другими людьми.

  Конкретная форма copyleft, используемая проектом GNU, называется Универсальной Общественной Лицензией GNU. См. [Копирование], с. 13.

⟨DEL⟩
: ⟨DEL⟩ — это знак, запускающий команду, которая удаляет один знак текста. См. Глава 4 [Основы редактирования], с. 35.

Dired
: Dired — это средство Emacs, которое показывает содержимое каталога файлов и позволяет вам “редактировать каталог”, выполняя действия над файлами в этом каталоге. См. Глава 28 [Dired], с. 291.

⟨ESC⟩
: ⟨ESC⟩ — это знак, используемый как префикс для набора Meta-знаков на клавиатурах, где нет клавиши ⟨META⟩. В отличие от клавиши ⟨META⟩ (которая подобно ⟨SHIFT⟩ остается нажатой, пока набирается другой знак), ⟨ESC⟩ нажимается так же, как клавиша с буквой, и относится к знаку, набранному следом за ней.

⟨HELP⟩
: ⟨HELP⟩ — это название для `C-h` или ⟨F1⟩ в Emacs. Вы можете набрать ⟨HELP⟩ в любое время, чтобы спросить, какой у вас есть выбор, или узнать, что делает та или иная команда. См. Глава 7 [Справка], с. 55.

Hyper
: Hyper — это имя бита-модификатора, который может иметь вводимый знак. Чтобы сделать знак Hyper-знаком, наберите его, прижав клавишу ⟨HYPER⟩. Таким знакам дают имена, начинающиеся с `Hyper-` (обычно записывается как `H-` для краткости). См. Раздел 2.1 [Пользовательский ввод], с. 29.

`M-`
: `M-` в имени знака — это сокращение от ⟨META⟩, одной из клавиш-модификаторов, которая может сопровождать любой знак. См. Раздел 2.1 [Пользовательский ввод], с. 29.

`M-C-`
: `M-C-` в имени знака — это сокращение для Control-Meta; это означает то же самое, что и `C-M-`. Если у вашего терминала нет реальной клавиши ⟨META⟩, то вы набираете Control-Meta-знак при помощи ⟨ESC⟩ и соответствующего знака с Control. См. Раздел 2.1 [Пользовательский ввод], с. 29.

`M-x`
: `M-x` — это последовательность ключей, которая используется для вызова команд Emacs по имени. Так вызываются команды, которые не привязаны к ключам. См. Глава 6 [M-x], с. 53.

Meta
: Meta — это имя бита-модификатора, который может иметь командный знак. Он присутствует в знаке, если тот набирается вместе с одновременным нажатием клавиши ⟨META⟩. Таким знакам даются имена, начинающиеся с `Meta-` (для краткости обычно пишут `M-`). Например, `M-<` набирается при помощи одновременного нажатия ⟨META⟩ и набора `<` (на большинстве терминалов `<` набирается путем нажатия ⟨SHIFT⟩ и `,`). См. Раздел 2.1 [Пользовательский ввод], с. 29.

Meta-знак
: Meta-знак — это знак, чей знаковый код включает Meta-бит.

MULE
: Под MULE понимают средства Emacs для редактирования не-ASCII-текста с использованием многобайтных знаков (см.). См. Глава 18 [MULE], с. 161.

Regexp
: Смотрите ‘регулярное выражение’.

⟨RET⟩
: ⟨RET⟩ — это знак, который запускает команду, вставляющую в текст перевод строки. Он также используется для завершения большинства аргументов, считываемых из минибуфера (см.). См. Раздел 2.1 [Пользовательский ввод], с. 29.

Rmail-файл
: Rmail-файл — это файл, содержащий текст в особом формате, используемый программой Rmail для хранения почты. См. Глава 27 [Rmail], с. 275.

S-выражение
: S-выражение (или коротко, sexp) — это основная синтаксическая единица Лиспа в ее текстуальной форме: или список, или атом. Многие команды Emacs работают с s-выражениями. Термин ‘s-выражение’ обобщается для остальных языков помимо Лиспа как синтаксически распознаваемое выражение. См. Раздел 22.2 [Списки], с. 206.

Super
: Super — это название бита-модификатора, который может присутствовать в знаке, введенном с клавиатуры. Чтобы сделать знак Super-знаком, наберите его, прижав клавишу ⟨SUPER⟩. Таким знакам даются имена, начинающиеся с `Super-` (для краткости обычно пишут `s-`). См. Раздел 2.1 [Пользовательский ввод], с. 29.

WYSIWYG
: WYSIWYG означает ‘What you see is what you get’.[2] Emacs обычно обеспечивает редактирование WYSIWYG для файлов, состоящих из знаков; в режиме Enriched (см. Раздел 21.11 [Форматированный текст], с. 198) он предоставляет редактирование WYSIWYG для файлов, включающих информацию форматирования.

[2] Что вы видите, то и получаете. *(Прим. переводчика)*

# Указатель ключей (клавиш)

## default



## 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

# Указатель команд и функций

## default



## 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

# Указатель переменных

## A



## B



## C



## D

## E



## F



## G



## H

## I



## J



## K



## L



## M



## N



## O

## P



## R



## S

## T



## U



## V



## W



## X

# Предметный указатель

## default



## A



## B



## C



## D



## E



## F



## G



## H



## I

## J



## K



## L



## M



## N



## O



## P



## Q



## R



## S



## T

## Г



## Д



## Е



## Ж



## З



## И



## К

## Л



## М



## Н

## О



## П

## P

## С



## Т



## У

## Ф



## Х



## Ц



## Ч



## Ш



## Щ



## Э



## Ю



## Я

# Краткое содержание

# Оглавление