*Работа с текстом на максимальной скорости и мощности*

**7 издание**
7 новых глав о Vim

# *Изучаем* vi и Vim *редакторы*

*Арнольд Роббинс, Элберт Ханна и Линда Лэмб*

# Learning the vi and Vim Editors

Seventh Edition

*Arnold Robbins, Elbert Hannah and Linda Lamb*

O'REILLY®

# Изучаем редакторы vi и Vim

Седьмое издание

*Арнольд Роббинс, Элберт Ханна*
*и Линда Лэмб*

*Санкт-Петербург — Москва*
*2013*

Арнольд Роббинс, Элберт Ханна и Линда Лэмб

# Изучаем редакторы vi и Vim, 7-е издание

Перевод И. Аввакумова

| | |
|---|---|
| Главный редактор | *А. Галунов* |
| Зав. редакцией | *Н. Макарова* |
| Научный редактор | *В. Синицын* |
| Редактор | *Ю. Бочина* |
| Корректор | *С. Беляева* |
| Верстка | *Д. Орлова* |

*Роббинс А., Ханна Э., Лэмб Л.*

Изучаем редакторы vi и Vim, 7-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2013. – 512 с., ил.

ISBN 978-5-93286-200-1

На протяжении 30 лет vi оставался стандартом для UNIX и Linux, а эта книга была главным пособием по vi. Однако сейчас UNIX уже не тот, что был 30 лет назад, и книга тоже не стоит на месте. Седьмое издание существенно расширено и включает подробную информацию о Vim – самом популярном клоне vi. Доступный стиль изложения сделал эту книгу классикой. Она незаменима, поскольку знание vi или Vim – обязательное условие, если вы работаете в Linux или UNIX.

Вы познакомитесь как с основами, так и с продвинутыми средствами, такими как интерактивные макросы и скрипты, расширяющие возможности редактора. Вы научитесь быстро перемещаться в vi, использовать буферы, применять глобальную функцию поиска и замены vi, настраивать vi и запускать команды UNIX, использовать расширенные текстовые объекты Vim и мощные регулярные выражения, редактировать в нескольких окнах и писать скрипты в Vim, использовать все возможности графической версии Vim (gvim), применять такие усовершенствования Vim, как подсветка синтаксиса и расширенные теги. Помимо Vim рассматриваются и другие клоны vi: nvi, elvis и vile.

**ISBN 978–5–93286–200–1**
**ISBN 978–0–596–52983–3 (англ)**

© Издательство Символ-Плюс, 2013
Authorized Russian translation of the English edition of Learning the vi and Vim Editors, Seventh Edition ISBN 9780596529833 © 2008 O'Reilly Media, Inc. All rights reserved. This translation is published and sold by permission of O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

Все права на данное издание защищены Законодательством РФ, включая право на полное или частичное воспроизведение в любой форме. Все товарные знаки или зарегистрированные товарные знаки, упоминаемые в настоящем издании, являются собственностью соответствующих фирм.

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 380-5007, www.symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Подписано в печать 12.12.2012. Формат 70×100 1/16.
Печать офсетная. Объем 32 печ. л.

*Посвящается моей жене Мириам*
*за любовь, терпение и поддержку.*

Арнольд Роббинс
(шестое и седьмое издания)

# Оглавление

**Предисловие** . . . . . . . . . . 10

**I. Базовый и продвинутый vi** . . . . . . . . . . 19

**1. Текстовый редактор vi** . . . . . . . . . . 21

Краткая историческая справка . . . . . . . . . . 24
Открытие и закрытие файлов . . . . . . . . . . 25
Выход без сохранения правок . . . . . . . . . . 29

**2. Простое редактирование** . . . . . . . . . . 32

Команды vi . . . . . . . . . . 32
Перемещение курсора . . . . . . . . . . 33
Простая правка текста . . . . . . . . . . 37
Другие способы вставки текста . . . . . . . . . . 50
Объединение двух строк с помощью J . . . . . . . . . . 52
Обзор основных команд vi . . . . . . . . . . 53

**3. Быстрое перемещение** . . . . . . . . . . 55

Перемещение по экранам . . . . . . . . . . 55
Перемещение по текстовым блокам . . . . . . . . . . 59
Перемещение по результатам поиска . . . . . . . . . . 60
Перемещение по номеру строки . . . . . . . . . . 64
Обзор команд перемещения курсора в vi . . . . . . . . . . 65

**4. За рамками основ** . . . . . . . . . . 67

Другие сочетания команд . . . . . . . . . . 67
Варианты запуска vi . . . . . . . . . . 68
Использование буферов . . . . . . . . . . 71
Отметка места . . . . . . . . . . 73
Другие продвинутые команды редактирования . . . . . . . . . . 74
Обзор команд vi для работы с буфером и метками . . . . . . . . . . 74

**5. Введение в редактор ex** . . . . . . . . . . 75

Команды ex . . . . . . . . . . 76
Редактирование в ex . . . . . . . . . . 78
Сохранение и выход . . . . . . . . . . 84
Копирование одного файла в другой . . . . . . . . . . 86
Редактирование нескольких файлов . . . . . . . . . . 87

**6. Глобальная замена** . . . . . 91
Подтверждаем замены . . . . . 92
Замена, зависящая от контекста . . . . . 93
Поиск по шаблону . . . . . 94
Примеры использования шаблонов . . . . . 102
Финальный взгляд на шаблоны . . . . . 110

**7. Продвинутое редактирование** . . . . . 116
Настройка vi . . . . . 117
Вызов команд UNIX . . . . . 121
Сохранение команд . . . . . 124
Использование скриптов ex . . . . . 137
Редактирование исходного кода программы . . . . . 143

**8. Представляем клоны vi** . . . . . 149
Знакомьтесь: Даррелл, Даррелл и Даррелл . . . . . 149
Многооконное редактирование . . . . . 151
Графические интерфейсы . . . . . 152
Расширенные регулярные выражения . . . . . 152
Улучшенные теги . . . . . 154
Улучшенные возможности . . . . . 160
Помощь программисту . . . . . 165
Итоги: сравним редакторы . . . . . 167
Ничто не сравнится с оригиналом . . . . . 167
Перспектива . . . . . 168

**II. Vim** . . . . . 169

**9. Vim (vi Improved): введение** . . . . . 171
Обзор . . . . . 172
Где взять Vim . . . . . 177
Как установить Vim в UNIX и GNU/Linux . . . . . 178
Установка Vim в окружении Windows . . . . . 183
Установка Vim в окружении Macintosh . . . . . 183
Другие операционные системы . . . . . 184
Помощь и упрощения для новичков . . . . . 185
Итог . . . . . 185

**10. Главные улучшения Vim по сравнению с vi** . . . . . 187
Встроенная справка . . . . . 187
Варианты запуска и инициализации . . . . . 189
Новые команды перемещения . . . . . 196
Расширенные регулярные выражения . . . . . 198
Сборка исполняемого файла под конкретные задачи . . . . . 201

**11. Многооконность в Vim** . . . . . 202
Инициализация многооконного сеанса . . . . . 203
Открытие окон . . . . . 206
Перемещение по окнам (движение курсора между окнами) . . . . . 209
Перемещение окон . . . . . 211

Изменение размера окна . . . 213
Буферы и их взаимодействие с окнами . . . 217
Теги и окна . . . 221
Редактирование с вкладками . . . 222
Закрытие и выход из окон . . . 223
Итог . . . 224

**12. Скрипты Vim** . . . 225

Какой ваш любимый цвет? . . . 225
Динамическая конфигурация типов файлов при помощи скриптов . . . 236
Дополнительные соображения, касающиеся скриптов Vim . . . 245
Ресурсы . . . 250

**13. Графический Vim (gvim)** . . . 251

Общее введение в gvim . . . 252
Настройка полос прокрутки, меню и панелей инструментов . . . 257
gvim в Microsoft Windows . . . 269
gvim в X Window System . . . 269
Опции GUI и обзор команд . . . 269

**14. Улучшения Vim для программистов** . . . 272

Свертка и контуры (режим контуров) . . . 273
Автоматические и умные отступы . . . 284
Ключевые слова и завершение слов по словарю . . . 293
Стеки тегов . . . 302
Подсветка синтаксиса . . . 305
Компиляция и поиск ошибок в Vim . . . 314
Заключительные соображения о написании программ . . . 319

**15. Другие полезности в Vim** . . . 320

Редактирование двоичных файлов . . . 320
Диграфы: не-ASCII символы . . . 322
Редактирование файлов из других мест . . . 324
Переход и смена каталогов . . . 326
Резервные копии в Vim . . . 328
Создание HTML из текста . . . 329
В чем разница? . . . 330
Отмена отмен . . . 332
На чем я остановился? . . . 333
На какой я строке? . . . 336
Сокращения команд и опций Vim . . . 338
Несколько мелочей (не обязательно для Vim) . . . 339
Другие ресурсы . . . 340

**III. Другие клоны vi** . . . 341

**16. nvi: новый vi** . . . 343

Автор и история . . . 343

Важные аргументы командной строки . . . 344
Онлайн-справка и другая документация . . . 345
Инициализация . . . 346
Многооконное редактирование . . . 346
Графические интерфейсы . . . 348
Расширенные регулярные выражения . . . 348
Улучшения в редактировании . . . 349
Помощь программисту . . . 352
Интересные функции . . . 352
Исходный код и поддерживаемые операционные системы . . . 353

**17. elvis** . . . 354

Автор и история . . . 354
Важные аргументы командной строки . . . 355
Онлайн-справка и другая документация . . . 356
Инициализация . . . 356
Многооконное редактирование . . . 358
Графические интерфейсы . . . 360
Расширенные регулярные выражения . . . 366
Улучшенные возможности редактирования . . . 366
Помощь программисту . . . 371
Интересные особенности . . . 374
Будущее elvis . . . 380
Исходный код и другие операционные системы . . . 380

**18. vile: vi Like Emacs (vi как Emacs)** . . . 382

Авторы и история . . . 382
Важные аргументы командной строки . . . 383
Онлайн-справка и другая документация . . . 384
Инициализация . . . 386
Многооконное редактирование . . . 387
Графические интерфейсы . . . 389
Расширенные регулярные выражения . . . 398
Улучшенные возможности редактирования . . . 400
Помощь программисту . . . 407
Интересные особенности . . . 410
Исходный код и поддерживаемые операционные системы . . . 417

**IV. Приложения** . . . 419

**A. Редакторы vi, ex и Vim** . . . 421

**B. Установка опций** . . . 458

**C. Возможные проблемы** . . . 479

**D. vi и Интернет** . . . 483

**Алфавитный указатель** . . . 495

# Предисловие

Редактирование текстов – одна из наиболее востребованных задач в любой компьютерной системе, а vi – один из наиболее полезных стандартных текстовых редакторов. С помощью vi можно создавать новые текстовые файлы или редактировать имеющиеся.

Как и многие классические программы, разработанные во времена становления UNIX, vi имеет репутацию сложной в управлении программы. Создавая улучшенный клон vi под названием Vim (от «vi improved»), Брам Моленар (Bram Moolenaar) сделал многое, чтобы устранить причины такого впечатления. Vim содержит многочисленные усовершенствования, визуальные подсказки и справочную систему. Он стал, вероятно, самой популярной версией vi, поэтому в седьмом издании этой книги ему посвящено семь новых глав в части II «Vim». Однако существует множество других клонов vi, три из которых мы рассмотрим в части III «Другие клоны vi».

## План книги

Книга разбита на 4 части и состоит из 18 глав и 4 приложений.

Часть I «Базовый и продвинутый vi» поможет быстро начать работу с vi, а также получить углубленные навыки, позволяющие использовать его более эффективно.

В главе 1 «Текстовый редактор vi» описываются некоторые простые команды vi, с которых можно начать знакомство с программой. Попрактикуйтесь в них, пока не освоите достаточно хорошо. Глава 2 «Простое редактирование» познакомит с некоторыми элементарными инструментами редактирования.

Однако функциональные возможности vi выходят далеко за рамки обычной обработки текста. Большое разнообразие команд и опций позволит сократить существенную часть рутинной работы. В главе 3 «Быстрое перемещение» и главе 4 «За рамками основ» уделяется внимание более простым способам выполнения задач. При первом чтении вы получите, по крайней мере, представление о возможностях vi и о том, какие команды можно приспособить под ваши нужды. Впоследствии можно вернуться к этим главам для более детального изучения.

Глава 5 «Введение в редактор ex», глава 6 «Глобальная замена» и глава 7 «Продвинутое редактирование» посвящены средствам, позволяющим переложить часть бремени редактирования на плечи компьютера. Вы познакомитесь со строковым редактором ex, лежащим в основе vi, и узнаете, как из vi обращаться к командам ex.

Глава 8 «Представляем клоны vi» знакомит с расширениями, доступными в четырех клонах vi. Здесь описываются многооконное редактирование, графические интерфейсы, расширенные регулярные выражения, функции, облегчающие редактирование, и некоторые другие особенности, тем самым показывая план оставшейся части книги. Кроме того, в этой главе есть ссылка на исходный код первоначального vi, который может быть легко скомпилирован на современных UNIX-системах (включая GNU/Linux).

Часть II «Vim» описывает Vim – наиболее популярный на сегодняшний день клон vi.

В главе 9 «Vim (vi Improved): введение» дается общая информация о Vim, в том числе, где взять бинарные версии для наиболее популярных операционных систем и каковы различные варианты применения Vim.

В главе 10 «Главные улучшения Vim по сравнению с vi» описываются наиболее существенные улучшения в Vim по сравнению с vi, такие как встроенная справка, управление инициализацией, дополнительные команды перемещения и расширенные регулярные выражения.

Глава 11 «Многооконность в Vim» уделяет внимание многооконному редактированию, которое, возможно, является наиболее значимым дополнением к стандартному vi. В главе рассматриваются все подробности создания и использования нескольких окон.

В главе 12 «Скрипты Vim» рассматривается язык команд Vim, который позволит вам писать скрипты, чтобы приспособить Vim под ваши нужды. Простота использования Vim «из коробки» во многом объясняется огромным количеством скриптов, написанных другими пользователями и включенных в дистрибутив Vim.

В главе 13 «Графический Vim (gvim)» рассматривается Vim в современных графических окружениях, например тех, которые являются стандартными на современных коммерческих UNIX-системах, в GNU/Linux и других UNIX-системах, а также в MS Windows.

Глава 14 «Улучшения Vim для программистов» сосредоточена на использовании Vim в качестве редактора для программистов, оставляя за рамками его возможности обычного редактирования текста. Особенно ценными являются функции сворачивания кода и редактирования планов-схем, умные отступы, подсветка синтаксиса и ускорение цикла «редактирование-компиляция-отладка».

Глава 15 «Другие полезности в Vim» является отчасти собирательной, так как в ней охватывается множество интересных вопросов, не вошедших в предыдущие главы.

Часть III «Другие клоны vi» посвящена трем другим популярным клонам vi: nvi, elvis и vile.

Глава 16 «nvi: новый vi», глава 17 «elvis» и глава 18 «vile: vi как Emacs» охватывают различные клоны vi: nvi, elvis и vile. В главах обсуждается, как использовать их расширения, и описываются особенности каждого из них.

Часть IV «Приложения» содержит полезные справочные материалы.

В приложении A «Редакторы vi, ex и Vim» перечисляются все команды vi и ex, отсортированные по функциям. Кроме того, приводится список команд ex в алфавитном порядке, а также некоторые команды vi и ex из Vim.

Приложение B «Установка опций» содержит список опций команды set для vi и всех четырех его клонов.

В приложении C «Возможные проблемы» обсуждаются возможные проблемы при работе с vi и его клонами, а также способы их устранения.

В приложении D «vi и Интернет» рассказывается о месте, которое занимает vi в более широкой культуре UNIX и Интернета.

# Способ представления материала

Наша задача – дать хороший обзор материала, который поможет новичкам изучить vi. Освоение нового редактора, особенно редактора со всеми возможностями vi, может показаться непреодолимой задачей. Мы сделали попытку представить основные концепции и команды в логичной и удобочитаемой форме.

После изложения общих основ vi, применимых везде, мы переходим к более глубокому рассмотрению Vim. Картину завершает обзор nvi, elvis и vile. Последующие разделы описывают условные обозначения, используемые в этой книге.

## Обсуждение команд vi

Здесь вы найдете краткое описание основной идеи, предшествующее узкоспециализированным разделам. Затем приводятся примеры применения этой команды в каждом конкретном случае наряду с ее описанием и синтаксисом использования.

## Условные обозначения

В описании синтаксиса и в примерах данные для ввода набраны шрифтом MonoCondensed. То же касается названий команд, имен файлов и опций. Переменные (то есть то, что не будет вводиться буквально, а будет заменяться при вводе команды на нужное значение) набраны *курсивным MonoCondensed*. Квадратные скобки означают, что переменная является необязательной. Например, в строке с синтаксисом:

```
vi [filename]
```

*filename* будет заменено на реальное имя файла. Скобки говорят о том, что команда vi может вызываться без указания имени файла. Сами скобки вводить не надо.

Некоторые примеры показывают результат работы команд, вводимых в командной строке UNIX. В таких примерах то, что вы реально вводите, набрано шрифтом **MonoCondensed Bold**, чтобы отличать это от отклика системы. Например:

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

В примерах кода *курсив* обозначает комментарий, который вводить не надо. В основном тексте *курсивом* выделены специальные термины либо то, на что следует обратить внимание.

Следуя общепринятым соглашениям по документации UNIX, ссылки вида *printf(3)* указывают на электронное справочное руководство (которое можно получить посредством команды man). Этот пример ссылается на страницу функции printf() в разделе 3 этого руководства (в большинстве систем нужно ввести man 3 printf, чтобы увидеть ее).

# Клавиши

На протяжении всей книги вы встретите таблицы, содержащие команды vi и результаты их работы:

| Клавиши | Результаты |
|---|---|
| ZZ | "practice" (New file) 6 lines, 320 characters<br><br>Введите команду выхода с сохранением — ZZ. Ваш файл будет сохранен как обычный файл UNIX. |

В этом примере команда ZZ приведена в левом столбце. В рамке справа содержится строка (или несколько строк) экрана, показывающая результат выполнения команды. Положение курсора показано инверсией фона и цвета символов. В этом случае, поскольку ZZ сохраняет файл и выходит из программы, после записи файла вы увидите строку состояния; положение курсора не показано. Под рамкой расположено объяснение команды и ее результата.

Иногда к командам vi обращаются при помощи одновременного нажатия клавиши CTRL с другой клавишей. В основном тексте такая комбинация клавиш обычно записывается так: CTRL-G. В примерах кода в таких случаях перед названием клавиши ставят знак вставки (^), например ^G означает, что при нажатии на G нужно удерживать нажатой клавишу CTRL.

## Возможные проблемы

В тех разделах, где у вас могут возникнуть затруднения, содержится перечень возможных ошибок при выполнении тех или иных задач. Вы можете просмотреть эти ошибки и вернуться к ним, когда столкнетесь с подобной проблемой на практике. Чтобы упростить доступ к перечню возможных ошибок, они приведены также в приложении C.

# Что нужно знать

Мы полагаем, что вы уже прочли «Learning the Unix Operating System» (O'Reilly) или какое-нибудь другое введение в UNIX. Вы должны знать, как:

- осуществлять вход в систему и выход из нее;
- вводить команды UNIX;
- менять каталоги;
- выводить список файлов в каталоге;
- создавать, копировать и удалять файлы.

Знакомство с `grep` (global search program, программа глобального поиска) и символами подстановки также будет полезным.

# Замечания и вопросы

Свои замечания и вопросы по этой книге отправляйте, пожалуйста, издателю:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

Технические вопросы и замечания о книге присылайте на электронный адрес:

*bookquestions@oreilly.com*

Веб-сайт этой книги содержит примеры, список ошибок и планы будущих изданий. Страница располагается по адресу:

*http://www.oreilly.com/catalog/9780596529833*

Для получения дополнительной информации о наших книгах, конференциях, программных продуктах, информационных центрах и о сети O'Reilly Network посетите наш веб-сайт:

*http://www.oreilly.com*

## Safari® Books Online

Если вы видите значок Safari® Books Online на вашей любимой технической книге, это означает, что книга доступна онлайн посредством O'Reilly Network Safari Bookshelf.

Safari предлагает лучшее решение, нежели электронные книги. Это виртуальная библиотека, где вы можете легко найти любую из тысяч технических книг, копировать и вставлять фрагменты кода, скачивать отдельные главы и быстро находить ответы, когда вам нужна самая точная и актуальная информация. Попробуйте это бесплатно на *http://safari.oreilly.com*.

## О предыдущих изданиях

В пятом издании книги под названием «Learning the vi Editor» команды редактора `ex` рассматривались более подробно. В главах 5, 6 и 7 с помощью большого количества примеров разъяснялись сложные функции `ex` и `vi` в таких темах, как синтаксис регулярных выражений, глобальные замены, файлы `.exrc`, сокращения слов, отображение клавиш и скрипты редактирования. Несколько примеров было взято из журнала «Unix World». Вальтер Зинц (Walter Zintz) написал учебник о `vi` из двух частей[1], рассказывающий о нескольких неизвестных нам вещах и содержащий множество грамотных примеров, которые иллюстрируют уже рассмотренные нами функции. Рэй Шварц (Ray Swartz) в одной из своих заметок также поделился полезными советами[2]. Мы благодарны им за идеи, изложенные в этих материалах.

Шестое издание «Learning the vi Editor» содержало ознакомительный обзор четырех доступных «клонов», то есть редакторов со схожими принципами работы. Многие из них содержали улучшения по сравнению с `vi`. Следовательно, можно говорить о существовании «семейства» редакторов `vi`. Издание в равной степени уделяло внимание `nvi`, `Vim`, `elvis` и `vile` с целью познакомить читателя с этими клонами.

Также в шестом издании было добавлено следующее:

- В основной текст внесены многочисленные исправления и дополнения.
- В конце каждой главы приведена сводка соответствующих команд.

---

1 Две статьи Вальтера Зинца: «`vi` Tips for Power Users», *Unix World*, апрель 1990 и «Using `vi` to Automate Complex Edits», *Unix World*, май 1990. (В приложении D указаны веб-адреса этих статей.)

2 «Answers to Unix», *Unix World*, август 1990.

- Новые главы, посвященные каждому из клонов vi, функции и/или расширения, общие для двух или более клонов, и многооконное редактирование.
- Главы, рассказывающие немного об истории, целях, уникальных особенностях, способах установки каждого из клонов vi.
- Новое приложение, где говорится о месте vi в более широкой культуре UNIX и Интернета.

## Предисловие к седьмому изданию

Седьмое издание «Learning the vi and Vim Editors» содержит все лучшее от шестого. Время показало, что именно Vim является самым популярным клоном vi, так что в этом издании обзор данного редактора существенно расширен (ему даже отведено место в названии книги). Но чтобы книга оставалась полезной для как можно большего числа читателей, мы оставили и обновили материалы о nvi, elvis и vile.

### Что нового

В этом издании появились следующие новые материалы:

- Внесены исправления в основном тексте.
- Семь новых глав, в которых всесторонне рассматривается Vim.
- Материал про современное состояние nvi, elvis и vile.
- Два приложения из предыдущего издания, содержащие справку по ex и vi, были объединены в одно, которое теперь содержит еще и дополнительные материалы по Vim.
- Обновлены другие приложения.

### Версии

При тестировании различных функций vi использовались следующие программы:

- Версия vi для Solaris как наиболее близкая к версии vi в UNIX.
- Версия программы nvi 1.79 Кейта Бостича (Keith Bostic).
- Версия программы elvis 2.2 Стива Киркендалля (Steve Kirkendall).
- Версия Vim 7.1 Брама Моленара (Bram Moolenaar).
- Версия vile 9.6 Кевина Бейттнера (Kevin Buettner), Тома Дики (Tom Dickey) и Пола Фокса (Paul Fox).

## Благодарности для шестого издания

В первую очередь благодарю свою жену Мириам за заботу о детях, пока я работал над книгой, в особенности во время «волшебных часов» непосредственно перед обедом. Я должен ей огромное количество тишины и мороженого.

Пол Манно (Paul Manno) из Технического колледжа компьютерных технологий Джорджии (Georgia Tech College of Computing) оказал неоценимую помощь в усмирении моих программ печати. Лен Мюлнер (Len Muellner) и Эрик Рэй (Erik Ray) из O'Reilly & Associates помогали с программами для SGML. Макрос `vi`, написанный Джерри Пиком (Jerry Peek), оказался бесценным.

Хотя при подготовке нового и исправления старого материала использовались все упомянутые программы, большая часть редактирования осуществлялась в Vim версий 4.5 и 5.0 под GNU/Linux (Red Hat 4.2).

Я благодарен Кейту Бостичу (Keith Bostic), Стиву Киркендаллю (Steve Kirkendall), Браму Моленару (Bram Moolenaar), Полу Фоксу (Paul Fox), Тому Дики (Tom Dickey) и Кевину Бейттнеру (Kevin Buettner), проверявшим книгу и снабдившим меня важными материалами для глав с 8 по 12 (номера этих глав соответствуют шестому изданию).

Без электричества, вырабатываемого энергетической компанией, работать на компьютере невозможно. Однако когда электричество есть в розетке, вы перестаете думать о нем. Точно так же и при написании книги – без редактора у вас ничего не получится, однако когда он делает свою работу, о нем легко забыть. Гиги Эстабрук (Gigi Estabrook) из O'Reilly – это просто жемчужина. Работать с ней одно удовольствие. Я высоко ценю все, что она делала и продолжает делать для меня.

И наконец, много благодарностей команде O'Reilly & Associates.

*Арнольд Роббинс (Arnold Robbins)*
*Ra'anana, Израиль, июнь 1998*

## Благодарности для седьмого издания

И снова Арнольд благодарит свою жену Мириам за любовь и поддержку. Размер долга в виде тишины и мороженого продолжает расти. Кроме того, он благодарен Дж.Д. «Илиаду» Фрейзеру (J.D. «Illiad» Frazer) за прекрасные комиксы *User Friendly*[1].

Элберт хотел бы поблагодарить Анну, Келли, Бобби и своих родителей за проявленный интерес к его работе в нелегкое время. Их энтузиазм был заразительным и бесценным.

Благодарим Кейта Бостича (Keith Bostic) и Стива Киркендалля (Steve Kirkendall) за вклад в доработку глав об их редакторах. Том Дики (Tom Dickey) внес значительный вклад в подготовку главы о `vile` и таблицы опций команды `set` в приложении В. Брам Моленар (Bram Moolenaar), автор Vim, в этот раз также выполнил проверку всей книги. Роберт П. Дж. Дэй (Robert P.J. Day), Мэтт Фрай (Matt Frye), Юдит Майерсон (Judith Myerson) и Стивен Фиггинс (Stephen Figgins) дали ценные замечания по всему тексту.

Арнольд и Элберт хотят поблагодарить Энди Ора (Andy Ora) и Изабель Кункле (Isabel Kunkle) за редакторскую работу, а также всех сотрудников O'Reilly Media.

*Арнольд Роббинс (Arnold Robbins)*
*Ноф Айалон, Израиль, 2008*

*Элберт Ханна (Elbert Hannah)*
*Килдир, Иллинойс, США, 2008*

[1] Если вы ничего не слышали о *User Friendly*, зайдите на *http://www.user-friendly.org*.

# I

# Базовый и продвинутый vi

Часть I поможет быстро начать работу с vi, а также получить углубленные навыки, позволяющие использовать vi более эффективно. Материал охватывает оригинальный базовый vi, а рассматриваемые команды можно использовать в любой его версии; последующие главы посвящены популярным клонам vi. Часть I состоит из следующих глав:

- Глава 1 «Текстовый редактор vi»
- Глава 2 «Простое редактирование»
- Глава 3 «Быстрое перемещение»
- Глава 4 «За рамками основ»
- Глава 5 «Введение в редактор ex»
- Глава 6 «Глобальная замена»
- Глава 7 «Продвинутое редактирование»
- Глава 8 «Представляем клоны vi»

# 1

# Текстовый редактор vi

UNIX[1] содержит множество редакторов, которые могут обрабатывать текстовые файлы, будь то файлы, содержащие данные, исходный код или обычный текст. Таковыми являются, например, строковые редакторы ed и ex, отображающие на экране лишь одну строку из файла. Кроме того, есть экранные редакторы, например vi и Emacs, у которых на экране терминала отображается часть файла. Текстовые редакторы, основанные на X Window System, также широко доступны и становятся все популярнее. Как в GNU Emacs, так и в его потомке XEmacs допускается использование нескольких X-окон; двумя другими интересными вариантами являются редакторы sam и Acme от Bell Labs. В Vim также доступен интерфейс, основанный на X.

vi – это наиболее полезный стандартный текстовый редактор в вашей системе. (vi – это сокращение от «**vi**sual editor», то есть визуальный редактор; произносится как «ви-ай». Это хорошо проиллюстрировано на рис. 1.1.) В отличие от Emacs, он доступен практически в неизменном виде на любой современной системе UNIX, тем самым являясь подобием *лингва-франка*[2] текстового редактирования. То же можно сказать

---

[1] В настоящее время термин «UNIX» включает как коммерческие системы, выведенные из оригинальной кодовой базы UNIX, так и UNIX-подобные системы с доступным исходным кодом. Примерами первых являются Solaris (хотя проект OpenSolaris придал ему некоторое «промежуточное» положение в такой схеме классификации. – *Примеч. науч. ред.*), AIX и HP-UX, а вторых представляют GNU/Linux и разнообразные системы, основанные на BSD. Сказанное в этой книге применимо ко всем системам такого типа, если нет специальной оговорки.

[2] GNU Emacs стал универсальной версией Emacs. Единственная проблема в том, что он не является стандартной частью большинства коммерческих UNIX-систем, поэтому его следует найти и установить самостоятельно.

про ed и ex, однако пользоваться экранными редакторами намного удобнее (настолько удобнее, что строковые редакторы сейчас практически не используются). В экранном редакторе можно пролистывать страницы, перемещать курсор, удалять строки, вставлять символы и многое другое, при этом вы сразу видите результат своих действий. Экранные редакторы стали популярными благодаря возможности вносить изменения при чтении файла, как если бы вы редактировали распечатанный экземпляр, только быстрее.

***Рис. 1.1.*** *Произносите vi правильно*

Многим новичкам vi кажется непонятным и громоздким – вместо того чтобы использовать комбинации клавиш для обработки текста и позволить вам печатать обычным образом, в этом редакторе каждой клавише назначена своя команда. Когда ожидается вызов команды нажатием клавиши, говорят, что vi находится в *командном режиме* (*command mode*). Для того чтобы начать печатать собственно текст на экране, необходимо перейти в специальный *режим вставки* (*insert mode*). Следует отметить, что в vi огромное множество команд.

Однако начав освоение vi, вы обнаружите, что он хорошо продуман. Чтобы заставить его выполнить сложную работу, порой достаточно нескольких нажатий клавиш. По мере дальнейшего изучения vi вы узнаете комбинации клавиш, которые все больше и больше работы по редактированию будут передавать компьютеру, то есть туда, где ей и место.

vi (как и любой текстовый редактор) не является текстовым процессором типа «what you see is what you get» (что видишь, то и получишь). Если вам нужно создать отформатированный документ, то придется писать коды, понимаемые другой программой, которая и будет управлять видом печатаемой копии. Так, если у нескольких абзацев должен быть отступ, поместите специальный код там, где начинается и заканчивается отступ. Коды форматирования позволят вам экспериментировать или менять внешний вид печатаемых файлов. Во многих случаях они дают вам намного больше контроля над внешним видом документа,

нежели текстовый процессор. UNIX поддерживает пакет форматирования `troff`[1]. Популярными и широкодоступными альтернативами являются издательские системы TEX и LATEX[2].

(В действительности, `vi` поддерживает простые механизмы форматирования. Например, он может сам переносить слова при достижении конца строки или делать автоматические отступы у новых строк. Кроме того, в Vim версии 7 есть автоматическая проверка орфографии.[3])

Как и при любой деятельности, чем больше вы редактируете, тем быстрее освоите редактор и тем выше будет ваша производительность. А когда вы изучите все возможности `vi`, то вряд ли захотите вернуться к «более простым» редакторам.

В чем же заключается процесс редактирования? Во-первых, вам может понадобиться *вставить* (*insert*) текст (например, пропущенное или новое слово либо пропущенное предложение) или, наоборот, *удалить* (*delete*) текст (отдельный символ или целый абзац). Также должна быть возможность *менять* (*change*) буквы или слова (чтобы исправить опечатки или изменить термин). Возможно, вам придется *переносить* (*move*) текст из одной части файла в другую. Кроме того, порой требуется *копировать* (*copy*) текст, чтобы создать его дубликат в другой части файла.

В отличие от многих текстовых процессоров, изначальным режимом, или режимом «по умолчанию», в `vi` является командный режим, в котором сложные интерактивные правки можно выполнять нажатием всего лишь нескольких клавиш. (А для вставки неформатированного текста просто выполните любую из нескольких команд «вставки», после чего начинайте набор.)

В качестве базовых команд используются один или несколько символов. Например:

`i`

   вставка (insert).

`cw`

   изменить слово (change word).

---

[1] `troff` предназначен для лазерных принтеров и наборных машин и является «братом-близнецом» `nroff` – пакета форматирования для строчных принтеров и терминалов. Оба понимают один и тот же набор команд. Следуя общепринятому в UNIX соглашению, мы называем `troff` оба пакета. В настоящее время все, кто использует `troff`, работают с его GNU-версией, `groff`. За более подробной информацией обратитесь на сайт *http://www.gnu.org/software/groff/*.

[2] Для получения информации о TEXи LATEX посетите сайты *http://www.ctan.org* и *http://www.latex-project.org* соответственно.

[3] Vim «из коробки» также может делать выключку текста по левому, правому краю или по центру. – *Прим. науч. ред.*

Используя буквы в качестве команд, вы сможете редактировать файл с огромной скоростью. Вам необязательно запоминать все сочетания функциональных клавиш или растягивать пальцы, чтобы нажать неудобную комбинацию клавиш. Вам никогда не потребуется убирать руки с клавиатуры или путаться в многоуровневых меню! Многие команды можно запомнить по первым двум буквам их названий, и почти все команды следуют одинаковым правилам и связаны друг с другом.

Вообще говоря, команды vi:

- Зависят от регистра клавиши (прописная и строчная буквы соответствуют разным командам: I – не то же, что i).
- Не отображаются на экране, когда их вводят.
- Не требуют нажатия ENTER после ввода команды.

Также есть группа команд, которые отображаются в нижней строке экрана. Они начинаются со специальных символов. Косая черта (/) и знак вопроса (?) запускают команды поиска; об этом рассказано в главе 3. Все команды ex начинаются с двоеточия (:); они используются в строковом редакторе ex. Этот редактор доступен при работе в vi, поскольку ex является базовым редактором, а vi – это просто его «визуальный» режим. Команды и понятия ex обсуждаются в главе 5, но уже в этой главе вы узнаете о команде ex для выхода из файла без сохранения.

# Краткая историческая справка

Перед погружением во все тонкости vi полезно понять, какими глазами он «смотрит» на ваше окружение. В частности, это поможет осмыслить кажущиеся туманными сообщения vi об ошибках, а также разобраться, насколько клоны vi развились по сравнению с оригиналом.

vi восходит к тем временам, когда пользователи работали за терминалами, которые последовательно подсоединялись к центральным компьютерам. По всему миру были распространены сотни разновидностей терминалов. Каждый из них выполнял одни и те же действия (очистка экрана, перемещение курсора и т. п.), однако команды для управления ими были различными. Кроме того, система UNIX позволяет выбирать, какие символы использовать для забоя, генерации сигнала прерывания и других команд, применяемых на последовательных терминалах, например подавления и возобновления вывода. Такие функции управлялись (и до сих пор управляются) командой stty.

Первоначальная версия vi, созданная в Калифорнийском университете в Беркли (University of California, Berkeley, UCB), абстрагировала информацию об управлении терминалом из кода (который было сложно изменить) в текстовую базу данных возможностей терминала (которую изменить было легко), поддерживаемую библиотекой termcap (от **term**inal **cap**abilities). В начале 80-х в System V были внедрены база данных, содержащая двоичную информацию о терминалах, и библиотека terminfo

(от **term**inal **info**rmation). Эти две библиотеки были в основном функционально эквивалентными. Чтобы сообщить vi, каким именно терминалом вы пользуетесь, необходимо было установить переменную окружения TERM. Обычно это проделывалось файлом запуска оболочки, таким как .profile или .login.

В настоящее время все пользуются эмуляторами терминалов в графическом окружении (например, xterm). Как правило, система сама заботится о задании переменной TERM. (Конечно, можно вызвать vi и в неграфической консоли вашего ПК. Это может очень сильно помочь при восстановлении системы в однопользовательском режиме. Хотя осталось немного людей, которые предпочли бы так работать на регулярной основе.) Скорее всего, для повседневного использования вы выберите графическую версию vi, например Vim или один из других клонов. В Microsoft Windows или Mac OS X он, возможно, будет запускаться по умолчанию. Однако когда вы запускаете vi (или какой-нибудь другой столь же винтажный экранный редактор) в эмуляторе терминала, он все еще использует TERM и данные termcap или terminfo, а также обращает внимание на установки stty. Запуск в эмуляторе терминала – такой же простой способ изучить vi, как и любой другой.

Другим важным для понимания vi фактом является то, что он развивался в то время, когда системы UNIX были намного менее стабильными, чем сейчас. Пользователи тех лет должны были быть готовыми к сбою в системе в любой момент, а в vi была предусмотрена поддержка восстановления тех файлов, которые редактировались в момент системного сбоя[1]. Так что если во время изучения vi вы увидите описание различных возникающих проблем, вспомните историю его развития.

## Открытие и закрытие файлов

vi можно использовать для редактирования произвольного текстового файла. vi копирует редактируемый файл в *буфер* (временно выделяемую область памяти), отображает буфер (хотя в каждый момент времени вы видите только ту часть, которая поместилась на экране) и позволяет вам добавлять, удалять или менять текст. При сохранении результатов редактирования vi копирует отредактированный буфер обратно в постоянный файл, замещая старый файл с тем же именем. Не забывайте, что вы всегда работаете с *копией* вашего файла, хранимой в буфере, поэтому все ваши правки не изменят первоначальный файл, пока вы не сохраните буфер. Сохранение изменений часто называют «сохранением буфера» или просто «сохранением файла».

---

[1] К счастью, ситуации такого рода случаются гораздо реже, хотя системы все еще могут аварийно завершить работу из-за внешних причин, например из-за прекращения подачи питания.

## Открытие файла

vi – это команда UNIX, которая вызывает редактор vi для существующего или для совершенно нового файла. Синтаксис использования этой команды следующий:

```
$ vi [filename]
```

Скобки, показанные в этой строке, означают, что имя файла – необязательный параметр. Сами скобки набирать не надо. Знак $ – это приглашение командной строки UNIX. Если не указать имя файла, то vi откроет безымянный буфер. Имя можно указать при сохранении буфера в файл. А пока давайте остановимся на указании имени файла в командной строке.

Имя файла должно быть уникальным в пределах одного каталога. Оно может содержать любой из 8-битных символов, кроме знака косой черты (/), зарезервированного в качестве разделителя между файлами и каталогами в пути файла, и ASCII NUL – символа с нулевыми разрядами. В имени файла можно даже использовать пробелы; в этом случае перед пробелом следует поставить обратную косую черту (\). Тем не менее на практике имена файлов в основном содержат различные сочетания больших и маленьких букв, цифр, символов точки (.) и подчеркивания (_). Помните, что UNIX чувствителен к регистру: строчные буквы отличаются от прописных. Также не забывайте нажимать на ENTER, чтобы сообщить UNIX о том, что вы закончили ввод команды.

Если вы хотите создать в каталоге новый файл, задайте в команде vi новое имя файла. Например, чтобы в текущем каталоге открыть новый файл с именем practice, введите:

```
$ vi practice
```

Поскольку это новый файл, буфер будет пустым, и на экране вы увидите следующее:

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

Тильды (~) в левом столбце экрана указывают, что в файле нет никакого текста, нет даже пустых строк. Строка приглашения (также называемая строкой состояния) внизу экрана отображает имя и состояние файла.

Если вы укажете имя любого из существующих в каталоге файлов, то сможете отредактировать его. Предположим, что существует файл с абсолютным путем /home/john/letter. Если вы уже находитесь в каталоге /home/john, используйте относительный путь к файлу. Например

```
$ vi letter
```

выдаст на экран файл letter.

Если вы находитесь в другом каталоге, введите полный путь к файлу, чтобы начать его редактирование:

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

## Проблемы при открытии файлов

- *При запуске* vi *появляется сообщение* [open mode]

  Возможно, неправильно распознается тип вашего терминала. Немедленно выйдите из сеанса редактирования, введя команду :q. Проверьте переменную окружения $TERM. Ей нужно присвоить имя вашего терминала. Или можете попросить системного администратора дать вам правильное значение типа терминала.

- *Вы видите одно из следующих сообщений*:

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

  Либо тип вашего терминала не опознан, либо что-то не так с его записью в вашем terminfo или termcap. Введите :q, чтобы выйти. Проверьте переменную окружения $TERM или попросите системного администратора выбрать тип терминала для вашего окружения.

- *Появляется сообщение* [new file]*, когда вы считаете, что файл уже существует.*

  Проверьте, правильный ли регистр символов вы использовали в имени файла (имена файлов в UNIX чувствительны к регистру). Если все верно, возможно, вы находитесь в другом каталоге. Введите :q для выхода. После этого проверьте, находитесь ли вы в том же каталоге, что и файл (введите pwd в командной строке UNIX). Если вы в нужном каталоге, выведите список содержащихся в нем файлов (с помощью ls) и проверьте, нет ли файла под немного другим именем.

- *Вы запустили* vi*, однако попали в приглашение с двоеточием (что говорит о том, что вы находитесь в режиме строкового редактирования* ex*).*

  Возможно, вы ввели прерывание перед тем, как vi успел отрисовать экран. Войдите в vi, введя в приглашении ex (:) команду vi.

- *Появляется одно из следующих сообщений*:

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

  «Read only» означает, что вы можете только просматривать файл; ваши изменения не могут быть сохранены. Возможно, вы запустили vi

в режиме просмотра (либо через `view`, либо как `vi -R`) либо у вас нет прав на запись этого файла. Обратитесь к разделу «Проблемы при сохранении файлов» на стр. 30.

- *Появляется одно из следующих сообщений*:

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

  Файл, который вы хотите отредактировать, не является обычным текстовым файлом. Введите `:q!` для выхода и проверьте этот файл, например командой `file`.

- *При вводе* `:q` *по одной из вышеназванных причин появляется сообщение*:

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

  Вы ненароком внесли изменение в файл. Для выхода из `vi` введите `:q!`. В этом случае изменения, сделанные во время сеанса, не будут сохранены.

## Образ действия

Как упоминалось ранее, концепция текущего «режима» является фундаментальной в работе `vi`. Существуют два режима: *режим вставки* и *командный режим*. Сразу после запуска активен командный режим, в котором каждое нажатие клавиши вызывает команду. В режиме вставки все, что вы печатаете, становится содержимым вашего файла.

Иногда случайно можно попасть в режим вставки или, наоборот, ненароком выйти из него. В любом случае то, что вы введете, скорее всего, нежелательно отразится на содержимом файла.

Нажмите ESC, чтобы попасть в командный режим. Если вы уже в нем, `vi` даст звуковой сигнал (beep) при нажатии ESC. (Поэтому командный режим иногда называют сигнальным режимом.)

Благополучно перейдя в командный режим, вы можете исправить любые случайные изменения, после чего вернуться к редактированию вашего текста.

## Сохранение файла и выход

В любой момент можно прекратить работу с файлом, сохранить правки и вернуться в приглашение командной строки UNIX. Команда `vi`, которая сохраняет изменения и прекращает работу редактора, называется `ZZ`. Обратите внимание, что `ZZ` пишется прописными буквами.

Предположим, вы создали файл под названием practice и ввели в нем шесть строчек текста. Чтобы сохранить файл, сначала нажатием ESC проверьте, что вы попали в командный режим, после чего введите ZZ.

| Клавиши | Результат |
|---|---|
| ZZ | `"practice" [New file] 6 lines, 320 characters`<br>Введена команда записи ZZ. Ваш файл сохранится как обычный файл UNIX. |
| ls | `ch01 ch02 practice`<br>Вывод списка файлов в каталоге покажет, что вы создали новый файл practice. |

Результаты редактирования можно сохранить и с помощью команд ex. Чтобы сохранить (write) файл, не выходя из vi, введите :w. Если вы ничего не меняли в файле, выйти можно с помощью команды :q, а введя :wq, вы сохраните изменения и покинете vi. (:wq эквивалентно ZZ.) В главе 5 мы подробно расскажем об использовании команд ex. Сейчас просто запомните эти несколько команд для записи и сохранения файлов.

# Выход без сохранения правок

При первом знакомстве с vi, особенно если вы бесстрашный экспериментатор, вам могут понадобиться две другие команды ex, чтобы избавиться от созданной вами путаницы.

Если вы захотите отменить все сделанные за сеанс изменения и вернуться к первоначальному файлу, то команда

```
:e! ENTER
```

вернет вас к последней сохраненной версии файла, и вы сможете начать все заново.

Если же вы хотите отказаться от изменений и выйти из vi, то команда

```
:q! ENTER
```

осуществит выход из редактируемого файла и возврат в приглашение UNIX. Обе эти команды приведут к потере всех изменений, сделанных в буфере со времени последнего сохранения. Обычно vi не позволяет отказаться от изменений. Восклицательный знак, добавленный к командам :e или :q, заставит vi отменить этот запрет и выполнить операцию, несмотря на то, что буфер был изменен.

## Проблемы при сохранении файлов

- *Вы пытаетесь записать файл, но получаете одно из следующих сообщений*:

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

  Введите `:w! file`, чтобы перезаписать существующий файл, или `:w newfile`, чтобы сохранить текущую редакцию в новом файле.

- *Вы хотите записать файл, но у вас нет разрешения на запись для него. Вам выдается «Permission denied.»*

  Используйте команду `:w newfile`, чтобы записать содержимое буфера в новый файл. При наличии прав на запись для этого каталога вы сможете с помощью команды `mv` заменить первоначальную версию новым файлом. Если у вас нет разрешения на запись для этого каталога, введите `:w pathname/file`, чтобы записать буфер в том каталоге, где у вас есть разрешение на запись (например, домашний каталог или `/tmp`).

- *Вы пытаетесь записать файл, но получаете сообщение о том, что файловая система переполнена.*

  Введите `:!rm junkfile`, чтобы удалить (большой) ненужный файл, тем самым освободив немного места. (Если команду `ex` начать с восклицательного знака, то вы получите доступ в UNIX.)

  Или введите `:!df`, чтобы посмотреть, есть ли свободное место в другой файловой системе. Если есть, выберите каталог в той системе и запишите файл туда, воспользовавшись командой `:w pathname`. (`df` — это команда UNIX, которая проверяет свободное место на дисках; название происходит от **d**isk **f**ree.)

- *Система переводит вас в открытый режим (open mode) и сообщает, что файловая система переполнена.*

  Диск переполнен временными файлами `vi`. Введите `:!ls /tmp`, чтобы посмотреть, есть ли файлы, которые можно удалить, дабы получить немного места на диске[1]. Если таковые имеются, создайте временную оболочку UNIX, из которой вы сможете удалить эти файлы, или обратитесь к другим командам UNIX. Оболочку можно создать, если ввести `:sh`; для выхода из оболочки и возврата в `vi` нажмите CTRL-D или введите команду `exit`. (В современных системах UNIX при использовании оболочки с управлением заданиями можно просто нажать CTRL-Z, чтобы приостановить `vi` и вернуться в командную

---

[1] `vi` может хранить временные файлы в `/usr/tmp`, `/var/tmp` или в вашем домашнем каталоге; возможно, вам придется немного покопаться, чтобы разузнать, что именно занимает столько места. Vim обычно держит свои временные файлы в том же каталоге, что и редактируемый файл.

строку UNIX; для возврата в vi введите fg.) После высвобождения места на диске сохраните ваш файл командой :w!.

- *Вы пытаетесь записать файл, но получаете сообщение о том, что достигнуты дисковые квоты.*

  Попробуйте заставить систему записать ваш буфер с помощью команды :pre (сокращение от :preserve). Если это не сработало, поищите файлы для удаления. Воспользуйтесь командой :sh (или CTRL-Z, если система поддерживает управление заданиями), чтобы выйти из vi и удалить файлы. Для возврата в vi нажмите CTRL-D (или введите fg). Затем запишите файл командой :w!.

## Упражнения

Единственный способ выучить vi – это практика. Сейчас вы уже знаете достаточно, чтобы создать новый файл и вернуться в приглашение UNIX. Создайте файл с именем practice, внесите в него немного текста, затем сохраните файл и выйдите.

| | |
|---|---|
| Открыть файл с именем practice в текущем каталоге: | vi practice |
| Вставьте текст: | i *любой текст* |
| Возврат в командный режим: | ESC |
| Выход из vi с сохранением правок: | ZZ |

# 2

# Простое редактирование

Эта глава, построенная в виде руководства, познакомит вас с редактированием в `vi`. Вы узнаете, как перемещать курсор и делать простые правки. Если вы еще ни разу не работали в `vi`, то лучше прочтите эту главу целиком.

Последующие главы призваны углубить ваши навыки, что позволит вам работать быстрее и эффективнее. Одно из главных *преимуществ* `vi` для опытного пользователя – огромный выбор опций (при этом один из главных *недостатков* для новичка – огромное число команд редактора).

Нельзя освоить `vi`, просто запомнив все команды. Начните с изучения простых команд, о которых рассказывается в этой главе, и обращайте внимание на общие шаблоны их использования.

Изучая `vi`, берите на заметку задачи, которые вы можете поручить редактору, а затем найдите команды, решающие их. В последующих главах вы узнаете о продвинутых свойствах `vi`, но прежде чем браться за сложное, нужно освоить азы.

В этой главе рассказывается о том, как:

- Перемещать курсор
- Добавлять и менять текст
- Удалять, перемещать и копировать текст
- Переходить в режим вставки разными способами

## Команды vi

В `vi` есть два режима: командный и режим вставки. При входе в файл вы оказываетесь в командном режиме, и редактор ждет ввода команд. Они позволяют перемещаться на любое место в файле, производить правки или переходить в режим вставки, чтобы добавить новый текст.

Команды также нужны для выхода из файла (с сохранением изменений или без), чтобы вернуться в командную строку UNIX.

Оба режима работы можно рассматривать как две разные «клавиатуры». В режиме вставки ваша клавиатура работает подобно печатной машинке. В командном режиме каждая клавиша имеет свое значение или вызывает какую-либо инструкцию.

Есть несколько способов сообщить `vi` о переходе в режим вставки. Самый простой из них – нажать клавишу `i`. Сама буква `i` на экране не появится, но после ее нажатия все, что вы набираете, *возникнет* на экране и будет передаваться в буфер. При этом курсор отмечает место для вставки нового текста[1]. Для выхода из режима вставки нажмите ESC. Это действие переместит курсор на один символ назад (он встанет на последний введенный вами символ) и вернет `vi` в командный режим.

Например, вы открыли новый файл и хотите вставить туда слово «introduction». Если ввести `iintroduction`, то на экране появится:

```
introduction
```

При открытии нового файла `vi` начинает работу в командном режиме и понимает первую клавишу (`i`) как команду вставки. После этого все введенные символы рассматриваются им как текст, пока вы не нажмете ESC. Чтобы исправить ошибку в режиме вставки, вернитесь назад с помощью клавиши BACKSPACE и наберите символ заново. В зависимости от типа используемого вами терминала BACKSPACE может либо удалять набранный текст с экрана, либо перемещать курсор поверх него. В любом случае замещаемый текст будет удален. Обратите внимание, что вы не сможете использовать клавишу BACKSPACE дальше того места, где включили режим вставки. (Если в Vim выключить совместимость с `vi`, то там возможно перемещение курсора дальше места начала режима вставки.)

В `vi` есть опция, позволяющая определить правый отступ и выполняющая возврат каретки всякий раз при его достижении. Пока же во время вставки текста для перехода на новую строку используйте ENTER.

Иногда сложно определить, в каком из двух режимов вы находитесь. Если `vi` ведет себя не так, как ожидалось, нажмите ESC один или два раза, чтобы проверить, в каком вы режиме. Звуковой сигнал означает, что в командном.

# Перемещение курсора

Скорее всего, во время сеансов редактирования вы будете уделять мало внимания вставке нового текста, так как большая часть времени уйдет на правку существующего.

[1] В некоторых версиях строка состояния показывает, что вы находитесь в режиме вставки.

В командном режиме можно перемещать курсор в любую часть файла. Поскольку основное редактирование (изменение, удаление и копирование текста) начинается с перемещения курсора в тот фрагмент, который вы хотите изменить, вам, вероятно, захочется переместить его туда как можно быстрее.

В vi есть команды, которые перемещают курсор:

- Вверх, вниз, влево или вправо на один *символ*.
- Вперед или назад на *текстовые* блоки, такие как слова, предложения или абзацы.
- Вперед или назад по файлу на один *экран*.

На рис. 2.1 знак подчеркивания указывает на текущее положение курсора. Кружки показывают, какие его перемещения относительно текущей позиции вызовут те или иные команды vi.

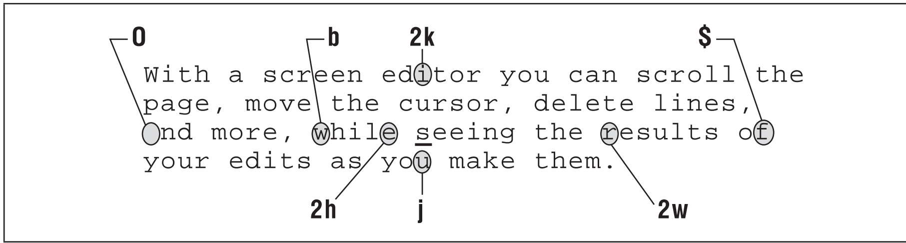



***Рис. 2.1.*** *Примеры команд перемещения курсора*

## Простые движения

Клавиши h, j, k и l, удобно расположенные под кончиками ваших пальцев, перемещают курсор:

h

  На один символ влево.

j

  На одну строку вниз.

k

  На одну строку вверх.

l

  На один символ вправо.

Также для перемещения вверх и вниз можно использовать курсорные клавиши (←, ↓, ↑, →), + и – либо ENTER и BACKSPACE, но они расположены в стороне. Поначалу может показаться, что пользоваться буквенными кнопками вместо курсорных неудобно, но потом вы поймете, что это одно из лучших качеств vi, позволяющее перемещаться по тексту, не отрывая пальцев от центра клавиатуры.

Перед перемещением курсора нажмите ESC, чтобы убедиться, что вы находитесь в командном режиме. Используйте `h`, `j`, `k` и `l`, чтобы двигаться вперед или назад относительно текущего положения курсора. Достигнув предела в каком-либо направлении, вы услышите звуковой сигнал, и курсор остановится. Например, находясь в начале или конце строки, нельзя воспользоваться `h` или `l` для возврата в конец (или начало) предыдущей (или следующей) строки, поэтому следует нажать `j` или `k`[1]. Аналогично нельзя ни подвинуть курсор дальше знака тильды (~), который обозначает строку без текста, ни переместить курсор выше первой строки в файле.

## Числовые аргументы

Перед командами перемещения можно указывать число. Рисунок 2.2 показывает, как команда `4l` переместит курсор на четыре символа вправо, как если бы вы нажали четыре раза на `l` (`llll`).

Способность дублировать команды дает больше возможностей и контроля над каждой изученной командой. Не забывайте об этом, когда будете знакомиться с другими командами.

```
     4l
With a screen editor you can scroll the
```

***Рис. 2.2.*** *Дублирование команд с помощью коэффициентов*

## Перемещение в строке

Ранее при сохранении файла `practice` `vi` выдал сообщение о количестве строк в этом файле. *Строка* не обязательно имеет одинаковую длину со строкой на экране (часто ограниченной 80 символами). Строка – это любой текст между двумя переводами строк. (Символ *перевода строки* вставляется в файл всякий раз при нажатии ENTER в режиме вставки.) Если перед нажатием на ENTER вы введете 200 символов, `vi` воспримет все эти 200 символов как одну строку (несмотря на то, что визуально они займут несколько строк на экране).

Как мы уже упоминали в главе 1, `vi` имеет опцию, позволяющую установить расстояние от правого поля, при котором автоматически вставляется перевод строки. Эта опция называется `wrapmargin` (сокращается до `wm`). Можно установить `wrapmargin` на 10 символах:

```
:set wm=10
```

---

[1] Vim с установленной опцией `nocompatible` позволяет переместиться с конца строки на начало следующей с помощью клавиш `l` или пробела.

Данная команда не повлияет на уже набранные строки. В главе 7 мы поговорим о задании опций подробнее, но конкретно эту нельзя было отложить на потом!

Если вы не используете автоматический перенос `wrapmargin`, то следует завершать строки возвратом каретки, чтобы они имели приемлемый размер.

Для перемещения по строке есть две очень полезные команды:

`0` **(цифра «ноль»)**

Перемещает в начало строки.

`$`

Перемещает в конец строки.

В следующем примере отображаются номера строк. (В `vi` их можно вывести, если воспользоваться опцией `number`, которая включается при вводе `:set nu` в командном режиме. Эта операция описана в главе 7.)

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

Количество логических строк (3) не соответствует тому, что вы видите на экране (5). Если ввести `$`, когда курсор расположен на букве *d* слова *delete*, то он перейдет на точку после слова *them*. При вводе `0` курсор переместится назад к букве *m* в слове *move* в самом начале строки 2.

## Перемещение по текстовым блокам

Курсор можно перемещать и по текстовым блокам, таким как слова, предложения, абзацы и т. д. Команда `w` перемещает курсор вперед на одно слово, при этом символы и знаки препинания тоже считаются словами. Следующая строка демонстрирует перемещение курсора с помощью `w`:

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

Командой `W` можно перемещаться без учета символов и знаков препинания (можете рассматривать это как «обобщенное» или «большое» слово – Word).

Передвижение курсора при помощи `W` выглядит так:

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

Для движения по словам в обратном направлении используется `b`. Прописная `B` позволит делать то же самое, но без учета пунктуации.

Как упоминалось ранее, команды перемещения могут снабжаться числовыми аргументами, то есть как `b`, так и `w` могут умножаться на числа. `2w` передвинет курсор вперед на два слова, а `5B` — назад на пять слов без учета знаков препинания.

Для перехода на определенную строку используйте команду `G`. Просто `G` переместит вас в конец файла, `1G` — в его начало, а `42G` — на строку под номером 42. Подробнее об этом читайте в разделе «Команда G (Go To)» на стр. 64.

В главе 3 мы рассмотрим перемещение по предложениям и абзацам, а пока попрактикуйтесь в перемещении курсора уже известными вам командами в сочетании с числовыми указателями.

# Простая правка текста

При наборе текста в файле он редко получается идеальным. Можно обнаружить опечатки или захотеть улучшить отдельные фразы. Также, если вы пишете программу, то она может содержать ошибки. После ввода текста полезно иметь возможность изменить, удалить, переместить или скопировать его. На рис. 2.3 показаны разновидности правок, которым может подвергнуться файл. На исправления указывают отметки корректора.

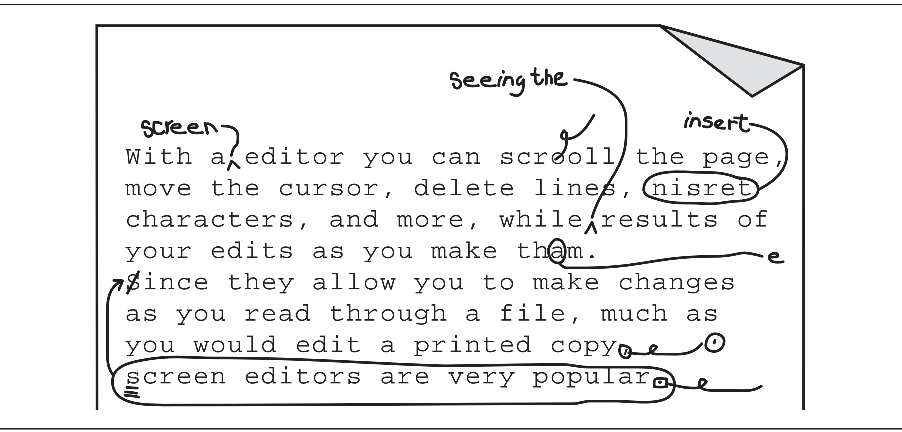



***Рис. 2.3.*** *Правки с отметками корректора*

В `vi` можно выполнять любые из этих изменений с помощью всего нескольких клавиш: `i` — для вставки (вы это уже знаете), `a` (от *append*) — для добавления, `c` — для изменения (от *change*) и `d` — для удаления (*delete*). Чтобы переместить или скопировать текст, используйте последовательности команд. Например, текст перемещается путем его «удаления» командой `d` и последующей «вставки» командой `p` (*put*); при копировании текста следует сначала нажать `y` для «копирования» (*yank*), затем `p` — для

«вставки». В данной главе описаны все команды редактирования. На рис. 2.4 показаны команды vi, используемые в правках на рис. 2.3.

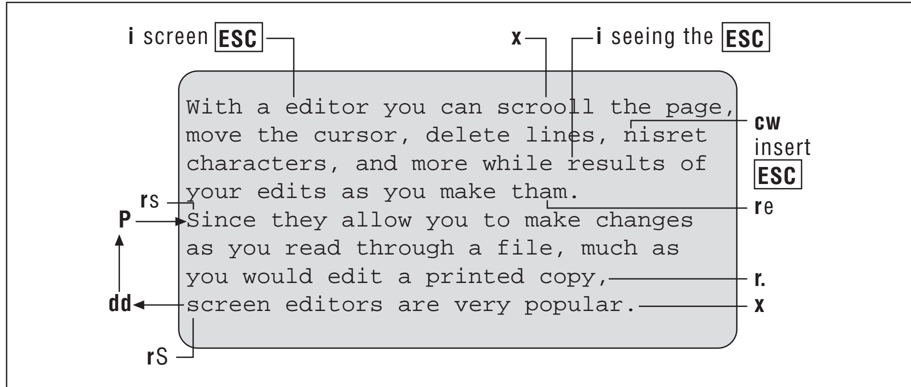



***Рис. 2.4.*** *Редактирование командами vi*

## Ввод нового текста

Вы уже знакомы с командой вставки, которая применяется для ввода текста в новом файле. Эта команда также используется при редактировании существующего текста и добавлении недостающих символов, слов и предложений. Предположим, в файле practice есть следующее предложение:

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

Здесь отмечено положение курсора. Чтобы вставить в начало предложения *With a screen editor*, введите следующее:

| Клавиши | Результат |
|---|---|
| 2k | `you can scroll` <br> `the page, move the cursor, delete` <br> `lines, and insert characters.` <br><br> Командой k переместите курсор вверх на две строки, где нужно вставить новые слова. |
| iWith a | `With a you can scroll` <br> `the page, move the cursor, delete` <br> `lines, and insert characters.` <br><br> Введите i для перехода в режим вставки и начните набирать текст. |

| Клавиши | Результат |
|---|---|
| `screen editor`<br>ESC | `With a screen editor█you can scroll`<br>`the page, move the cursor, delete`<br>`lines, and insert characters.`<br>После ввода текста нажмите ESC, чтобы выйти из режима вставки в командный режим. |

## Добавление текста

С помощью команды добавления `a` (*append*) вы можете добавлять текст в любом месте файла. Эта команда работает аналогично `i`, но позволяет вставлять текст *после* курсора, а не *перед* ним. Возможно, вы заметили, что при нажатии на `i` для перехода в режим вставки курсор не двигается, пока вы не введете что-нибудь. Напротив, после нажатия на `a` и перехода в режим вставки курсор переместится на одну позицию вправо. Теперь при вводе текст появится после изначального положения курсора.

## Изменение текста

Текст в файле можно заменять при помощи соответствующей команды `c` (*change*). Чтобы сообщить ей, сколько именно текста нужно поменять, сочетайте ее с командой перемещения. В этом случае команда перемещения служит как текстовый объект, на который влияет команда `c`. Например, `c` можно использовать для изменения текста, расположенного:

`cw`

До конца слова.

`c2b`

Назад на два слова.

`c$`

До конца строки.

`c0`

До начала строки.

После вызова команды изменения указанный отрезок текста можно заменить любым количеством нового, пустой строкой, одним словом или сотней строк. Подобно `i` и `a`, команда `c` оставляет пользователя в режиме вставки до нажатия ESC.

Если замена затрагивает только текущую строку, `vi` помечает конец изменяемого текста знаком `$`, и тогда видно, какая часть строки изменяется (см. пример к `cw` чуть ниже).

## Слова

Чтобы изменить слово, используйте сочетание команд `c` (*change*) и `w`. Вы можете заменить слово (`cw`) на более длинное или короткое (или на любое количество текста). `cw` можно рассматривать как «удалить помеченное слово и добавлять новый текст, пока не будет нажата ESC».

Представим, что у нас есть следующая строка в файле `practice`:

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

и мы хотим поменять *an* на *screen*. Нужно изменить всего одно слово:

| Клавиши | Результат |
|---|---|
| `w` | `With an editor you can scroll the page,` <br> С помощью `w` переместитесь на то место, где хотите начать изменение. |
| `cw` | `With a$ editor you can scroll the page,` <br> Задайте команду замены слова. Конец изменяемого текста отмечается символом `$` (знак доллара). |
| `a screen` | `With a screen editor you can scroll the page,` <br> Введите текст замены и нажмите ESC для возврата в командный режим. |

`cw` также работает с фрагментом слова. Например, чтобы исправить *spelling* на *spelled*, подведите курсор к *i*, введите `cw`, затем – *ed* и завершите ввод, нажав на ESC.

## Строки

Чтобы полностью заменить текущую строку, используйте команду изменения `cc`. Она меняет всю строку независимо от положения курсора в ней, подставляя вместо нее любое количество текста, введенного перед нажатием на ESC.

Работа команды типа `cw` отличается от команды типа `cc`. При вызове `cw` старый текст остается на месте, пока вы не начнете печатать поверх него. После этого старый текст стирается при нажатии ESC. А если воспользоваться командой `cc`, то старый текст удаляется сразу, оставляя пустое место для вставки нового.

Подход «печать поверх» применяется для всех команд изменения, которые затрагивают часть строки, тогда как подход «пустая строка» используется, если команда изменения влияет на целую строку или больше.

### Общий вид команд vi

Можно заметить, что рассмотренные выше команды изменения отвечают следующему шаблону:

*(command)(text object)*

*command* — это команда изменения `c`, а *text object* — команда перемещения (без скобок). Однако `c` — не единственная команда, требующая текстового объекта. Команды `d` (удаления) и `y` (копирования) также отвечают этому шаблону.

Помните, что команды перемещения могут сопровождаться числовыми аргументами, то есть к текстовым объектам команд `c`, `d` и `y` можно добавлять числа. Например, команды `d2w` и `2dw` удаляют два слова. Зная об этом, можно увидеть, что большинство команд `vi` следуют более общему формату:

*(command)(number)(text object)*

или в эквивалентной записи:

*(number)(command)(text object)*

*number* и *command* необязательны. Без них получится просто команда перемещения. Если вы добавите число, перемещение будет повторяться. С другой стороны, сочетая команду (`c`, `d` или `y`) с текстовым объектом, можно получить команду редактирования.

Когда вы осознаете, сколько таких комбинаций можно составить, `vi` станет для вас поистине мощным редактором!

Команда `C` меняет символы от текущей позиции курсора до конца строки. Ее действие сходно с сочетанием `c` со специальным индикатором конца строки (`c$`).

На самом деле, `cc` и `C` являются сокращениями других команд, поэтому они не отвечают общему синтаксису, принятому в `vi`. Мы рассмотрим другие сокращения, когда будем говорить об удалении и копировании.

## Символы

Еще одна замена производится с помощью команды `r`. Эта команда заменяет отдельный символ другим. В данном случае после правки не нужно нажимать ESC, чтобы вернуться в командный режим. В следующей строке есть опечатка:

```
Pith a screen editor you can scroll the page,
```

Здесь нужно поменять всего одну букву. Нет необходимости использовать cw, так как в этом случае придется заново набирать все слово. Введите r, чтобы заменить всего один символ на позиции курсора:

| Клавиши | Результат |
|---|---|
| rW | With a screen editor you can scroll the page,<br>Введите команду замены r, а затем – новый символ *W*. |

## Подстановка текста

Предположим, вам нужно изменить не все слово, а лишь несколько символов. Команда подстановки (s, от *substitute*) сама по себе заменяет отдельный символ. Если перед ней указать число, вы сможете заменить несколько символов. Как и в случае команды c, последний символ текста помечается знаком $, чтобы было видно, сколько текста будет заменено.

Команда S, аналогично другим командам, вводимым прописными буквами, позволяет менять целые строки. В отличие от C, меняющей остаток строки после текущего положения курсора, S удаляет всю строку независимо от положения курсора. При этом vi переводит пользователя в режим вставки в начало строки. Число, предшествующее команде, задает количество заменяемых строк.

Команды s и S переводят пользователя в режим вставки; по окончании ввода нового текста нажмите ESC.

Команда R, как и ее строчный аналог, заменяет текст. Разница между ними в том, что R просто переводит в режим замещения. При этом вводимые символы будут заменять текст на экране символ за символом до нажатия ESC. Замещать текст таким образом можно не более чем в одной строке; при нажатии ENTER vi начнет новую строку, тем самым переведя пользователя в режим вставки.

# Смена регистра

Смена регистра у буквы – особый вид замены. Команда «тильда» (~) меняет строчную букву на прописную, а прописную – на строчную. Поместите курсор на букву, регистр которой вы хотите изменить, и нажмите ~. Регистр символа поменяется, а курсор перейдет к следующему знаку.

В старых версиях vi для этой команды нельзя было указывать числовой префикс или текстовый объект. Современные вариации редактора позволяют это делать.

Если вам нужно поменять регистр сразу нескольких строк, то следует пропустить текст через команду UNIX, например через tr, как описано в главе 7.

# Удаление текста

Любой текст в файле можно удалить при помощи команды удаления d (*delete*). Как и команда изменения, d требует задания текстового объекта (количества текста для обработки). Вы можете удалять по словам (dw), по строкам (dd или D) или с помощью других команд перемещения, о которых узнаете позже.

## Слова

Предположим, в файле набран такой текст:

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

При этом курсор стоит на отмеченном месте. Допустим, вы хотите удалить слово *are* в первой строке:

| Клавиши | Результат |
|---|---|
| 2w | `Screen editors are are very popular,`<br>`since they allowed you to make`<br>`changes as you read through a file.`<br>Переместите курсор туда, где нужно внести исправление (*are*). |
| dw | `Screen editors are very popular,`<br>`since they allowed you to make`<br>`changes as you read through a file.`<br>Вызовите команду удаления (dw), чтобы удалить слово *are*. |

dw удалит слово, которое начинается там, где стоит курсор. Обратите внимание, что пробел после слова тоже удалился.

dw также можно применять для удаления части слова. Например:

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

Вы хотите удалить *ed* на конце слова *allowed*.

| Клавиши | Результат |
|---|---|
| dw | `since they allowyou to make`<br>Вызовите команду удаления слова (dw), начиная с положения курсора. |

dw всегда удаляет пробел, стоящий перед следующим словом в строке. Однако в нашем примере это не нужно. Чтобы пробел между словами не удалялся, используйте команду de. Она удалит символы только до конца слова, включая пунктуацию.

Также можно удалять в обратном направлении (db), до конца строки (d$) или до ее начала (d0).

## Строки

Команда dd удаляет всю строку, на которой находится курсор, а не ее часть. Как и аналог этой команды cc, dd — это особая команда. Возьмем тот же текст, который использовался в предыдущем примере, и пусть курсор стоит в первой строке, как показано ниже:

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

Вы сможете удалить первые две строки:

| Клавиши | Результат |
|---|---|
| 2dd | `changes as you read through a file.`<br>Введите команду для удаления двух строк (2dd). Обратите внимание, что хотя курсор и не стоял в начале строки, она все равно удалилась целиком. |

Команда D удаляет символы, начиная с позиции курсора и до конца строки. (D — это сокращение для d$.) Например, если курсор стоит как показано:

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

вы сможете удалить часть строки справа от него:

| Клавиши | Результат |
|---|---|
| D | `Screen editors are very popular,`<br>`since they allow you to make`<br>`changes`<br>Введите команду для удаления части строки справа от курсора (D). |

### Символы

Часто необходимо удалить только один или два символа. Подобно команде r, заменяющей отдельный символ, x удаляет только символ, на котором стоит курсор. В строке:

```
zYou can move text by deleting text and then
```

вы можете удалить букву *z*, нажав на x[1]. Заглавная X удаляет символ, стоящий перед курсором. Числовой префикс перед любой из этих команд указывает на число символов, которые будут удалены. Например, 5x удалит пять символов справа от курсора.

### Проблемы, возникающие при удалении

- *Вы удалили не тот текст и хотите вернуть его обратно.*

  Есть несколько способов восстановить удаленный текст. Сразу после удаления нажмите u, чтобы отменить последнюю команду (например, dd). Это сработает, если вы не вводили никаких других команд, поскольку u отменяет только самую последнюю. В качестве альтернативы U восстановит строку в первоначальном состоянии, то есть до внесения в нее *каких-либо* изменений.

  Кроме того, у вас есть возможность восстановить недавнее удаление с помощью команды p, так как vi сохраняет последние девять удалений в девяти пронумерованных буферах. Например, если вы знаете, что для восстановления нужно отменить третье удаление, введите:

  ```
  "3p
  ```

  чтобы «вставить» содержимое буфера номер 3 в строку под курсором.

  Это сработает только при удалении *строк*. Слова и фрагменты строк не сохраняются в буфере. Если вам нужно восстановить удаленное слово или часть строки, а u не работает, используйте самостоятельную команду p. Она восстановит последнее удаление. В следующих подразделах мы вернемся к командам u и p.

  Обратите внимание, что Vim поддерживает «бесконечную» отмену, что сильно облегчает жизнь. В разделе «Отмена отмен» на стр. 332 об этом рассказывается подробнее.

## Перемещение текста

В vi можно перемещать текст путем его удаления, а затем вставки удаленного текста в любом другом месте файла, примерно как при действии «вырезать и вставить». Всякий раз когда вы удаляете текстовый блок, это удаление сохраняется в специальном буфере. Перейдите на другое место в файле и используйте команду вставки (p), чтобы разместить

---

[1] Мнемоника команды x в том, что она подобна «забивке» ошибок иксами на печатной машинке. Хотя кто сейчас ими пользуется?

текст в новой позиции. Так вы можете перемещать любой фрагмент текста, хотя удобнее перемещать строки, а не отдельные слова.

Команда вставки `p` (от *put*) вставляет текст из буфера после позиции курсора. Прописная версия этой команды (`P`) помещает текст перед курсором. Если вы удаляли одну или более строк, `p` поместит удаленный текст на новую строку (строки) ниже курсора. Если было удалено меньше, чем строка, эта команда поместит удаленный текст в текущую строку после курсора.

Предположим, в файле `practice` есть текст:

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

и вы хотите переместить вторую строку *like a "cut and paste,"* под третьей. С помощью удаления можно проделать следующие исправления:

| Клавиши | Результат |
|---|---|
| `dd` | `You can move text by deleting it and then,`<br>`placing the deleted text elsewhere in the file.`<br>`each time you delete a text block.`<br><br>Поместив курсор во второй строке, удалим ее. Текст перейдет в буфер (в зарезервированную память). |
| `p` | `You can move text by deleting it and then,`<br>`placing that deleted text elsewhere in the file.`<br>`like a "cut and paste"`<br>`each time you delete a text block.`<br><br>Введем команду вставки `p`, чтобы восстановить удаленную строку под текущей. Чтобы закончить перестановки в этом предложении, поменяйте регистр букв и пунктуацию (с помощью `r`), оформив полученный текст грамотно. |

При удалении текста следует восстановить его до вызова новых команд изменения или удаления. Если вы сделали новые правки, повлиявшие на буфер, удаленный ранее текст будет потерян. Таким образом, команду вставки можно повторять снова и снова до новой правки. В главе 4 вы научитесь сохранять удаленный текст в именованном буфере, чтобы к нему можно было обратиться позже.

## Перестановка двух букв

Команда `xp` (удаление символа и его вставка после курсора) позволяет менять местами два символа. Например, в слове *mvoe* буквы *vo* стоят

в неверном порядке. Чтобы это исправить, расположите курсор на букве *v* и нажмите сначала `x`, а потом `p`.

Команды для перестановки слов нет. В разделе «Другие примеры отображения клавиш» на стр. 129 рассматриваются короткие последовательности команд, меняющие слова местами.

## Копирование текста

Часто можно сэкономить время редактирования (и число нажатий на клавиши), если скопировать части файла в нужные места. С помощью двух команд `y` (от *yank* – копировать) и `p` (от *put* – вставить) можно скопировать текст любого размера и поместить его в любом месте файла. Команда `y` копирует выбранный текст в специальный буфер, где он хранится до вызова новой команды копирования (или удаления). Затем командой вставки можно поместить эту копию в любом месте файла.

Как и команды изменения и удаления, `y` может сочетаться с любой из команд перемещения (`yw`, `y$`, `4yy`). Чаще всего команду копирования используют для строки (или нескольких строк), так как копирование и вставка отдельного слова обычно занимает больше времени, чем просто вставка слова «с нуля».

Сокращение `yy` действует на всю строку аналогично командам `dd` и `cc`. Однако действие команды `Y` отличается от `D` или `C`. Вместо копирования с текущей позиции до конца строки она скопирует всю строку. То есть `Y` делает то же, что и `yy`.

Предположим, в файле `practice` содержится текст:

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

Вы хотите сделать три отдельных предложения, каждое из которых начинается с *With a screen editor you can*. Вместо того чтобы перемещаться по строкам и каждый раз вводить эти слова, можно использовать копирование и вставку, чтобы размножить добавляемый текст:

| Клавиши | Результат |
|---|---|
| `yy` | `With a screen editor you can`<br>`scroll the page.`<br>`move the cursor.`<br>`delete lines.`<br><br>Скопируйте строку текста, которую вы хотите поместить в буфер. Курсор может находиться в любом месте этой строки (или в первой строке, если копируется несколько строк). |

| Клавиши | Результат |
|---|---|
| 2j | `With a screen editor you can`<br>`scroll the page.`<br>`move the cursor.`<br>`delete lines.`<br><br>Переместите курсор на то место, куда хотите вставить скопированный текст. |
| P | `With a screen editor you can`<br>`scroll the page.`<br>`With a screen editor you can`<br>`move the cursor.`<br>`delete lines.`<br><br>С помощью P поместите скопированный текст выше строки с курсором. |
| jp | `With a screen editor you can`<br>`scroll the page.`<br>`With a screen editor you can`<br>`move the cursor.`<br>`With a screen editor you can`<br>`delete lines.`<br><br>Переместите курсор на строку ниже и вставьте скопированный текст ниже текущей строки, нажав p. |

Команда копирования использует тот же буфер, что и команда удаления. Каждое новое удаление или копирование заменяет предыдущее содержимое этого буфера. Как мы увидим в главе 4, команды вставки могут использовать до девяти предыдущих копирований или удалений. Также можно копировать или удалять текст непосредственно в один из 26 именованных буферов, что позволит одновременно управлять несколькими текстовыми блоками.

## Повторение или отмена последней команды

Каждая введенная вами команда редактирования хранится в буфере до вызова следующей команды. Например, если вы вставляете *the* после слова, то команда, используемая для вставки текста, временно сохраняется наравне с самим введенным текстом.

### Повторение

Если нужно повторить одну и ту же команду редактирования несколько раз, то можно сэкономить время, продублировав ее при помощи команды повторения – точки (.). Поместите курсор туда, где вы хотите повторить команду, и нажмите точку.

Предположим, в вашем файле есть такие строки:

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

Можно удалить одну строку и ввести точку, чтобы удалить вторую:

| Клавиши | Результат |
|---|---|
| dd | `With a screen editor you can`<br>`scroll the page.`<br>`move the cursor.`<br>Удаление строки командой dd. |
| . | `With a screen editor you can`<br>`scroll the page.`<br>Повторение удаления. |

В старых версиях vi при повторении команд возникали проблемы. Например, можно было столкнуться с трудностями при вставке большого куска текста, если ранее был задан wrapmargin. Если у вас старая версия vi, то рано или поздно эта ошибка даст о себе знать. Особо с этим ничего не поделаешь, но следует предупредить вас об этом (в современных версиях такой проблемы, кажется, нет). Есть два способа защиты от потенциальных проблем при повторении больших вставок. Перед повторением можно сохранить файл (:w), чтобы в случае ошибки вернуться к сохраненной версии. Также можно отключить wrapmargin следующим образом:

```
:set wm=0
```

В разделе «Другие примеры отображения клавиш» на стр. 129 мы покажем вам простой способ использования решения с отключением wrapmargin. В некоторых версиях vi команда CTRL-@ повторяет последнюю вставку. Она вызывается в режиме вставки и возвращает пользователя в командный режим.

## Отмена

Как упоминалось ранее, в случае ошибки можно отменить последнюю команду. Для этого нажмите u. При этом курсор должен стоять на той строке, где было сделано исходное изменение.

Продолжим предыдущий пример, где показывалось удаление строк в файле practice:

| Клавиши | Результат |
|---|---|
| u | ```<br>With a screen editor you can<br>scroll the page.<br>move the cursor.<br>```<br>u отменяет последнюю команду и восстанавливает удаленную строку. |

U, прописная версия u, отменяет все правки, которые проделывались в строке, *пока курсор оставался на ней*. Покинув эту строку, вы уже не сможете использовать U.

Обратите внимание, что с помощью u вы можете отменять последнюю отмену, тем самым переключаясь между двумя версиями текста. u также отменяет U, которая, в свою очередь, отменяет всякие изменения целиком в строке, в том числе и те, что были сделаны командой u.

Совет: тот факт, что u отменяет саму себя, открывает необычный способ перемещения по файлу. Если вам потребовалось попасть на место последней правки, просто отмените ее. После этого вы попадете на нужную строку. Когда вы отмените отмену, вы останетесь на этой строке.

Vim позволяет «повторить» отмененную операцию при помощи CTRL-R. Если сочетать ее с бесчисленными отменами, вы сможете перемещаться вперед и назад по истории правок файла. За большей информацией обратитесь к разделу «Отмена отмен» на стр. 332.

# Другие способы вставки текста

Ранее вы вставляли текст перед курсором при помощи следующей последовательности:

i *текст для вставки* ESC

Команда a позволяет вставлять текст после курсора. Приведем другие команды для вставки текста в разные позиции по отношению к курсору:

A

Приписывает текст к концу текущей строки.

I

Вставляет текст в начало строки.

o (строчная буква «o»)

Создает пустую строку ниже курсора, переводит туда курсор.

O (прописная буква «o»)

Создает пустую строку выше курсора, переводит туда курсор.

s
: Удаляет символ на позиции курсора и подставляет текст.

S
: Удаляет строку и подставляет текст.

R
: Замещает существующие символы новыми.

Все эти команды переводят пользователя в режим вставки. Не забывайте нажимать на ESC после ввода текста, чтобы перейти в командный режим.

Команды A (*append*) и I (*insert*) избавляют от необходимости перемещаться в начало или конец строки перед переключением в режим вставки. (Команда A эквивалентна $a. Нажатие одной клавиши не выглядит большой экономией, однако чем более искушенным – и нетерпеливым – редактором вы станете, тем больше будете лениться нажимать лишние клавиши.)

o и O (*open*) избавляют от лишнего нажатия переноса строки. Эти команды можно вводить, находясь в любом месте строки.

s и S (*substitute*) позволяют удалить символ или строку целиком и заменить удаленный фрагмент любым количеством нового текста. s эквивалентна двухклавишной команде c SPACE, а S – это то же, что и cc. Самое полезное применение s – замена одного символа несколькими.

R («большая», *replace*) полезна, когда вы хотите заменить текст, но не знаете точно, сколько придется менять. Например, чтобы не гадать, что ввести: 3cw или 4cw, просто введите R, а после нее – текст вашей замены.

## Числовые аргументы у команд вставки

За исключением o и O, все перечисленные команды вставки (а также i и a) могут сопровождаться числовыми префиксами. С их помощью вы сможете использовать команды i, I, a и A, чтобы вставить ряд подчеркиваний или чередующиеся символы. Например, ввод 50i* ESC вставит 50 звездочек, а ввод 25a*- ESC добавит 50 символов (25 пар звездочек и дефисов). Следует заметить, что лучше дублировать небольшие строки[1].

Команда r, снабженная числовым аргументом, заменит соответствующее количество символов повторяющимися экземплярами одного. Например, если в программе C или C++ вам нужно заменить || на &&, то можно поставить курсор на первую вертикальную черту и ввести 2r&.

Чтобы заменить несколько строк, следует приписать к команде S соответствующее число. Однако если использовать c с командой перемещения, то получится удобнее и быстрее.

[1] В очень старых версиях vi имелись сложности со вставкой более чем одной строки текста.

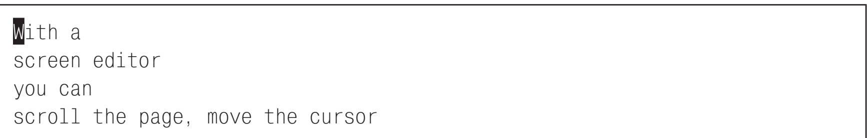
Хороший пример использования команды `s` с числовым префиксом – замена нескольких букв в середине слова. Команда `r` здесь не годится, а `cw` изменит слишком много лишнего. Применение `s` с числом обычно действует аналогично `R`.

Существуют и другие сочетания команд, которые хорошо работают вместе. Например, `ea` пригодится при добавлении текста к концу слова. Рекомендуем потренироваться в применении подобных комбинаций и довести их использование до автоматизма.

## Объединение двух строк с помощью J

Иногда в процессе редактирования может появиться серия коротких строк, с которыми сложно работать. Если вы хотите объединить две строки в одну, поместите курсор в любое место первой строки и нажмите `J`.

Пусть в файле `practice` есть такие строки:




| **Клавиши** | **Результат** |
|---|---|
| `J` | `With a screen editor`<br>`you can`<br>`scroll the page, move the cursor`<br><br>`J` объединяет строку, на которой стоит курсор, со следующей строкой. |
| `.` | `With a screen editor you can`<br>`scroll the page, move the cursor`<br><br>С помощью `.` (символа точки) повторите предыдущую команду (`J`), и следующая строка присоединится к текущей. |

Числовой аргумент у команды `J` объединит несколько строк подряд в одну. В нашем примере три строки можно было объединить командой `3J`.

### Возможные проблемы

- *При вводе команды текст скачет по экрану, и ничего не работает как надо.*

  Убедитесь, что вы вводите `j`, а не `J`.

  Возможно, вы нечаянно нажали клавишу CAPS LOCK. `vi` чувствителен к регистру, то есть прописные команды (`I`, `A`, `J`, и т. д.) отлича-

ются от строчных (i, a, j). Если CAPS LOCK нажата, все ваши команды рассматриваются не как строчные, а как прописные. Нажмите клавишу CAPS LOCK еще раз для возврата в строчный режим, затем ESC для перехода в командный режим. После этого введите либо U, чтобы восстановить последнюю измененную строку, либо u – для отмены последней команды. Возможно, придется сделать дополнительные правки, чтобы восстановить искаженную часть файла.

## Обзор основных команд vi

Таблица 2.1 представляет несколько команд, которые можно вызывать, сочетая команды c, d и y с различными текстовыми объектами. В последних двух строчках показаны новые команды редактирования. В табл. 2.2 и 2.3 перечислены еще некоторые основные команды, а в табл. 2.4 сведены остальные команды, описанные в этой главе.

*Таблица 2.1. Команды редактирования*

| Текстовый объект | Изменение | Удаление | Копирование |
|---|---|---|---|
| Одно слово | cw | dw | yw |
| Два слова без учета пунктуации | 2cW или c2W | 2dW или d2W | 2yW или y2W |
| На три слова назад | 3cb или c3b | 3db или d3b | 3yb или y3b |
| Одна строка | cc | dd | yy или Y |
| До конца строки | c$ или C | d$ или D | y$ |
| До начала строки | c0 | d0 | y0 |
| Один символ | r | x или X | yl или yh |
| Пять символов | 5s | 5x | 5yl |

*Таблица 2.2. Перемещение*

| Перемещение | Команды |
|---|---|
| ←, ↓, ↑, → | h, j, k, l |
| На первый символ следующей строки | + |
| На первый символ предыдущей строки | - |
| В конец слова | e или E |
| Вперед на одно слово | w или W |
| Назад на одно слово | b или B |
| В конец строки | $ |
| В начало строки | 0 |

*Таблица 2.3. Другие операции*

| **Операция** | **Команды** |
|---|---|
| Поместить текст из буфера | `P` **или** `p` |
| Запустить `vi`, открыть файл, если таковой указан | `vi` *`file`* |
| Сохранить изменения, выйти | `ZZ` |
| Выйти, не сохраняя изменений | `:q!` |

*Таблица 2.4. Команды создания и изменения текста*

| **Действие редактирования** | **Команда** |
|---|---|
| Вставить текст в текущей позиции | `i` |
| Вставить текст в начале строки | `I` |
| Добавить текст в текущей позиции | `a` |
| Добавить текст в начале строки | `A` |
| Создать новую строку ниже курсора | `o` |
| Создать новую строку выше курсора | `O` |
| Удалить строку и подставить текст | `S` |
| Замещать существующие символы | `R` |
| Объединить текущую строку со следующей | `J` |
| Поменять регистр | `~` |
| Повторить последнее действие | `.` |
| Отменить последнее изменение | `u` |
| Вернуть строку в первоначальное состояние | `U` |

Вы можете начать работать с `vi`, пользуясь только командами из этих таблиц. Однако чтобы обуздать всю мощь `vi` (и повысить свою производительность), потребуется больше инструментов. О них рассказывается в следующих главах.

# 3

# Быстрое перемещение

Вряд ли вам потребуется vi только для создания новых файлов, ведь большую часть работы составляет редактирование существующих. При этом редко бывает нужно, открыв файл, перемещаться по нему строка за строкой – скорее всего, нужно попасть в определенное место этого файла и начать работу.

При редактировании все начинается с перемещения курсора в нужный фрагмент (или, в случае строкового редактора ex, указания номера редактируемой строки). В этой главе показано, как осуществить перемещение несколькими способами (по экранам, по тексту, по шаблону или по номерам строк). В vi существует множество способов перемещения по файлу. При этом скорость редактирования напрямую зависит от возможности достичь места назначения нажатием всего нескольких клавиш.

В этой главе рассказывается о перемещении:

- по экранам,
- по текстовым блокам,
- путем поиска по шаблону,
- по номеру строки.

## Перемещение по экранам

При чтении книги ваше «местоположение» в тексте определяется страницей, на которой было закончено чтение, или номером страницы в оглавлении. При редактировании такое удобство пропадает. Некоторые файлы занимают всего несколько строк, поэтому весь текст можно увидеть сразу. Но во многих файлах содержатся сотни (и тысячи!) строк.

Файл можно рассматривать как длинный рулон бумаги. Экран – это окно, содержащее (как правило) 24 строки текста этого огромного рулона.

В режиме вставки по мере наполнения экрана текстом вы дойдете до самой нижней строки экрана. По ее достижении и после нажатия на ENTER верхняя строка скроется, а внизу появится пустая для нового текста. Это называется *прокруткой* (*scrolling*).

В командном режиме можно перемещаться по файлу путем прокрутки экрана вперед или назад. А поскольку команды перемещения курсора можно снабжать числовыми префиксами, вы сможете быстро попасть в любое место файла.

## Прокрутка экрана

Ниже приведены команды `vi`, которые перемещают курсор вперед и назад по файлу на полный экран или на полэкрана.

`^F`

Прокрутить вперед на один экран.

`^B`

Прокрутить назад на один экран.

`^D`

Прокрутить вперед (вниз) на полэкрана.

`^U`

Прокрутить назад (вверх) на полэкрана.

(В этом списке символ ^ обозначает клавишу CTRL. Так, `^F` означает, что нужно удерживать клавишу CTRL и при этом нажать f.)

Также есть команды прокрутки на одну строку вверх (`^E`) и вниз (`^Y`). Однако эти команды не перемещают курсор на начало строки. Он остается на том же месте, где и был до применения команды.

## Смена позиции экрана с помощью z

Если вы хотите прокрутить на экран вверх или вниз без изменения положения курсора, воспользуйтесь командой `z`.

`z`*ENTER*

Переместить текущую строку на самый верх экрана и прокрутить.

`z.`

Переместить текущую строку в центр экрана и прокрутить.

`z-`

Переместить текущую строку в самый низ экрана и прокрутить.

В случае команды `z` использование числового префикса бессмысленно. (Кроме того, перемещение курсора наверх экрана нужно только один раз. Дальнейшее применение команды `z` не даст никакого эффекта.) Команда `z` воспринимает числовой префикс в качестве номера строки, которая будет взята вместо текущей. Например `z`ENTER переместит теку-

щую строку наверх экрана, а 200zENTER проделает это действие со строкой под номером 200.

## Перерисовка экрана

Иногда во время редактирования на экране появляются системные сообщения. Они не изменяют содержимого редактируемого файла, но мешают работе. При их возникновении следует обновить или перерисовать экран.

При каждой прокрутке вы перерисовываете часть экрана (или весь экран), поэтому всегда можно избавиться от нежелательных сообщений, прокрутив экран и вернувшись обратно. Кроме того, сочетание CTRL-L перерисовывает экран без прокрутки.

## Перемещение внутри экрана

Можно не менять текущий экран (то есть представление файла), а перемещаться только внутри него:

`H`
: Переход в начало (наверх) экрана.

`M`
: Переход в середину экрана.

`L`
: Переход на последнюю строку экрана.

*n*`H`
: Переход на *n* строк ниже самой верхней строки.

*n*`L`
: Перейти на *n* строк выше самой нижней строки.

`H` перемещает курсор из любого место экрана на первую (или «home») строку. `M` перемещает в середину экрана, `L` – на последнюю строку. Чтобы оказаться в строке, расположенной под первой, введите `2H`.

| Клавиши | Результат |
|---|---|
| L | `With a screen editor you can`<br>`scroll the page, move the cursor,`<br>`delete lines, insert characters, and more,`<br>`while seeing the results of your`<br>`edits as you make them.`<br>`Screen editors are very popular,`<br>`since they allow you to make changes`<br>`as you read through a file.`<br><br>Перейдите на последнюю строку экрана с помощью команды `L`. |

| Клавиши | Результат |
|---|---|
| 2H | With a screen editor you can<br>scroll the page, move the cursor,<br>delete lines, insert characters, and more,<br>while seeing the results of your<br>edits as you make them.<br>Screen editors are very popular,<br>since they allow you to make changes<br>as you read through a file.<br><br>Перейдите на вторую строку экрана, используя команду 2H. (Одна H переместит вас на первую строку экрана.) |

## Перемещение по строкам

Оставаясь внутри текущего экрана, можно пользоваться командами перемещения по строкам. Нам уже знакомы j и k. Существуют и другие команды:

*ENTER*

Перейти на первый символ следующей строки.

+

Перейти на первый символ следующей строки.

-

Перейти на первый символ предыдущей строки.

Эти три команды перемещают вниз или вверх на первый *непустой* символ строки, игнорируя пробелы и табуляции. Отличие от j и k состоит в том, что последние перемещают курсор вниз или вверх на первую позицию строки, даже если она пустая (при условии, что до перемещения курсор также стоял на первой позиции).

### Перемещение по текущей строке

Не забывайте, что h и l перемещают курсор влево и вправо, а 0 (ноль) и $ – в начало и конец строки соответственно. Также можно применять:

^

Переход на первый непустой символ в текущей строке.

*n*|

Переход на *n*-й столбец в текущей строке.

Как и в случае описанных выше команд перемещения по строкам, ^ перемещает на первый символ строки, игнорируя пробелы и табуляции. 0, напротив, переводит курсор на первую позицию в строке, даже если она пустая.

# Перемещение по текстовым блокам

Другой способ движения по файлу в vi – перемещение по текстовым блокам, таким как слова, предложения, абзацы и разделы. Вы уже знаете, как перемещаться по словам (w, W, b или B). Вдобавок, можно пользоваться следующими командами:

e

Перемещение в конец слова.

E

Перемещение в конец слова (игнорируя пунктуацию).

(

Перемещение в начало текущего предложения.

)

Перемещение в начало следующего предложения.

{

Перемещение в начало текущего абзаца.

}

Перемещение в начало следующего абзаца.

[[

Перемещение в начало текущего раздела.

]]

Перемещение в начало следующего раздела.

При поиске конца предложения vi ищет один из следующих знаков препинания: ?, . или !. Конец предложения найден, когда после одного из этих знаков стоят хотя бы два пробела либо когда это последний непустой символ в строке. Если после точки стоит только один пробел или предложение заканчивается кавычками, то vi предложение не распознает.

Абзац определяется как текст до следующей пустой строки или до одного из макросов абзаца (.IP, .PP, .LP, .QP) из пакета макросов MS системы troff. Аналогично раздел – это текст до макроса раздела (.NH, .SH, .H 1 или .HU). Макросы, задающие абзацы и разделы, можно переопределить командой :set, как описано в главе 7.

Помните, что команды перемещения можно сочетать с числами. Например, 3) передвинет вас вперед на три предложения. Также не забывайте, что эти команды можно использовать при редактировании: d) удалит символы до конца текущего предложения, а 2y} скопирует два следующих абзаца.

# Перемещение по результатам поиска

Один из наиболее полезных способов перемещения по большому файлу заключается в поиске отрывка текста, или, правильнее, символьного *шаблона* (*pattern*). Например, может потребоваться найти слово с опечаткой или все вхождения переменной в программу.

Команда поиска – это специальный символ / (косая черта). При вводе она появляется в нижней строке экрана. После этого наберите *шаблон*, по которому будет вестись поиск: /шаблон.

Шаблоном может выступать все слово или любая другая последовательность символов (она называется «строка символов»). Например, если вы ищете символы *red*, вы найдете как это слово целиком, так и слово occur*red*. Если в *шаблон* внести пробелы слева или справа, то они будут рассматриваться как часть слова. Как и для всех команд, вводимых внизу экрана, для завершения набора строки символов нажмите ENTER. vi, как и многие другие редакторы в UNIX, имеет собственный язык поиска по шаблону, что позволяет находить более сложные вещи, например любое слово, начинающееся с прописной буквы, или слово *The* в начале строки.

Мы остановимся на синтаксисе управления поиском по шаблону в главе 6. Пока же воспринимайте *шаблон* просто как слово или фрагмент фразы.

vi начинает поиск с позиции курсора и идет вперед, а по достижении конца файла переходит в его начало. При этом курсор перейдет на первое вхождение шаблона. Если соответствия не найдены, то в строке состояния появится сообщение «Pattern not found» (шаблон не найден)[1]. В файле practice мы переместим курсор с помощью поиска:

| Клавиши | Результат |
|---|---|
| /edits | <pre>With a screen editor you can scroll the<br>page, move the cursor, delete lines, insert<br>characters, and more, while seeing the<br>results of your edits as you make them.</pre> Поиск по шаблону *edits*. Нажмите ENTER для старта. Курсор перейдет на первое вхождение шаблона в тексте. |
| /scr | <pre>With a screen editor you can scroll the<br>page, move the cursor, delete lines, insert<br>characters, and more, while seeing the<br>results of your edits as you make them.</pre> Поиск по шаблону *scr*. Нажмите ENTER для старта. Обратите внимание, что пробела после *scr* нет. |

[1] Текст сообщения будет разным у различных модификаций vi, но его значение везде одно. Не обращайте внимание на подобные изменения, так как смысл передаваемой информации не меняется.

Поиск циклически дойдет до начала файла. Обратите внимание, что можно задавать любую последовательность символов, а не только слово целиком.

Для поиска в обратном направлении введите ? вместо /:

```
?pattern
```

В обоих случаях при необходимости можно продолжить поиск с начала или конца файла.

## Повторный поиск

Последний использовавшийся для поиска шаблон сохраняется на протяжении всего сеанса редактирования. После поиска можно не набирать заново команду, а использовать возобновление поиска по последнему шаблону:

n

Повторить поиск в том же направлении.

N

Повторить поиск в противоположном направлении.

/*ENTER*

Повторить поиск вперед.

?*ENTER*

Повторить поиск назад.

Поскольку последний шаблон не теряется, можно произвести поиск, потом проделать какую-либо работу, после чего снова выполнить поиск по тому же шаблону, используя только n, N, / или ? и не набирая команду поиска заново. Направление поиска (/ – вперед, ? – назад) отображается в левом нижнем углу экрана. (nvi не показывает направление для команд n и N. Vim помещает в командную строку также текст поиска, а сохраненную историю команд поиска позволяет пролистать курсорными клавишами.)

Поскольку шаблон *scr* доступен для поиска, то для продолжения предыдущего примера можно проделать следующее:

| **Клавиши** | **Результат** |
|---|---|
| n | With a screen editor you can scroll the<br>page, move the cursor, delete lines, insert<br>characters, and more, while seeing the<br>results of your edits as you make them.<br><br>Перейдите к следующему вхождению шаблона *scr* (от *scr*een к *scr*oll ), используя команду n (next). |

| Клавиши | Результат |
|---|---|
| ?you | `With a screen editor you can scroll the page, move the cursor, delete lines, insert characters, and more, while seeing the results of your edits as you make them.`<br>С помощью команды ? выполните обратный поиск от позиции курсора до первого вхождения *you*. После ввода шаблона нажмите ENTER. |
| N | `With a screen editor you can scroll the page, move the cursor, delete lines, insert characters, and more, while seeing the results of your edits as you make them.`<br>Повторите предыдущий поиск слова *you*, но уже в обратном направлении (вперед). |

Иногда требуется найти слово, расположенное ниже по тексту, то есть не нужно выполнять циклический поиск и переходить на начало файла. Опция vi wrapscan управляет цикличностью поиска. Ее можно отключить, если ввести:

```
:set nowrapscan
```

Если опция nowrapscan задана и поиск вперед не дал результата, то в строке состояния отображается сообщение:

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

Если опция nowrapscan задана и поиск назад не дал результата, то в этом сообщении появится «TOP» вместо «BOTTOM».

## Редактирование во время поиска

Вы можете сочетать операции поиска / и ? с командами изменения текста, например c и d. Продолжим предыдущий пример:

| Клавиши | Результат |
|---|---|
| d?move | `With a screen editor you can scroll the page, your edits as you make them.`<br>Удаление текста от положения курсора вверх до слова *move*. |

Обратите внимание, что удаление происходит посимвольно, строки не удаляются.

В этом разделе приводится лишь базовая информация о поиске по шаблону. В главе 6 вы узнаете намного больше о шаблонах и их использовании для глобальных изменений в файле.

## Поиск в текущей строке

Существуют также «облегченные» версии команд поиска, которые работают только внутри текущей строки. Команда `f`*x* переместит курсор на следующее вхождение символа *x* (вместо *x* может стоять любой символ). Команда `t`*x* переместит курсор на символ, *предшествующий* следующему вхождению *x*. Для повторного поиска в том же направлении используется клавиша ; (точка с запятой).

Ниже представлены команды поиска в строке. Ни одна из них не переведет курсор на другую строку.

`f`*x*

Найти следующее вхождение *x* в строке и переместить на него курсор (*x* – произвольный символ).

`F`*x*

Найти предыдущее вхождение *x* в строке и переместить на него курсор.

`t`*x*

Найти символ перед следующим вхождением *x* в строке и переместить на него курсор.

`T`*x*

Найти символ перед предыдущим вхождением *x* в строке и переместить на него курсор.

`;`

Повторить предыдущую команду поиска в том же направлении.

`,`

Повторить предыдущую команду поиска в противоположном направлении.

Числовой префикс *n* с каждой из этих команд означает *n*-е вхождение. Предположим, в файле `practice` вы редактируете строку:

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

| Клавиши | Результат |
|---|---|
| `fo` | `With a screen editor you can scroll the`<br>Командой `f` найдите первое вхождение *o* в текущей строке. |
| `;` | `With a screen editor you can scroll the`<br>Командой `;` (найти следующее *o*) перейдите к следующему вхождению *o*. |

Команда dfx удаляет все до указанного символа *x* включительно. Она полезна при удалении или копировании части строки и может оказаться удобнее dw в случае, если в строке есть символы или знаки препинания, усложняющие подсчет слов. Команда t работает аналогично f, но она помещает курсор перед символом, который вы ищете. Например, команда ct. может использоваться для изменения текста до конца предложения, оставляя завершающую точку на месте.

## Перемещение по номеру строки

Строки в файле пронумерованы последовательно, так что можно перемещаться по файлу путем указания номера строки.

Номера строк полезны для обозначения начала и конца большого текстового блока. Также они используются программистами, так как компиляторы указывают на номера строк, содержащих ошибки. Наконец, номера строк используются командами ex, с которыми вы познакомитесь в следующих главах.

Чтобы перемещаться по номерам строк, необходимо как-то узнать эти номера. Их можно отобразить на экране с помощью опции :set nu, описанной в главе 7. В vi также можно включить отображение номера текущей строки внизу экрана.

Команда CTRL-G показывает внизу экрана номер текущей строки, полное количество строк в файле и процентную долю номера текущей строки по отношению к общему количеству строк. Например, в файле practice команда CTRL-G выведет следующее:

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

CTRL-G полезна как для отображения номера строки, применяемого в командах, так и для того, чтобы узнать расположение курсора, если вы потерялись в процессе редактирования.

В зависимости от используемой модификации vi вы можете увидеть и дополнительную информацию, например номер текущего столбца и данные о том, менялся ли файл с момента последнего сохранения. Формат сообщения тоже может быть разным.

### Команда G (Go To)

Номера строк могут использоваться для перемещения курсора по файлу. Команда G (Go To) использует номер строки в качестве числового аргумента и перемещает курсор прямо на эту строку.

Например, 44G переместит курсор на начало строки 44. G без номера строки переместит курсор на последнюю строку файла.

Если вы не делали никаких изменений, то ввод двух обратных кавычек (``) вернет вас в первоначальное положение (где была вызвана команда G).

Если пользователь совершил какое-либо редактирование, а затем переместил курсор командой, отличной от G, то команда `` вернет курсор на место последней правки. Если вы вызывали команду поиска (/ или ?), то `` вернет курсор на место начала этого поиска. Пара апострофов ('') работает почти так же, как и пара обратных кавычек, но переносит курсор не на его предыдущее место в строке, а на начало этой строки.

Общее число строк, отображаемое при нажатии CTRL-G, может дать примерное представление о том, на сколько строк следует переместиться. Если вы находитесь в 10-й строке файла из 1000 строк:

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

и хотите начать редактирование где-то около его конца, то примерное расположение вашей цели будет задаваться командой 800G.

Перемещение по номеру строки – хороший способ быстрого перехода в больших файлах.

# Обзор команд перемещения курсора в vi

В табл. 3.1 собраны команды, рассмотренные в этой главе.

*Таблица 3.1. Команды перемещения*

| Перемещение | Команда |
|---|---|
| Пролистать вперед на один экран | ^F |
| Пролистать назад на один экран | ^B |
| Пролистать вперед на полэкрана | ^D |
| Пролистать назад на полэкрана | ^U |
| Пролистать вперед на одну строку | ^E |
| Пролистать назад на одну строку | ^Y |
| Переместить текущую строку наверх экрана и пролистать | z ENTER |
| Переместить текущую строку в центр экрана и пролистать | z. |
| Переместить текущую строку вниз экрана и пролистать | z- |
| Перерисовать экран | ^L |
| Перейти в самую верхнюю строку экрана | H |
| Перейти в среднюю строку экрана | M |
| Перейти в самую нижнюю строку экрана | L |
| Перейти на первый символ следующей строки | ENTER |
| Перейти на первый символ следующей строки | + |
| Перейти на первый символ предыдущей строки | - |
| Перейти на первый непустой символ текущей строки | ^ |

*Таблица 3.1 (продолжение)*

| Перемещение | Команда |
|---|---|
| Перейти на *n*-й столбец текущей строки | *n*\| |
| Перейти в конец слова | e |
| Перейти в конец слова (игнорируя знаки препинания) | E |
| Перейти к началу текущего предложения | ( |
| Перейти к началу следующего предложения | ) |
| Перейти к началу текущего абзаца | { |
| Перейти к началу следующего абзаца | } |
| Перейти к началу текущего раздела | [[ |
| Перейти к началу следующего раздела | ]] |
| Искать ниже по тексту с использованием шаблона | /*pattern* |
| Искать выше по тексту с использованием шаблона | ?*pattern* |
| Повторить последний поиск | n |
| Повторить последний поиск в противоположном направлении | N |
| Повторить последний поиск ниже по тексту | / |
| Повторить последний поиск выше по тексту | ? |
| Перейти на следующее вхождение *x* в текущей строке | f*x* |
| Перейти на предыдущее вхождение *x* в текущей строке | F*x* |
| Перейти на символ перед следующим вхождением *x* в текущей строке | t*x* |
| Перейти на символ после предыдущего вхождения *x* в текущей строке | T*x* |
| Повторить последнюю команду поиска в том же направлении | ; |
| Повторить последнюю команду поиска в противоположном направлении | , |
| Перейти на заданную строку *n* | *n*G |
| Перейти в конец файла | G |
| Вернуться к предыдущей метке или контексту | `` |
| Вернуться на начало строки, содержащей предыдущую метку | '' |
| Показать информацию о текущей строке (не команда перемещения) | ^G |

# 4

# За рамками основ

Вы уже познакомились с основными командами редактирования vi: i, a, c, d и y, и в этой главе ваши знания расширятся. Она охватывает:

- Описание дополнительных средств редактирования, включая обзор общей формы команды.
- Дополнительные способы входа в vi.
- Работу с буферами, содержащими скопированные и удаленные фрагменты.
- Отметку вашего положения в файле.

## Другие сочетания команд

В главе 2 вы изучили команды редактирования c, d и y, а также научились сочетать их с перемещением и числовыми аргументами (например, 2cw или 4dd). В главе 3 мы добавили в вашу копилку большое количество команд перемещения. Хотя возможность совмещения команд редактирования и перемещения для вас уже не нова, табл. 4.1 позволяет увидеть все разнообразие вариантов правки текста.

Обратите внимание, что все последовательности команд в табл. 4.1 отвечают следующему формату:

(*number*)(*command*)(*text object*)

*number* – это необязательный числовой аргумент. Для рассматриваемых случаев *command* – это команды c, d и y, а *text object* – команда перемещения.

Общий вид команд vi обсуждался в главе 2 (их перечень указан в табл. 2.1 и 2.2).

*Таблица 4.1. Новые команды редактирования*

| Изменить | Удалить | Копировать | От курсора до... |
|---|---|---|---|
| `cH` | `dH` | `yH` | верха экрана |
| `cL` | `dL` | `yL` | низа экрана |
| `c+` | `d+` | `y+` | следующей строки |
| `c5\|` | `d5\|` | `y5\|` | 5-го столбца в следующей строке |
| `2c)` | `2d)` | `2y)` | 2-го предложения ниже по тексту |
| `c{` | `d{` | `y{` | предыдущего абзаца |
| `c/`*pattern* | `d/`*pattern* | `y/`*pattern* | шаблона *pattern* |
| `cn` | `dn` | `yn` | следующего вхождения *pattern* |
| `cG` | `dG` | `yG` | конца файла |
| `c13G` | `d13G` | `y13G` | строки номер 13 |

# Варианты запуска vi

До сих пор вы запускали редактор `vi` такой командой:

```
$ vi file
```

У команды `vi` есть и другие полезные опции. Например, можно открыть файл на определенной строке, или с поиском по указанному шаблону, или только для чтения. Еще одна опция восстанавливает изменения в файле, который редактировался в момент сбоя системы.

## Переход на указанную позицию

Можно открыть файл и перейти на первое вхождение шаблона или строку с определенным номером, а также указать исходное перемещение либо поиском, либо по номеру строки прямо в командной строке[1]:

`$ vi +`*`n file`*

Откроет *file* на строке *n*.

`$ vi + `*`file`*

Откроет *file* на последней строке.

`$ vi +/`*`pattern file`*

Откроет *file* на первом вхождении *pattern*.

В файле `practice` для открытия файла и перехода на строку со словом *Screen* введите:

[1] Согласно стандарту POSIX, вместо показанной здесь команды `+`*`command`* `vi` должен использовать `-c` *`command`*. Как правило, в целях обратной совместимости работают обе версии.

| Клавиши | Результат |
|---|---|
| `vi +/Screen practice` | `With a screen editor you can scroll`<br>`the page, move the cursor, delete`<br>`lines, and insert characters, while`<br>`seeing the results of your edits as`<br>`you make them.`<br>`Screen editors are`<br>`very popular, since they allow you`<br>`to make changes as you read`<br><br>Передача опции `+/pattern` команде `vi` и переход к строке со словом *Screen*. |

Как видно в этом примере, шаблон вашего поиска не обязательно окажется в самом верху экрана. Если в шаблоне есть пробелы, то необходимо заключить весь шаблон в одинарные или двойные кавычки[1]:

```
+/"you make"
```

или поставить обратную косую черту перед пробелом:

```
+/you\ make
```

Кроме того, если вы хотите использовать стандартный синтаксис для шаблонов, описанный в главе 6, то, скорее всего, вам придется экранировать специальные символы с применением кавычек или обратной косой черты, чтобы оболочка не интерпретировала их по-своему.

Использование опции `+/pattern` будет полезным, если вам пришлось выйти из сеанса редактирования до окончания работы. В этом случае можете пометить текущее место в файле, вставив специальный шаблон, например `ZZZ` или `HERE`. Повторно вернувшись к файлу, нужно будет лишь ввести `/ZZZ` или `/HERE` для перехода к месту окончания предыдущего сеанса работы.

Как правило, при работе в `vi` опция `wrapscan` включена. Если в ваших настройках опция `wrapscan` всегда выключена (см. «Повторный поиск» на стр. 61), то вы не сможете воспользоваться `+/pattern`. При попытке открытия файла с этой опцией `vi` поместит курсор на последнюю строку и отобразит сообщение «Address search hit BOTTOM without matching pattern».

## Режим «только для чтения»

Бывают случаи, когда нужно просмотреть файл и при этом защитить его от случайных изменений (например, если вы захотите открыть большой файл для тренировки команд перемещения `vi` либо пролистать командный файл или текст программы). Вы можете открыть файл

[1] Наличие кавычек – это требование оболочки, а не `vi`.

в режиме «только для чтения» (read-only), перемещаться по нему с помощью любых команд перемещения, но не иметь возможности внести изменения в файл. Для этого введите либо:

```
$ vi -R file
```

либо

```
$ view file
```

(Команда view, подобно команде vi, может сопровождаться любыми опциями для указания определенного места в тексте[1].) Если вы решите сделать изменения в файле, то можно отменить режим «только для чтения», добавив к команде сохранения восклицательный знак:

```
:w!
```

или:

```
:wq!
```

При возникновении проблем с сохранением файла обратитесь к перечню возможных проблем, приведенному в приложении C.

## Восстановление буфера

Иногда во время редактирования файла может произойти отказ системы. Обычно в этом случае все изменения, сделанные в файле после последней записи (сохранения), теряются. Однако существует опция -r, позволяющая восстановить буфер, который вы редактировали во время системного сбоя.

В традиционной системе UNIX с оригинальным vi при первом входе в систему после сбоя вы получаете почтовое сообщение, где говорится, что буфер сохранен. Кроме того, если ввести команду:

```
$ ex -r
```

или:

```
$ vi -r
```

то можно получить список всех файлов, сохраненных системой.

Для восстановления редактируемого буфера используйте опцию -r вместе с именем файла. Например, чтобы восстановить редактируемый буфер файла practice после системного сбоя, введите:

```
$ vi -r practice
```

Целесообразно восстановить файл сразу же, чтобы не сделать в нем случайных правок, а затем выбирать между сохраненным буфером и только что отредактированным файлом.

Командой :pre (сокращение от :preserve) можно заставить систему сохранять буфер, даже когда никакого сбоя нет. Это может оказаться по-

[1] Как правило, view — это просто ссылка на vi.

лезным, когда вы сделали изменения в файле и обнаружили, что не можете сохранить его из-за отсутствия прав на запись. (В этом случае можно также записать копию файла под другим именем либо в каталог, где у вас есть права на запись. См. «Проблемы при сохранении файлов» на стр. 30.)

В разных версиях и модификациях vi восстановление работает по-разному, поэтому лучше всего обратиться к локальной документации. vile вообще не поддерживает восстановление. Согласно его документации рекомендуется использовать опции autowrite и autosave. О том, как это делается, читайте в разделе «Настройка vi» на стр. 117.

# Использование буферов

Вы уже видели, что во время редактирования последнее удаление (d или x) или копирование (y) сохранялось в буфере[1] (специально отведенной области памяти). Вы можете обращаться к содержимому этого буфера и командой вставки (p или P) помещать сохраненный текст обратно в файл.

vi сохраняет последние девять удалений в нумерованных буферах. К любому из них можно обратиться, чтобы восстановить какое-либо из последних девяти удалений (или все сразу). (Небольшие удаленные фрагменты, составляющие неполную строку, не хранятся в нумерованных буферах. Их можно восстановить только командами p или P сразу после удаления.)

vi также позволяет помещать скопированный командой y текст в буферы под определенными именами. Вы можете задействовать до 26 (a–z) буферов, содержащих скопированный текст, и восстанавливать этот текст командой вставки p в любой момент сеанса редактирования.

## Восстановление удалений

Возможность удалять большие куски текста полезна, но что если вы по ошибке удалили 53 нужных вам строки? Можно восстановить любое из *девяти* последних удалений, так как они сохранены в нумерованных буферах. Последнее хранится в буфере 1, предпоследнее – в буфере 2 и т. д.

Чтобы восстановить удаление, введите " (двойную кавычку), укажите номер буфера для восстановления, а затем введите команду вставки p. Так, для восстановления предпоследнего удаления наберите:

```
"2p
```

Удаленный текст из буфера 2 будет помещен после курсора.

[1] Не путайте его с буфером из предыдущего раздела: здесь речь идет о специально выделенной области памяти для временного хранения (обмена) информации; в предыдущем разделе речь о содержимом файла, открытом в vi и в данный момент находящемся в ОЗУ. – *Прим. науч. ред.*

Если вы не уверены, какой именно буфер содержит нужный текст, необязательно раз за разом вводить "*n*p. Команда повторения (.) в сочетании с p после u автоматически увеличит номер буфера. В результате вы сможете искать в нумерованных буферах, используя:

"1pu.u.u *и т.д.*

чтобы помещать содержимое каждого из последующих буферов в файл один за другим. Каждый раз при вводе u восстановленный текст исчезает, а когда вы набираете точку (.), в файл вставляется содержимое *следующего* буфера. Повторяйте ввод u и ., пока не восстановите нужный фрагмент.

## Копирование в именованный буфер

Вы уже знаете, что вставить содержимое неименованного буфера (командами p или P) можно только до внесения других изменений в файл, иначе этот буфер будет перезаписан. Также можно применить y и d совместно с 26 именованными буферами (от a до z), используемыми специально для копирования и перемещения текста. Если вы присвоили имя буферу, чтобы сохранить скопированный текст, то можно запросить его содержимое в любой момент сеанса редактирования.

Чтобы скопировать текст в именованный буфер, поставьте перед командой копирования y двойную кавычку (") и букву, задающую имя буфера, в который вы хотите скопировать текст. Например:

"dyy *Копировать текущую строку в буфер d.*
"a7yy *Копировать следующие семь строк в буфер a.*

После загрузки текста в именованные буферы и перемещения в другой фрагмент воспользуйтесь p или P, чтобы вставить текст:

"dP *Поместить содержимое буфера d перед курсором.*
"ap *Поместить содержимое буфера a после курсора.*

Поместить часть буфера не получится – его содержимое вставляется только целиком.

В следующей главе вы познакомитесь с редактированием нескольких файлов. Научившись переключаться между файлами без выхода из vi, вы сможете применять именованные буферы для выборочной передачи текста между файлами. А если вы пользуетесь функцией многооконного редактирования в клонах vi, то также сможете использовать для этой цели неименованный буфер.

Также можно удалять текст в именованные буферы. При этом используется похожая конструкция:

"a5dd *Удалить пять строк в буфер a.*

Если написать имя буфера с прописной буквы, то скопированный или удаленный текст будет *присоединен* к текущему содержимому этого бу-

фера. Это дает больше простора в копировании или перемещении. Например:

"zd)

Удалить от курсора до конца текущего предложения и сохранить в буфере z.

2)

Перейти на два предложения ниже.

"Zy)

Добавить новое предложение в буфер z. Вы можете и дальше добавлять текст в именованный буфер, но имейте в виду, что если вы хоть раз забудетесь и скопируете или удалите текст в буфер, указав его имя строчной буквой, то перезапишете буфер и тем самым потеряете все, что было в нем накоплено.

# Отметка места

В vi можно помечать свое положение в файле невидимой «закладкой», чтобы после работы с другими фрагментами текста можно было вернуться на эту метку. В командном режиме введите:

m*x*

Эта команда ставит метку *x* на текущую позицию (*x* может быть любой буквой). Заметим, что первоначальная версия vi допускала использование только строчных букв. В Vim прописные и строчные буквы различаются.

'*x*

(Апостроф). Помещает курсор на первый символ строки с меткой *x*.

`*x*

(Обратная кавычка). Помещает курсор на символ, помеченный как *x*.

``

(Обратные кавычки). Возвращает в точности на предыдущую метку[1] или в контекст.

''

(Апострофы). Возвращает на начало строки, содержащей предыдущую метку или контекст.

Метки устанавливаются только для текущего сеанса vi и не сохраняются в файле.

---

[1] Автор использует термин «предыдущая метка» (previous mark) для обозначения места, в котором находился курсор до выполнения какой-либо команды перехода. – *Прим. науч. ред.*

# Другие продвинутые команды редактирования

Существуют и иные команды продвинутого редактирования, но для их использования вам сначала нужно немного узнать о редакторе ex, о котором рассказано в следующей главе.

# Обзор команд vi для работы с буфером и метками

В табл. 4.2 приведены опции командной строки vi, общие для всех версий редактора. В табл. 4.3 и 4.4 представлены команды работы с буфером и метками.

*Таблица 4.2. Опции командной строки*

| Команда | Значение |
|---|---|
| +*n file* | Открывает файл *file* на строке с номером *n*. |
| + *file* | Открывает файл *file* на последней строке. |
| +/*pattern file* | Открывает файл *file* на первом вхождении шаблона (в POSIX используется -c). |
| -c *command file* | Выполняет команду *command* после открытия файла *file*. Как правило, это переход на определенную строку или поиск (POSIX-версия +). |
| -R | Включает режим «только для чтения» (то же, что и вызов view вместо vi). |
| -r | Восстанавливает файл после сбоя. |

*Таблица 4.3. Имена буферов*

| Имя буфера | Использование буфера |
|---|---|
| 1–9 | Последние девять удалений (чем больше номер, тем старше удаление). |
| a–z | Именованные буферы. Прописные буквы добавляют текст в буфер. |

*Таблица 4.4. Команды работы с буфером и метками*

| Команда | Значение |
|---|---|
| "*b command* | Выполнить команду *command* с буфером *b*. |
| m*x* | Поставить метку *x* на текущую позицию. |
| '*x* | Переместить курсор на первый символ в строке с меткой *x*. |
| `*x* | Переместить курсор на символ с меткой *x*. |
| `` | Возвращает на точную позицию предыдущей метки или контекста. |
| '' | Возвращает на начало строки, содержащей предыдущую метку или контекст. |

# 5

# Введение в редактор ex

Вы спросите: если эта книга рассказывает о vi, то зачем включать в нее главу о другом редакторе? На самом деле, ex не совсем другой редактор. vi – это визуальный режим более общего, базового строкового редактора, которым и является ex. Некоторые его команды могут пригодиться при работе с vi и сэкономить много времени. Многие из них можно использовать, не покидая vi[1].

Вы уже знаете, что файл можно рассматривать как набор пронумерованных строк. ex обеспечивает пользователя более мобильными и мощными командами редактирования. С его помощью вы сможете легко перемещаться между файлами и передавать текст из одного файла в другой множеством способов, а также быстро редактировать текстовые блоки, размер которых больше одного экрана. Кроме того, с помощью глобальной замены можно сделать изменение по заданному шаблону во всем файле.

Эта глава знакомит вас с ex и его командами. Вы узнаете, как:

- перемещаться по файлу, используя номера строк;
- применять команды ex, чтобы копировать, перемещать и удалять текстовые блоки;
- сохранять файлы или их части;
- работать с несколькими файлами (считывать текст или команды, перемещаться между файлами).

---

[1] vile отличается от других модификаций тем, что в нем не работают многие продвинутые команды ex. Подробное описание каждой команды приводится в главе 18.

## Команды ex

Задолго до изобретения vi или какого бы то ни было другого экранного редактора люди общались с компьютером с помощью печатных терминалов, а не через экран дисплея (растровый экран с устройством позиционирования вроде «мыши» и программами эмуляции терминала). Номера строк играли роль указателя, позволявшего знать, над какой частью файла вы работаете. Строковые редакторы были адаптированы для редактирования подобных файлов. Программист или другой пользователь компьютера распечатывал строку (или несколько строк) в терминале печати, вводил команды для редактирования именно этой строки, затем печатал заново и проверял отредактированную строку.

Печатные терминалы уже давно никто не применяет, но некоторые из команд ex до сих пор востребованы пользователями более изощренного редактора, построенного на базе ex. Хотя само редактирование проще выполнять в vi, ориентация ex на строки дает большое преимущество при широкомасштабных изменениях в нескольких частях файла.

Многие из команд, с которыми мы встретимся в этой главе, будут иметь в качестве аргумента имя файла, поэтому не рекомендуется давать файлам имена, содержащие пробелы, хотя это и не запрещено. ex будет сбит с толку, а чтобы объяснить ему, что к чему, потребуется куда больше усилий, нежели следовало бы уделять именованию файлов. Пользуйтесь знаком подчеркивания, тире или точками, чтобы отделять компоненты в именах файлов, и у вас будет гораздо меньше проблем.

Перед тем как вы начнете запоминать команды ex (или, что не рекомендуется, игнорировать их), давайте сначала приоткроем завесу тайны над строковыми редакторами. Если вы увидите, как работает ex при непосредственном вызове, то непонятный синтаксис его команд станет для вас более осмысленным.

Откройте файл practice и попробуйте выполнить некоторые команды ex. Запуск редактора с файлом совершенно аналогичен подобному действию в vi. При вызове ex вы увидите сообщение с информацией о полном количестве строк в файле и приглашение в виде двоеточия. Например:

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

Пока вы не скажете ex отобразить строки, вы не увидите ни одной строки этого файла.

Команды ex состоят из адреса строки (обычно это просто ее номер) и самой команды, заканчивающейся возвратом каретки (нажатием ENTER). Одной из основных команд является p (от print) для печати на экране.

Например, если ввести в командном приглашении 1p, то вы увидите первую строку файла.

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

Фактически можно опустить p, так как номер строки сам по себе эквивалентен команде печати для этой строки. Чтобы вывести более одной строки, укажите диапазон номеров строк (например, 1,3 — два числа, разделенные запятыми, без пробелов между ними). Примерно так:

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

Команда без номера строки оперирует с текущей строкой. Так, команду подстановки (s), позволяющую делать замену одного слова на другое, можно ввести следующим способом:

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

Обратите внимание, что измененная строка вывелась заново после выполнения команды. Те же изменения можно проделать и другим способом:

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

Даже если вы будете пользоваться командами ex из vi и не станете выполнять их непосредственно, все-таки стоит уделить немного времени ex как таковому. Вы увидите, как именно следует указывать редактору строку, с которой вы хотите работать, и какую команду выполнить.

Дав несколько команд ex в файле practice, запустите vi с тем же файлом, чтобы увидеть его в более привычном визуальном режиме. Команда :vi перенесет вас из ex в vi.

Чтобы вызвать команду ex из vi, нужно ввести специальный символ : (двоеточие) для перехода в нижнюю строку. Затем наберите команду и нажмите ENTER, чтобы выполнить ее. Например, в редакторе ex перемещение на строку происходит путем ввода номера этой строки в приглашении с двоеточием. Чтобы переместиться на строку 6 посредством этой команды в vi, введите:

```
:6
```

После этого нажмите ENTER.

После следующего упражнения мы обсудим команды ex с точки зрения их вызова из vi.

## Упражнение: редактор ex

| | |
|---|---|
| В приглашении командной строки UNIX вызовите редактор ex с файлом practice: | ex practice |
| Появится сообщение: | "practice" 6 lines, 320 characters |
| Перейдите на первую строку и напечатайте (выведите): | :1 |
| Напечатайте (выведите) строки с 1 по 3: | :1,3 |
| В строке 1 сделайте замену слова *screen*: | :1s/screen/line |
| Вызовите редактор vi с этим файлом: | :vi |
| Перейдите на первую строку: | :1 |

## Перечень возможных проблем

- *Во время редактирования в vi вы неожиданно попали в редактор ex.*

  Клавиша Q, нажатая в командном режиме vi, вызывает ex. Попав в него, можно всегда вернуться в vi командой vi.

# Редактирование в ex

Многие команды редактирования ex имеют более простые в обращении аналоги в vi. Очевидно, для удаления одного слова или строки вы будете использовать dw или dd, а не команду delete из ex. Однако команды ex удобны, когда нужно произвести изменения в нескольких строках. Они позволяют менять большие текстовые блоки всего одной командой.

Эти команды приведены ниже вместе с их аббревиатурами. Помните, что в vi перед каждой командой ex нужно ставить двоеточие. В зависимости от того, что вам легче запомнить, можете использовать либо саму команду, либо ее сокращение.

| Полное название | Сокращение | Значение |
|---|---|---|
| delete | d | Удаляет строки |
| move | m | Перемещает строки |
| copy | co | Копирует строки |
| | t | Копирует строки (аналог co) |

Для облегчения чтения команды ex можно разделять ее элементы пробелами. Например, подобным образом можно отделять адреса, шаблоны и команды. Тем не менее нельзя использовать пробел как разделитель внутри шаблона или в конце команды подстановки.

## Адреса строк

Каждой команде редактирования ex необходимо сообщить номер строки (или строк), которые будут редактироваться. А командам ex для копирования и перемещения нужно вдобавок сообщить, куда именно вы хотите переместить текст.

Адреса строк можно задать несколькими способами:

- явным указанием номера строки;
- символами, указывающими номера строк относительно текущей позиции в файле;
- шаблонами поиска в качестве *адресов* строк, на которые будет действовать команда.

Рассмотрим несколько примеров.

## Определение диапазона строк

Для явного задания диапазона строк можно использовать их номера. Адреса, использующие явные номера строк, называются абсолютными адресами строк. Например:

:3,18d

Удалить строки с 3 по 18.

:160,224m23

Переместить строки с 160 по 224 сразу после строки 23 (похоже на удаление и вставку в vi).

:23,29co100

Скопировать строки с 23 по 29 и вставить их после строки 100 (аналогично копированию и вставке в vi).

Чтобы облегчить редактирование с помощью номеров строк, можно включить их отображение в левой части экрана. Команда:

```
:set number
```

или ее сокращение:

```
:set nu
```

покажет номера строк. Файл practice при этом будет выглядеть так:

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

При записи файла номера строк не сохраняются на диске и не распечатываются на принтере. Они перестанут отображаться после выхода из vi или отключения опции set:

```
:set nonumber
```

или

```
:set nonu
```

Чтобы временно вывести номера для набора строк, используйте знак #. Например, команда:

```
:1,10#
```

покажет номера строк с 1 по 10.

Как уже рассказывалось в главе 3, команда CTRL-G используется для вывода номера текущей строки. Таким образом, если переместиться на начало блока, нажать CTRL-G, затем перейти в конец блока и нажать CTRL-G еще раз, то можно определить номера строк, соответствующих началу и концу текстового блока.

Еще один способ узнать номер строки – команда ex =:

`:=`

Вывод полного числа строк.

`:.=`

Вывод номера текущей строки.

`:/pattern/=`

Вывести номер следующей строки, которая отвечает шаблону *pattern*.

## Символы адресации строк

Для адресации строк также можно использовать символы. Точка (.) соответствует текущей строке, символ $ обозначает последнюю строку файла, а знак % соответствует всем его строкам. Аналогичное значение имеет сочетание 1,$. Эти символы можно комбинировать с абсолютными номерами строк.

Например,

`:.,$d`

Удаление с текущей строки до конца файла.

`:20,.m$`

Переместить в конец файла блок, содержащий строки с 20-й по текущую.

`:%d`

Удалить все строки в файле.

`:%t$`

Скопировать все строки и поместить их в конец файла (исходное содержимое файла повторится два раза).

Кроме абсолютного номера строки можно указывать адрес по отношению к текущей строке. Символы + и - выполняют соответствующие арифметические операции. Если поставить их перед числом, то оно будет добавлено или вычтено. Например:

`:.,.+20d`

Удалить текущую строку и следующие за ней 20 строк.

`:226,$m.-2`

Поместить строки с 226-й до последней в файле сразу после строки, расположенной на две строки выше текущей.

`:.,+20#`

Отобразить номера строк с текущей до лежащей ниже на 20 строк.

Фактически при использовании + или - точку (.) ставить необязательно, так как текущая строка считается стартовой позицией по умолчанию.

Если после + или - не указать число, то они станут эквивалентными +1 и -1 соответственно[1]. Аналогично ++ и -- расширяют диапазон строк еще на одну строку, и т. д. В следующем разделе будет показано, как использовать + и - с шаблонами поиска.

Номер 0 отвечает самому началу файла (воображаемая нулевая строка) и эквивалентен 1-. Оба элемента позволяют копировать или перемещать текст в начало файла, то есть перед первой строкой, открывающей текст. Например:

`:-,+t0`

Копирование трех строк (от строки выше курсора до строки ниже него) и их вставка в начало файла.

## Шаблоны поиска

ex может обращаться к строкам посредством шаблонов поиска. Например:

:/*pattern*/d

Удалить следующую найденную строку, содержащую шаблон *pattern*.

:/*pattern*/+d

Удалить строку, расположенную *ниже* той, которая содержит *pattern*. (Вместо + можно написать +1.)

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

Удалить от первой строки, содержащей *pattern1*, до первой строки, содержащей *pattern2*.

---

[1] При использовании относительных адресов не нужно отделять плюс и минус от следующих за ними чисел. Например, +10 значит «10 следующих строк», а + 10 означает «11 следующих строк (1+10)», то есть не совсем то, что вам нужно.

:.,/*pattern*/m23

Взять текст от текущей строки (.) до первой, содержащей *pattern*, и переместить его после строки 23.

Обратите внимание, что шаблон ограничен косыми чертами *слева* и *справа*.

Если вы удаляете текст с использованием шаблонов и в vi, и в ex, то знайте, что в работе этих двух редакторов есть отличия[1]. Пусть файл practice содержит следующие строки:

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

| Клавиши | Результат |
|---|---|
| d/while | With a screen editor you can scroll the<br>page, move the cursor, while seeing results<br>of your edits as you make them.<br><br>В vi команда удаления до *pattern* стирает текст от курсора до слова *while*, но не трогает остатки обеих строк. |
| :.,/while/d | With a screen editor you can scroll the<br>of your edits as you make them.<br><br>Команда ex удаляет весь диапазон, заданный этими строками. В нашем случае это текущая строка и строка, содержащая шаблон. |

## Переопределение текущего положения курсора

Иногда использование относительного адреса в команде может привести к непредсказуемым результатам. Предположим, что курсор находится на строке 1 и вы хотите вывести строку 100 и еще пять строк ниже нее. Если набрать:

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

то появится сообщение об ошибке «First address exceeds second» (первый адрес превышает второй). Причина неудачи в том, что второй адрес вычисляется по отношению к текущей позиции курсора (строка 1), так что на самом деле ваша команда эквивалентна следующей:

```
:100,6 p
```

Нужно каким-то образом сделать так, чтобы команда считала строку 100 текущей, несмотря на то, что курсор стоит на первой строке.

[1] ex – строковый редактор, и поэтому оперирует строками как целым, тогда как vi может обрабатывать часть строки. – *Прим. науч. ред.*

В ex есть такая возможность. Если вы используете точку с запятой вместо запятой, то текущим будет считаться адрес первой строки. Например, команда:

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

выведет требуемые строки. Теперь +5 вычисляется по отношению к строке 100. Точка с запятой полезна при использовании как шаблонов поиска, так и абсолютных адресов. Например, чтобы вывести строку, содержащую *pattern,* а также 10 следующих строк, введите команду:

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

## Глобальный поиск

Вы уже знаете, как в vi использовать / (косую черту) для поиска в файле по шаблону. В ex есть команда глобального поиска g (от global), позволяющая искать по шаблону и выводить все строки с этим шаблоном. Команда :g! имеет противоположное действие: она (или ее синоним :v) ищет строки, не содержащие шаблоны.

Команду глобального поиска можно использовать для всех строк в файле либо указывать диапазон строк, ограничивающих применение команды глобального поиска.

:g/*pattern*

Находит (переходит на) последнее вхождение *pattern* в файле.

:g/*pattern*/p

Находит и выводит все строки в файле, содержащие *pattern*.

:g!/*pattern*/nu

Находит и выводит все строки файла (и их номера), которые не содержат *pattern*.

:60,124g/*pattern*/p

Находит и выводит все строки с номерами, лежащими в диапазоне от 60 до 124, содержащие *pattern*.

Как можно догадаться из названия, g может также использоваться для глобальной замены. Об этом мы поговорим в главе 6.

## Сочетания команд ex

Чтобы выполнить новую команду ex, необязательно каждый раз вводить двоеточие. Вертикальная черта (|) служит разделителем команд в ex, что позволяет комбинировать несколько команд в одном приглашении ex (примерно так же, как точка с запятой разделяет несколько команд в одном приглашении командной строки оболочки UNIX). При использовании | нужно следить за указываемыми адресами строк. Если одна из команд меняет порядок строк в файле, то следующая будет работать уже с новым расположением строк. Например:

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

Удаляет строки с 1-й по 3-ю (после этого вы оказываетесь на верхней строке файла), а затем производит замену в текущей строке (которая имела номер 4 перед вызовом команды `ex`).

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

Перемещает строки с 1-й по 5-ю на позицию после строки 10, а затем отображает все строки (с номерами), содержащие *pattern*.

Обратите внимание, что здесь используются пробелы, чтобы облегчить чтение команд.

## Сохранение и выход

Вы уже знаете команду `vi ZZ`, с помощью которой осуществляется выход из программы с сохранением (записью) файла. Однако часто оказывается удобнее выходить из файла с помощью команд `ex`, так как они обеспечивают больший контроль. По ходу изложения мы уже упоминали некоторые из них. Теперь предпримем более формальный подход:

`:w`

Записывает (сохраняет) буфер в файл, но не выходит из программы. Вы можете (и должны) использовать `:w` во время сеанса редактирования, чтобы защитить правки от системных сбоев и серьезных ошибок при редактировании.

`:q`

Выходит из редактора (и возвращает в приглашение командной строки UNIX).

`:wq`

Выполняет два действия: записывает файл и выходит из редактора. Производится безусловная запись, даже если файл не был изменен.

`:x`

Записывает файл и выходит из редактора. Файл записывается только в том случае, если в нем произошли изменения[1].

`vi` защищает и существующие файлы, и сделанные правки в буфере. Например, при попытке записать содержимое буфера в существующий файл `vi` выдаст предупреждение. Аналогично, если вы вызвали `vi` с файлом, сделали изменения и хотите выйти *без сохранения*, редактор выдаст сообщение об ошибке примерно следующего вида:

`No write since last change.` *(последние изменения не записаны)*

---

[1] Различие в `:wq` и `:x` важно при редактировании исходного кода и использовании `make`, действия которого зависят от времени изменения файлов.

Эти предупреждения могут предотвратить существенные ошибки, но иногда вам действительно нужно выполнить такую команду. Восклицательный знак (!) после команды отменяет все предупреждения:

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

:w! также можно использовать для сохранения файла, открытого в режиме «только для чтения» командами vi -R или view (предполагается, что у пользователя есть право на запись файла в файловой системе).

:q! — это самая важная команда редактирования, позволяющая выходить из программы, не затрагивая первоначальный файл независимо от изменений, сделанных во время сеанса. Содержимое буфера при этом будет потеряно.

## Переименование буфера

Команда :w может также использоваться для сохранения всего буфера (или копии редактируемого вами файла) под другим именем.

Пусть есть файл practice, содержащий 600 строк. Вы открываете файл, делаете множество правок, затем хотите выйти, но при этом сохранить как старую версию practice, так и исправленную, чтобы их можно было сравнивать. Чтобы сохранить отредактированный буфер в файле под названием practice.new, выполните команду:

```
:w practice.new
```

Старая версия, расположенная в файле practice, остается неизменной (если только вы не выполняли :w). Теперь командой :q можно выйти из редактирования новой версии.

## Сохранение фрагмента файла

При редактировании файла часто может понадобиться сохранить его часть в отдельном файле. К примеру, так можно поступать с кодами форматирования и текстом, которые будут использоваться в других файлах в качестве заголовков.

Для сохранения части файла можно использовать комбинацию команды адресации строк с командой записи w. Например, если вы находитесь в файле practice и хотите сохранить его часть под именем newfile, можно ввести:

:230,$w *newfile*

Сохраняет от 230-й строки до конца файла в файл newfile.

:.,600w *newfile*

Сохраняет строки с текущей до 600-й в newfile.

## Добавление к сохраненному файлу

Вместе с командой w можно использовать операцию перенаправления UNIX с добавлением (>>). Тогда вы добавите все содержимое буфера или его часть к существующему файлу. Например, если ввести:

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

а затем

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

то *newfile* будет содержать строки с 1 по 10, а также со строки 340 до конца буфера.

# Копирование одного файла в другой

Иногда приходится копировать уже имеющиеся в системе текст или данные в редактируемый файл. В vi можно считать содержимое другого файла с помощью команды ex:

```
:read filename
```

или ее аббревиатурой:

```
:r filename
```

Эта команда вставляет содержимое *filename*, начиная со строки после положения курсора в файле. Если нужно указать строку, отличную от той, где находится курсор, просто введите ее номер (или любой другой адрес строки) перед командой read или r.

Предположим, вы редактируете файл practice и хотите считать файл под названием data из другого каталога, например /home/tim. Переместите курсор на одну строку выше той, куда вы хотите вставить новые данные, и введите:

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

Все содержимое файла /home/tim/data будет считано в practice, начиная со строки ниже курсора.

Чтобы считать тот же файл и поместить его после строки 185, введите:

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

Есть и другие способы считать файл:

:$r /home/tim/data

Помещает считываемый файл в конец текущего.

:0r /home/tim/data

Помещает считываемый файл в самое начало текущего.

:/*pattern*/r /home/tim/data

Помещает считываемый файл в текущий после строки, содержащей *pattern*.

# Редактирование нескольких файлов

Команды ex позволяют переключаться между несколькими файлами. Преимуществом редактирования сразу нескольких файлов является скорость. При использовании системы совместно с другими пользователями выход из vi и его повторный запуск для следующего файла займут определенное время. Когда вы остаетесь внутри одного сеанса редактирования, вы выигрываете не только в скорости переключения между файлами. Таким образом вы сохраните ранее определенные сокращения и последовательности команд (см. главу 7), а буферы копирования (yank) позволят копировать текст из одного файла в другой.

## Вызов vi с несколькими файлами

При первом запуске vi можно указывать более одного файла для редактирования, а впоследствии пользоваться командами ex для переключения между ними. Например:

```
$ vi file1 file2
```

Вначале будет редактироваться file1. После окончания редактирования первого файла командой ex :w запишите (сохраните) file1, а командой :n вызовите следующий файл (file2).

Пусть нужно отредактировать два файла: practice и note.

| Клавиши | Результат |
|---|---|
| vi practice note | `With a screen editor you can scroll`<br>`the page, move the cursor, delete lines,`<br>`insert characters, and more, while seeing`<br><br>Открываются два файла, practice и note. Сначала на экране появится указанный первым файл practice. Внесите какие-либо изменения. |
| :w | `"practice" 6 lines, 328 characters`<br><br>Сохраните отредактированный файл practice командой ex :w и нажмите ENTER. |
| :n | `Dear Mr.`<br>`Henshaw:`<br>`Thank you for the prompt . . .`<br><br>Командой ex :n вызовите следующий файл note, нажмите ENTER и сделайте исправления. |
| :x | `"note" 23 lines, 1343 characters`<br><br>Сохранение второго файла note и выход из сеанса редактирования. |

## Использование списка аргументов

ex позволяет выполнять гораздо больше действий, нежели простое переключение на следующий файл командой :n. Команда :args (сокращается до :ar) перечисляет все файлы, присутствовавшие в командной строке, при этом текущий файл заключается в скобки.

| Клавиши | Результат |
|---|---|
| vi practice note | `With a screen editor you can scroll`<br>`the page, move the cursor, delete lines,`<br>`insert characters, and more, while seeing`<br><br>Открываются два файла, practice и note. Сначала на экране появится указанный первым файл practice. Внесите какие-либо изменения. |
| :args | `[practice] note`<br><br>vi отобразит аргументы в строке состояния (имя текущего файла заключено в скобки). |

Команда :rewind (:rew) делает текущим первый файл в списке файлов, указанных в командной строке. В elvis и Vim есть соответствующая команда :last, которая переключается на последний файл в этом списке.

## Вызов новых файлов

Необязательно вызывать несколько файлов в начале сеанса редактирования. Командой :e можно переключиться на новый файл в любой момент. Если вы хотите отредактировать в vi другой файл, сохраните сначала текущий (:w), а затем наберите команду:

```
:e filename
```

Допустим, вы работаете с файлом practice и решили редактировать файл letter, а потом снова вернуться к practice:

| Клавиши | Результат |
|---|---|
| :w | `"practice" 6 lines, 328 characters`<br><br>Сохраните practice командой w и нажмите ENTER. practice сохранен и остается на экране. Сейчас можно переключиться на другой файл, поскольку ваши правки сохранены. |
| :e letter | `"letter" 23 lines, 1344 characters`<br><br>Вызовите командой e файл letter и нажмите ENTER. Проделайте какие-либо изменения. |

vi «помнит» два имени файла как текущее и альтернативное. К ним можно обращаться с помощью символов % (имя текущего файла) и # (альтернативное). # удобно использовать с командой :e, так как она позволяет легко переключаться вперед и назад по этим двум файлам. В только что приведенном примере командой :e # можно вернуться к первому файлу (practice). Содержимое файла practice можно считать в текущий файл, если вызвать :r #.

Если текущий файл не был сохранен, то vi не позволит перемещаться между файлами посредством команд :e или :n, пока вы не скажете ему сделать это принудительно, добавив восклицательный знак после команды.

Например, если после изменений в letter нужно отказаться от них и вернуться к practice, наберите :e! #.

Следующая команда тоже полезна. Она отменяет правки и возвращает пользователя к последней сохраненной версии текущего файла:

```
:e!
```

В отличие от символа #, % полезен только при записи содержимого текущего буфера в новый файл. Например, в недавнем разделе «Переименование буфера» на стр. 85 мы показали, как сохранить дополнительную версию файла practice при помощи команды:

```
:w practice.new
```

Так как имени текущего файла соответствует %, эту же строку можно ввести так:

```
:w %.new
```

## Переключение между файлами из редактора vi

Поскольку вам, скорее всего, придется часто переключаться на предыдущий файл, необязательно для этого обращаться к командной строке ex. Команда vi ^^ (клавиши Ctrl и каретка) сделает это за вас. Вызов этой команды эквивалентен вводу :e #. Как и в случае команды :e, если текущий буфер не сохранен, vi не позволит перейти к предыдущему файлу.

## Правки между файлами

Если присвоить буферу копирования (yank) однобуквенное имя, то появится удобный способ перемещения текста из одного файла в другой. При загрузке нового файла в буфер vi командой :e содержимое именованных буферов не очищается. Таким образом, скопировав или удалив текст из одного файла (при необходимости его можно поместить в несколько именованных буферов), вызвав новый файл командой :e и вставив именованные буферы в новый файл, вы сможете переносить фрагменты текста между файлами.

Следующий пример иллюстрирует, как перенести текст из одного файла в другой.

| Клавиши | Результат |
| --- | --- |
| "f4yy | `With a screen editor you can scroll`<br>`the page, move the cursor, delete lines,`<br>`insert characters, and more, while seeing`<br>`the results of the edits as you make them`<br><br>Скопируйте четыре строки в буфер f. |
| :w | `"practice" 6 lines, 238 characters`<br><br>Сохраните файл. |
| :e letter | `Dear Mr.`<br>`Henshaw:`<br>`I thought that you would`<br>`be interested to know that:`<br>`Yours truly,`<br><br>Зайдите в файл letter командой :e и переместите курсор туда, где должен будет располагаться скопированный текст. |
| "fp | `Dear Mr.`<br>`Henshaw:`<br>`I thought that you would`<br>`be interested to know that:`<br>`With a screen editor you can scroll`<br>`the page, move the cursor, delete lines,`<br>`insert characters, and more, while seeing`<br>`the results of the edits as you make them`<br>`Yours truly,`<br><br>Вставьте скопированный текст из именованного буфера f ниже курсора. |

Другой способ перемещения текста между файлами состоит в использовании команд ex :ya (от yank) и :pu (от put). Они работают примерно так же, как их vi-эквиваленты y и p, но используются с именованными буферами и адресацией строк, принятой в ex.

Например, команда:

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

скопирует строки со 160-й по 224-ю в буфер a. После этого командой :e нужно перейти к файлу, куда вы хотите вставить эти строки, поместить курсор на строку, где необходимо вставить скопированные строки, а затем ввести:

```
:pu a
```

чтобы вставить содержимое буфера a после текущей строки.

# 6

# Глобальная замена

Иногда в наполовину готовом документе или законченном черновике могут обнаружиться неточности. Например, в руководстве по эксплуатации может измениться название товара, присутствующее во всем файле (ох уж этот маркетинг!). Достаточно часто нужно вернуться и что-нибудь исправить, после чего сделать аналогичные изменения в нескольких местах.

Глобальная замена – мощный инструмент для подобных правок. Используя только одну команду, вы сможете автоматически заменять слово (или строку символов) в любом месте файла.

Во время глобальной замены редактор ex проверяет каждую строку файла на наличие заданного шаблона символов. Во всех строках, где такой шаблон обнаружен, ex заменяет его на *новую строку символов*. Сначала в качестве шаблона поиска мы будем рассматривать простую строку, а далее в этой главе изучим мощный язык поиска по шаблону, известный как *регулярные выражения*.

При глобальной замене ex использует только две команды: :g (global) и :s (substitute). Поскольку их синтаксис может быть довольно сложным, мы начнем рассматривать его поэтапно.

Команда замены имеет следующий вид:

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

При этом в текущей строке *первое* вхождение шаблона *old* заменится на *new*. Знак / (косая черта) служит разделителем между разными частями команды (когда косая черта является последним символом в строке, ее можно не ставить).

Команда замены с синтаксисом:

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

меняет в текущей строке *каждое* вхождение *old* на *new*. Команда :s допускает наличие опций после строки замены. Опция g в приведенном примере означает *глобальность*. (Опция g относится ко всем вхождениям шаблона в строке; не путайте ее с командой :g, затрагивающей каждую строку файла.)

Если перед командой :s приписать адреса, то диапазон ее действия расширится на несколько строк. Например, следующая команда будет менять каждое вхождение *old* на *new* в строках с 50-й по 100-ю:

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

а эта поменяет каждое вхождение *old* на *new* во всем файле:

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

Чтобы задавать действие в каждой строке файла, можно вместо 1,$ указывать %. Так, последняя команда может быть записана следующим образом:

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

Глобальная замена работает намного быстрее поиска и замены каждого вхождения строки. Поскольку эта команда может использоваться для самых разных исправлений и обладает большими возможностями, мы сначала проиллюстрируем простые замены, а затем дойдем до сложных контекстно-зависимых правок.

# Подтверждаем замены

При использовании команд поиска и замены лишняя осторожность не помешает, так как иногда получается совсем не то, что ожидается. Например, вводом u можно отменить последнюю команду поиска и замены. Однако часто нежелательные изменения обнаруживаются, когда уже поздно их отменять. Другой способ защиты файла – сохранить его командой :w перед глобальной заменой. Так у вас по крайней мере будет возможность выйти из файла без сохранения и вернуться к сохраненному варианту. Также целесообразно воспользоваться командой :e!, чтобы считать предыдущую версию буфера.

Рекомендуется проявлять осторожность и хорошо понимать, что именно нужно изменить в файле. Если вы хотите увидеть результат поиска и подтверждать каждую замену перед ее выполнением, добавьте в конец команды замены опцию c (confirm):

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

ex отобразит всю строку с искомым текстом и пометит его набором кареток (^^^^):

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

Чтобы выполнить эту замену, введите y (yes) и нажмите ENTER. Для отмены правки просто нажмите ENTER.

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

Сочетание команд vi n (повтор последнего поиска) и точки (.) (повторение последней команды) также является чрезвычайно важным и быстрым способом пройтись по файлу и сделать множественные замены, если вы не хотите применять их глобально. Например, если нужно заменить все *which* на *that*, то можно сделать выборочную проверку каждого вхождения *which* и поменять только неправильные из них:

| | |
|---|---|
| /which | Поиск *which* |
| cwthat ESC | Сменить на *that* |
| n | Повторить поиск |
| n | Повторить поиск, пропустить изменение |
| . | Повторить изменение (если возможно) |
| | (и т. д.) |

# Замена, зависящая от контекста

Простейшая глобальная замена меняет одно слово (или фразу) на другое. Если был набран файл с повторяющимися опечатками (*editer* вместо *editor*), то можно сделать глобальную замену следующим образом:

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

При этом *editer* будет заменен на *editor* во всем тексте. Для глобальной замены есть еще один, более сложный синтаксис. Он позволяет искать по одному шаблону, а после нахождения содержащей его строки сделать замену по другому шаблону. Это можно рассматривать как замену, зависящую от контекста.

Синтаксис имеет следующий вид:

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

Первая g предписывает команде обработать все строки в файле. По *pattern* ищутся строки, где должна быть произведена замена, а в строках, содержащих *pattern*, ex должен заменить (s) *старые* символы (old) на *новые* (new). Последняя опция g указывает, что эта замена должна выполняться глобально *в данной строке*.

Например, при написании этой книги мы использовали XML-директивы <keycap> и </keycap> для обрамления сочетания ESC, обозначающего клавишу Escape. Допустим, вы хотите писать ESC прописными буквами, но не желаете менять другие *Escape*, встречающиеся в тексте. Чтобы изменить *Esc* на *ESC* только в строке с директивой <keycap>, используйте следующее выражение:

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

Если вам нужно не только найти строку по шаблону, но и поменять ее, не обязательно набирать этот шаблон дважды. Так, команда:

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

выполнит поиск всех строк, содержащих *string*, и выполнит замену этой самой *string*.

Обратите внимание, что команда:

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

будет делать то же, что и:

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

Во втором случае писать нужно немного меньше. Также возможно комбинировать команду `:g` не только с `:s`, но и с другими командами `ex`, например `:d`, `:mo` и `:co`. Этим способом можно проделывать глобальные удаления, перемещения и копирования.

# Поиск по шаблону

При выполнении глобальных замен UNIX-редакторы, такие как `vi`, позволяют искать не только фиксированные строки символов, но и настраиваемые шаблоны слов, которые называются *регулярными выражениями*.

Когда вы указываете в шаблоне строку символов-литералов, поиск может затронуть неподходящие вхождения. Проблема в том, что одно и то же слово может по-разному использоваться. Регулярные выражения позволяют провести поиск слов в контексте. Заметим, что регулярные выражения можно использовать как в командах поиска `vi` `/` и `?`, так и в командах `ex` `:g` и `:s`.

В большинстве случаев те же регулярные выражения подойдут и для других утилит UNIX, например `grep`, `sed`, и `awk`[1].

Регулярные выражения строятся комбинированием обычных символов с *метасимволами*[2]. Метасимволы и способы их использования перечислены ниже.

---

1 Гораздо больше информации о регулярных выражениях можно найти в двух книгах, изданных O'Reilly: «sed & awk» Дейла Догерти (Dale Dougherty) и Арнольда Роббинса (Arnold Robbins) и «Mastering Regular Expressions», Джеффри Е.Ф. Фридла (Jeffrey E.F. Friedl) – в переводе «Регулярные выражения», 3-е издание. – Символ-Плюс, 2008.

2 Говоря техническим языком, их правильнее называть *метапоследовательностями*, поскольку иногда два символа, стоящие рядом, имеют особое значение, отличное от их значения по отдельности. Тем не менее термин *метасимволы* широко используется в литературе по UNIX, поэтому мы будем его придерживаться.

## Метасимволы, используемые при поиске по шаблону

. (*точка*)

Соответствует любому *одиночному* символу, отличному от перевода строки. Не забывайте, что пробелы также рассматриваются как символы. Например, выражению `p.p` соответствуют такие строки, как *pep*, *pip* и *pcp*.

*

Соответствует повторению любого (в том числе и нулевого) количества раз символа, стоящего непосредственно перед звездочкой. Например, `bugs*` соответствует *bugs* (одна *s*), *bug* (ни одной *s*), *bugss*, *bugsss* и т. д.

Звездочка `*` может стоять также после метасимвола. К примеру, в силу того что `.` (точка) отвечает любому символу, сочетание `.*` означает «любое количество любых символов».

А вот особый пример: команда `:s/End.*/End/` удаляет все символы после *End* (остаток строки заменяется на «ничто»).

^

При использовании в начале регулярного выражения эта команда требует, чтобы последующее регулярное выражение находилось в начале строки. Например, `^Part` соответствует *Part*, стоящей в начале строки, а `^...` – первым трем символам в строке. Если знак `^` стоит не в начале регулярного выражения, то он означает сам себя – символ каретки.

$

При использовании в конце регулярного выражения требует, чтобы предшествующее регулярное выражение располагалось в конце строки. Например, `here:$` соответствует только тем *here:*, которые стоят в конце строки. Если `$` находится не в конце регулярного выражения, то это просто знак доллара.

\

Указывает, что последующий специальный символ должен рассматриваться как обычный. К примеру, сочетание `\.` означает просто точку, а не «любой одиночный символ»; `*` – символ звездочки, а не «произвольное количество символов». \ (обратная косая черта) предотвращает интерпретацию специального символа. Это называется «экранированием символа» (escaping the character). Сам символ обратной черты вводится как \\.

[ ]

Соответствует любому из символов, заключенных в квадратные скобки. Например `[AB]` соответствует *A* или *B*, а `p[aeiou]t` соответствует *pat*, *pet*, *pit*, *pot* или *put*. Последовательный диапазон символов можно определить, поставив знак «минус» между первым и последним символами этого диапазона. Например, `[A-Z]` будет соответствовать любой заглавной латинской букве, а `[0-9]` – любой цифре от *0* до *9*.

Внутри скобок можно размещать несколько диапазонов, а также смесь диапазонов и отдельных символов. Например, [:;A-Za-z(  )] соответствует всем буквам и четырем знакам препинания.

В самом начале развития vi и регулярных выражений предполагалось, что они будут работать только для набора символов ASCII. В эпоху глобализации современные системы поддерживают *локали*, что дает разную интерпретацию символов, лежащих между a и z. Чтобы получить правильные результаты, нужно использовать в регулярных выражениях специальные конструкции POSIX (о них чуть ниже) и избегать диапазонов вида a-z.

В скобках большая часть метасимволов теряет свое специальное значение, поэтому при их использовании в качестве обычных символов экранирование не требуется. Единственные символы, которые нужно экранировать внутри скобок – это \, - и ]. Здесь дефис (-) имеет значение указателя диапазона, а чтобы использовать «настоящий» дефис, его следует поставить внутри скобок на первом месте.

Каретка (^) имеет особое значение, только если является первым символом в скобках, но и в этом случае ее значение отличается от обычного метасимвола ^. Будучи первым знаком в квадратных скобках, ^ меняет их значение на обратное, после чего они будут соответствовать любому из символов *не* из списка. Например, [^0-9] обозначает произвольный нецифровой символ.

\( \)

Сохраняет шаблон, заключенный между \( и \), в специальном месте или временном буфере (hold buffer). Таким способом можно сохранять до девяти буферов. Например, шаблон:

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

сохраняет *That* во временный буфер номер 1, а *this* – во временный буфер номер 2. Сохраненные шаблоны могут быть «воспроизведены» с помощью конструкций от \1 до \9. Например, чтобы фраза *That or this* превратилась в *this or That*, можно ввести:

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

Внутри строки поиска или замены также можно использовать обозначение \n. Например:

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

меняет *abcdabcd* на *alphabet-soup*[1].

\< \>

Соответствует символам в начале (\<) или в конце (\>) слова. Начало и конец слова определяются либо по знаку препинания, либо по про-

[1] Это работает в vi, nvi и Vim, но не работает в elvis и vile.

белу. Например, выражение \<ac будет соответствовать только словам, начинающимся на *ac*, таким как *action*. В свою очередь, выражение ac\> соответствует словам, оканчивающимся на *ac*, например *maniac*, а слово *react* не отвечает ни одному из этих выражений. Обратите внимание, что, в отличие от \(...\), их можно использовать не в паре.

~

Соответствует регулярному выражению, которое использовалось в последнем поиске. Так, если вы искали *The*, то слово *Then* можно найти при помощи /~n. Данный шаблон применяется только при обычном поиске (клавишей /)[1]. Его нельзя задать в качестве шаблона в команде подстановки, но он имеет схожее значение при использовании в качестве части строки замены в команде подстановки.

Все клоны поддерживают необязательный (расширенный) синтаксис регулярных выражений. За подробностями обращайтесь к разделу «Расширенные регулярные выражения» на стр. 152.

## Выражения в скобках в стандарте POSIX

Мы рассказали об использовании скобок для сопоставления с любым из заключенных в них символов, например [a-z]. Стандарт POSIX вводит дополнительные функциональные возможности для сопоставления с буквами других алфавитов. Например, французское *è* – это буква, но в обычный класс символов [a-z] она не входит. Дополнительно в этом стандарте введены последовательности символов, с которыми нужно обращаться как с единым блоком при поиске соответствия и при упорядочении (сортировке) строковых данных.

Кроме того, в POSIX стандартизирована терминология. Например, группа символов в скобках называется *групповым выражением* (bracket expression). Внутри таких выражений, помимо литеральных символов (*a*, *!* и прочие), могут присутствовать дополнительные *компоненты*:

*Классы символов*

Класс символов POSIX состоит из ключевых слов, заключенных между [: и :]. Ключевые слова описывают различные классы символов, такие как алфавитные, управляющие символы и т. д. (см. табл. 6.1).

*Объединенные символы*

Объединенный символ – это многосимвольная последовательность, рассматриваемая как единое целое. Ее составляют символы, заключенные между [. и .].

---

[1] Это довольно странная особенность оригинального vi. После ее использования сохраненным шаблоном поиска становится *новый* текст, набранный после ~, *а не* совокупный новый шаблон, как можно было бы ожидать. Кроме того, ни один из клонов так себя не ведет. Так что хоть такая функция и существует, мы не будем рекомендовать ею пользоваться.

*Классы эквивалентности*

Класс эквивалентности – это список символов, которые считаются эквивалентными, например *e* и *è*. Этот класс содержит именованный элемент из конкретной локали, стоящий между [= и =].

Все три конструкции должны находиться в квадратных скобках скобкового выражения. Например, [[:alpha:]!] соответствует любому одиночному символу или восклицательному знаку, а [[.ch.]] – объединенному элементу *ch*, а не буквам *c* или *h*. Во французской локали [[=e=]] соответствует буквам *e*, *è* и *é*. Классы и символы соответствия показаны в табл. 6.1.

*Таблица 6.1. Классы символов POSIX*

| Класс | Соответствующие символы |
|---|---|
| [:alnum:] | Буквенно-цифровые символы |
| [:alpha:] | Алфавитные символы |
| [:blank:] | Символы пробела и табуляции |
| [:cntrl:] | Управляющие символы |
| [:digit:] | Цифровые символы |
| [:graph:] | Печатаемые и видимые символы (не-пробелы) |
| [:lower:] | Строчные символы |
| [:print:] | Печатаемые символы (включая пробельные) |
| [:punct:] | Знаки препинания |
| [:space:] | Пробельные символы |
| [:upper:] | Прописные символы |
| [:xdigit:] | Шестнадцатеричные цифры |

В системах HP-UX 9.x (и более новых) vi поддерживает скобковые выражения POSIX. То же касается /usr/xpg4/bin/vi (но не /usr/bin/vi) в Solaris. Эти возможности также реализованы в nvi, elvis, Vim и vile. В частности, современные системы GNU/Linux чувствительны к локали, выбранной во время установки, поэтому можно ожидать разумных результатов при поиске прописных и строчных букв, используя скобковые выражения POSIX.

## Метасимволы, используемые в строках замены

При глобальной замене описанные выше метасимволы регулярных выражений имеют особое значение только в строке поиска (первой части) команды.

Например, при использовании следующей команды:

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

обратите внимание, что в строке замены символы . и $ понимаются буквально, так что вам не нужно их экранировать. Более того, предположим, что вы ввели:

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

Если в данном случае вы рассчитывали заменить *A* на *a*, *B* на *b* и *C* на *c*, то будете разочарованы. Поскольку в строке замены скобки ведут себя как обычные символы, эта команда заменит любое вхождение *A*, *B* или *C* на пятисимвольную строку *[abc]*.

Для разрешения подобной проблемы следует как-то определить регулируемую строку замены. К счастью, существуют дополнительные метасимволы, имеющие особое значение в *строке замены*.

*n*

Заменяется на текст, соответствующий *n*-му шаблону, сохраненному ранее с помощью команд \\( и \\), где *n* – это число от 1 до 9, и сохраненные ранее шаблоны (находящиеся во временных буферах) считаются с левого края строки (об использовании \\( и \\) читайте в предыдущем разделе «Метасимволы, используемые при поиске по шаблону» на стр. 95).

\\

Требует, чтобы следующий специальный символ рассматривался как обычный. Обратная косая черта является таким же метасимволом в строке замены, как и в строке поиска. Чтобы определить сам символ обратной косой черты, введите две черты подряд (\\\\).

&

Находясь в строке замены, знак & заменяется на весь текст, соответствующий строке поиска. Это полезно, если вам хочется избежать повторного ввода текста:

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

Строка замены означает *Yazstremski, Carl*. Также & может заменять переменный шаблон (в зависимости от регулярного выражения). Например, чтобы заключить каждую из строк с 1-й по 10-ю в круглые скобки, введите:

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

Шаблон поиска соответствует всей строке, а символ & «воспроизводит заново» всю строку из вашего текста.

~

Имеет значение, сходное с тем, которое используется в строке поиска: найденная строка заменяется на текст, определенный в последней команде подстановки. Это полезно при повторении исправлений. Например, можно в одной строке написать :s/thier/their/, а в другой – :s/thier/~/. При этом строки поиска не обязательно должны быть одинаковыми.

Например, можно в одной строке ввести `:s/his/their/`, а в другой провести замену `:s/her/~/`[1].

`\u` *или* `\l`

Требует, чтобы следующий символ в строке замены исправлялся на прописной (uppercase) или строчной (lowercase) соответственно. Например, чтобы исправить *yes, doctor* на *Yes, Doctor*, можно написать:

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

Смысла в этом примере мало, так как проще написать слова в строке замены с заглавных букв. Как и для любого регулярного выражения, наиболее полезными `\u` и `\l` оказываются в случае переменных строк. Возьмем, например, упомянутую ранее команду:

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

В результате ее выполнения получится выражение *this or That*. Здесь нужно поправить регистр. Используем команду `\u`, чтобы сделать прописной первую букву в *this* (которая теперь содержится во временном буфере 2), затем введем `\l`, чтобы сделать строчной первую букву в *That* (она сейчас находится во временном буфере 1):

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

Результатом будет выражение *This or that* (не путайте номер один и букву `l`, стоящую рядом; сначала идет буква).

`\U` *или* `\L` *и* `\e` *или* `\E`

Действие команд `\U` и `\L` похоже на `\u` или `\l`, но в первом случае все последующие символы преобразуются в прописные или строчные, пока не встретится конец строки замены или один из метасимволов `\e` или `\E`. Если `\e` и `\E` отсутствуют, `\U` и `\L` действуют на всю строку замены. Например, чтобы перевести *Fortran* в прописные буквы, можно ввести:

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

или с использованием символа `&`, повторяющего строку поиска:

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

Все поисковые шаблоны регистрозависимы. В силу этого поиск для *the* не отыщет *The*. Это поправимо, если в шаблоне определить как прописную, так и строчную буквы:

```
/[Tt]he
```

Вводом команды `:set ic` можно заставить `vi` игнорировать регистр. В главе 7 об этом рассказывается подробнее.

[1] В современных версиях редактора `ed` смысл «использовать текст последней замены» имеет только символ `%` в строке подстановки.

## Другие трюки при заменах

Следует знать следующие важные факты о командах замены:

- Простое `:s` равносильно `:s//~/`. Другими словами, это повторение предыдущей замены. Если вы работаете с большим документом и выполняете раз за разом одну и ту же правку, но не хотите использовать глобальную замену, это поможет сэкономить кучу времени и сил.
- Вы можете рассматривать команду `&` как «то же самое» (по аналогии с метасимволом строки подстановки); такой подход можно считать достаточно мнемоничным. После `&` можно записать `g`, чтобы сделать глобальную замену по всей строке, и даже использовать строковые диапазоны:

  `:%&g` *Повторить последнюю замену по всему тексту.*

- Клавиша & может использоваться в качестве команды `vi`. Ее действие совпадает с командой `:&` (повторение последней замены). Это экономит еще больше сил, чем `:s` ENTER – одно нажатие клавиши против трех.
- Команда `:~` похожа на `:&`, но есть небольшое отличие. Здесь применяется поисковый шаблон, стоявший в регулярном выражении, используемым *любой* последней командой: эта команда не обязана быть командой подстановки.

  Например[1], в последовательности:

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

  действие `:~` эквивалентно `:s/green/blue/`.

- Кроме / можно использовать любой неалфавитно-цифровой и непробельный символ в качестве разделителя, кроме обратной косой черты (\), двойных кавычек (") и вертикальной черты (|). Это особенно удобно, когда необходимо произвести замену в пути к файлу:

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

- При включеннной опции `edcompatible` `vi` запоминает флаги (`g` – для глобальной замены, `c` – для подтверждения), использовавшиеся в последней команде, и применяет их к следующей.

  Это помогает, если вы двигаетесь по файлу и хотите сделать глобальные замены. Первую правку можно сделать следующим образом:

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

  Теперь все последующие замены будут глобальными.

  Несмотря на название, ни одна из известных версий `ed` для UNIX на самом деле подобным образом не работает.

---

[1] Спасибо Кейту Бостичу (Keith Bostic) за этот пример из документации `nvi`.

# Примеры использования шаблонов

Если ранее вы не были знакомы с регулярными выражениями, то, возможно, последний материал о специальных символах показался вам слишком сложным. Несколько примеров должны прояснить дело. В последующих примерах для обозначения пробела используется квадрат (□). Это не специальный символ.

Давайте проработаем вопрос об использовании в заменах некоторых специальных символов. Пусть у нас есть большой файл, где мы хотим заменить все слова *child* на *children*. Сначала сохраним буфер редактирования командой `:w` и попытаемся провести глобальную замену:

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

После этого можно заметить слова вроде *childrenish*. Мы сделали непреднамеренную правку в слове *childish*. Вернувшись к последнему сохраненному буферу с помощью команды `:e!`, попробуем следующее:

```
:%s/child□/children□/g
```

(После слова *child* стоит пробел.) Однако эта команда пропускает вхождения *child.*, *child,*, *child:* и т. д. Немного подумав, вспоминаем, что с помощью скобок можно задавать символ из списка, и приходим к следующему решению:

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

Здесь производится поиск слова *child*, после которого идет либо пробел (у нас он отмечен как □), либо один из знаков препинания: `,.;:!?`. Ожидается, что это слово будет заменено на *children*, после которого стоит пробел или соответствующий знак препинания, но на деле мы получим ворох знаков препинания после каждого слова *children*. Необходимо сохранить пробел и знаки препинания внутри пары `\(` и `\)`. После этого их можно «воспроизвести» с помощью `\1`. Новая попытка будет иметь следующий вид:

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

Когда поиск наткнется на символ, стоящий между `\(` и `\)`, команда `\1` в правой части восстановит этот символ. Ее синтаксис может показаться ужасно сложным, но она сэкономит вам много времени. *Всякое время, затраченное на изучение синтаксиса регулярных выражений, воздастся тысячу раз!*

Тем не менее наша команда еще не идеальна. Можно заметить, что вхождения *Fairchild* тоже подверглись замене, то есть нужно искать только те *child*, которые не являются частью другого слова.

Оказывается, `vi` (но отнюдь не все программы, использующие регулярные выражения) имеет особую конструкцию, означающую «только если шаблон является словом целиком». Последовательность символов `\<` требует, чтобы шаблон являлся началом слова, а `\>` требует, чтобы он был концом слова. Так, в задаче нашего примера `\<child\>` отыщет все

вхождения слова *child*, окруженного как знаками препинания, так и пробелами. Используем следующую команду замены:

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

## Поиск общего класса слов

Предположим, имя процедуры начинается с приставок *mgi*, *mgr* и *mga*:

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

Если вы хотите сохранить эти приставки, но поменять в имени процедуры *box* на *square*, то любая из приведенных ниже замен решит поставленную задачу. Первый пример показывает, как используются \( и \) для сохранения встретившегося шаблона, а второй – как искать по одному шаблону, но менять другой:

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

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

Глобальная замена запоминает, какая именно из букв (*i*, *r* или *a*) встретилась в слове. Здесь *box* меняется на *square* только в случае, если *box* является частью имени процедуры.

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

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

Эта команда оказывает то же действие, что и предыдущая, но она менее безопасная, поскольку может поменять другие вхождения *box* в той же строке, а не только те, которые появлялись в именах процедур.

## Перемещение блока по шаблону

Также можно перемещать текстовые блоки, границы которых задаются шаблонами. Предположим, имеется 150-страничное справочное руководство, написанное на `troff`. Каждая страница разбита на три абзаца с тремя заголовками SYNTAX, DESCRIPTION и PARAMETERS. Вот пример страницы из руководства:

```
.Rh 0 "Get status of named file" "STAT"
.Rh "SYNTAX"
```

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

Допустим, вы захотели переместить DESCRIPTION выше абзаца SYNTAX. Используя шаблоны, можно обработать все 150 страниц всего одной командой!

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

Она работает следующим образом: сначала `ex` находит и помечает каждую строку, соответствующую первому шаблону (то есть содержащую слово *SYNTAX*); затем каждая помеченная строка становится текущей (.), и для нее выполняется остальная часть команды. С помощью команды `move` она переносит блок, состоящий из строк, начинающихся с текущей (обозначенной точкой) до строки, предшествующей строке со словом *DESCRIPTION* (`/DESCRIPTION/-1`), на место, предшествующее строке, содержащей *PARAMETERS* (`/PARAMETERS/-1`).

Обратите внимание, что `ex` всегда помещает текст ниже указанной строки. Чтобы `ex` размещал текст строкой выше, вы вычитаете единицу (`-1`). В представленном примере одна команда экономит буквально часы работы (это пример из реальной жизни: однажды мы воспользовались похожими шаблонами, чтобы поменять порядок в справочном руководстве, содержащем сотни страниц).

Определение блока шаблонами можно использовать и с другими командами `ex`. Например, если нужно удалить все абзацы DESCRIPTION в главе руководства, можно ввести следующую команду:

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

Столь мощный механизм замены является следствием синтаксиса адресации строк в `ex`, но он не очевиден даже для опытных пользователей. По этой причине всякий раз, когда вы сталкиваетесь со сложной и многоступенчатой задачей редактирования, затратьте немного времени на

анализ проблемы и подумайте о возможности применения средств работы с шаблонами для выполнения этой задачи.

## Другие примеры

Поскольку примеры – наилучший способ изучить использование шаблонов, мы приводим список примеров, сопровождая их объяснениями. Внимательно следите за синтаксисом, чтобы понять принцип их работы. После этого вы сможете приспособить эти примеры под свои нужды:

1. Поместить код выделения курсивом в `troff` вокруг слова *ENTER*:

   ```
   :%s/ENTER/\\fI&\\fP/g
   ```

   Обратите внимание, что в замене требуется поставить две обратные косые черты, поскольку одна обратная косая черта в коде выделения курсивом будет считаться специальным символом (`\fI` будет интерпретирован как *fI*. Чтобы получить *\fI*, следует писать `\\fI`).

2. Изменить список путей к файлам:

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

   В тех случаях, когда косая черта (используемая как разделитель в конструкции глобальной замены) является частью шаблона или замены, она должна быть экранирована обратной косой чертой. Чтобы получить /, наберите \/. Другой способ достижения того же результата состоит в использовании другого символа в качестве разделителя в шаблоне. Например, аналогичную замену можно произвести, используя в качестве разделителя двоеточие. (Двоеточие-разделитель и двоеточие в команде `ex` – разные вещи.)

   Таким образом, мы получим:

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

   Это куда более удобочитаемая команда.

3. Поместить HTML-код курсива вокруг слова *ENTER*.

   ```
   :%s:ENTER:<I>&</I>:g
   ```

   Обратите внимание, что здесь используются `&` для представления текста, с которым было совпадение, и двоеточие в качестве разделителя.

4. В строках с 1-й по 10-ю поменять все точки на точки с запятой:

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

   В синтаксисе регулярных выражений точка имеет специальное значение, поэтому она должна быть экранирована обратной косой чертой (\.).

5. Поменять все вхождения слова *help* (или *Help*) на *HELP*:

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

   или

   ```
   :%s/[Hh]elp/\U&/g
   ```

\U меняет все буквы следующего за ним шаблона на прописные. Шаблон замены повторяет шаблон поиска, то есть *help* или *Help*.

6. Заменить *один или более* пробелов одним пробелом:

```
:%s/□□*/□/g
```

Убедитесь, что вам понятна работа звездочки в качестве специального символа. Звездочка, стоящая после любого символа (или любого регулярного выражения, выраженного одним символом, например . или [[:lower:]]), соответствует *любому* (в том числе *нулевому*) количеству копий этого символа. Следовательно, чтобы найти один или более пробелов (один пробел плюс ноль или большее количество пробелов), нужно указать два пробела и поставить после них звездочку[1].

7. Заменить один или более пробелов, перед которыми стоит двоеточие, на два пробела:

```
:%s/:□□*/:□□/g
```

8. Заменить один или более пробелов, перед которыми стоит точка *или* двоеточие, на два пробела:

```
:%s/\([:.]\)□□*/\1□□/g
```

Любой из двух символов, стоящих в квадратных скобках, считается удовлетворяющим шаблону. Этот символ сохраняется во временном буфере с помощью сочетаний клавиш \( и \) и восстанавливается в правой части конструкцией \1. Обратите внимание, что специальный символ между квадратными скобками (у нас это точка) экранировать не нужно.

9. Стандартизация различных вариантов слова или заголовка:

```
:%s/^Note[□:s]*/Notes:□/g
```

В квадратных скобках заключены три символа: пробел, двоеточие и буква *s*. Следовательно, шаблон Note[□s:] будет соответствовать *Note□*, *Notes* или *Note:*. К шаблону добавлена звездочка, так что он также соответствует *Note* (без пробелов после слова) и *Notes:* (правильное написание). Без звездочки *Note* не будет обрабатываться, а *Notes:* будет ошибочно изменено на *Notes:□:*.

10. Удалить все пустые строки:

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

Здесь ищется начало строки (^), за которым следует ее конец ($), и между ними ничего нет.

11. Удалить все пустые строки, а также строки, содержащие только пробельные символы:

```
:g/^[□tab]*$/d
```

---

[1] В других диалектах регулярных выражений (и клонах vi) для этих целей может использоваться особый спецсимвол, например \+. – *Прим. науч. ред.*

(В этом примере табуляция обозначена как *tab*.) Строка может выглядеть как пустая, но при этом содержать пробелы или табуляции. Команда из предыдущего примера не будет трогать подобные строки. В этом примере, как и в предыдущем, ищется начало и конец строки, но теперь между ними шаблон пытается отыскать любое количество пробелов или табуляций. Если ни тех, ни других нет, то строка пустая. Чтобы удалить строки, содержащие пробельные символы (то есть эти строки *не пустые)*, нужно, чтобы строка содержала *хотя бы один* пробел или табуляцию:

```
:g/^[□tab][□tab]*$/d
```

12. В каждой строке удалить все пробелы в начале строки:

    ```
    :%s/^□□*\(.*\)/\1/
    ```

    Сначала `^□□*` используется для поиска одного или нескольких пробелов в начале строки, затем последовательность `\(.*\)` сохраняет остальную часть этой строки в первый временный буфер, а `\1` восстанавливает строку без стоящих в начале нее пробелов.

13. Удалить все пробелы в конце каждой строки:

    ```
    :%s/\(.*\)□□*$/\1/
    ```

    В каждой строке команда `\(.*\)` сохраняет весь содержащийся в этой строке текст, исключая один или несколько пробелов в ее конце. При этом восстанавливаемый текст не содержит завершающих пробелов. В этом и в предыдущем примерах замена для строки происходит только единожды, так что опцию `g` после строки замены указывать необязательно.

14. Вставить в начало каждой строки файла `>□□`:

    ```
    :%s/^/>□□/
    ```

    Здесь начало строки просто «заменяется» на `>□□`. Конечно же, на самом деле начало строки (будучи логической конструкцией, а не реальным символом) никуда не девается!

    Эта команда полезна при ответе на почту или новостные посты Usenet. Зачастую желательно включать в ответ часть оригинального сообщения. По соглашению это включение отличается от вашего ответа тем, что в начале строки помещается правая угловая скобка и несколько пробелов. Как показано в примере, это легко проделать (как правило, включается только часть первоначального сообщения. Ненужный текст можно удалить либо до, либо после замены). Продвинутые почтовые системы делают это автоматически. Однако если для редактирования электронной почты вы используете `vi`, можете применять эту команду.

15. Добавить точку в конец следующих шести строк:

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

Адрес строк задается текущей строкой плюс пять строк. Знак $ указывает на конец строки. Как и в предыдущем примере, $ – это логическая конструкция. На самом деле конец строки не заменяется.

16. Поменять порядок пунктов, разделенных дефисом в списке.

```
:%s/\(.*\)□-□\(.*\)/\2□-□\1/
```

Последовательность \(.*\) используется для сохранения текста строки в первом временном буфере от начала до первого вхождения □-□. Затем команда \(.*\) сохраняет остаток строки во второй временный буфер. Эти «порции» строки восстанавливаются в обратном порядке. Ниже показан результат действия этой команды на нескольких примерах:

```
more - display files
```

становится:

```
display files - more
```

и

```
lp - print files
```

становится:

```
print files - lp
```

17. Поменять каждую букву в файле на прописную:

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

или

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

Флаг \U в начале строки замены приказывает vi сменить корректируемые знаки на прописные. Символ & воспроизводит текст, соответствующий строке поиска, в качестве замены. Эти две команды эквивалентны, но первая выполняется значительно быстрее, так как она сводится к одной замене в строке (.* соответствует всей строке и выполняется один раз за строку), тогда как вторая представляет собой многократные замены в одной строке (. соответствует одиночному символу, а замена повторяется из-за стоящего в конце g).

18. Обратить порядок строк в файле[1]:

```
:g/.*/mo0
```

Шаблон поиска соответствует всем строкам в файле (число символов в строке произвольно, в том числе и ноль). Каждая строка по очереди перемещается наверх файла (после воображаемой строки 0), смещая ниже перемещенные ранее строки. Так происходит, пока последняя строка не окажется сверху. Поскольку у каждой строки есть начало, тот же результат может быть достигнут более лаконично:

```
:g/^/mo0
```

[1] Из статьи Вальтера Зинтца (Walter Zintz) в *UNIX World*, май 1990.

19. Во всех строках текстовой базы данных, не отмеченных как *Paid in full*, приписать фразу *Overdue*:

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

или равносильно:

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

Чтобы подействовать на все строки, кроме удовлетворяющих шаблону, добавьте к команде `g` знак `!` либо просто используйте `v`.

20. Каждую строку файла, не начинающуюся с цифры, переместить в конец файла:

```
:g!/^[[:digit:]]/m$
```

или

```
:g/^[^[:digit:]]/m$
```

Будучи первым символом в квадратных скобках, каретка инвертирует их значение, так что эти две команды работают одинаково. Первая говорит: «не смотри на строки, начинающиеся с цифры», а вторая – «смотри на строки, которые не начинаются с цифры».

21. Сменить заголовки разделов, пронумерованных вручную (например, 1.1, 1.2 и т. д.), на макрос `troff` (например, `.Ah` для заголовка уровня A):

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

Строка поиска ищет соответствие любой отличной от нуля цифре, за которой стоит точка, а после нее – еще одна ненулевая цифра. Обратите внимание, что в замене точку экранировать не нужно (хотя знак \ ничего не испортит). Показанная здесь команда не найдет номера глав, содержащих две и более цифр. Чтобы это исправить, ее нужно привести к следующему виду:

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

Теперь она будет искать главы с 10-й по 99-ю (цифры с 1-й по 9-ю, за которыми стоит цифра), с 100-й по 999-ю (цифры с 1-й по 9-ю, за которыми – еще две цифры) и т. д. Главы с 1-й по 9-ю тоже найдутся (цифра с 1-й по 9-ю, после которой ничего нет).

22. Убрать нумерацию из заголовков документа. Допустим, вы хотите заменить строки вида:

```
2.1 Introduction
10.3.8 New Functions
```

на следующие:

```
Introduction
New Functions
```

Правку выполнит команда:

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

Поисковый шаблон напоминает аналог из предыдущего примера, но теперь числа могут иметь различную длину. Заголовки содержат, по меньшей мере, *номер, точку,* затем снова *номер*, поэтому шаблон поиска начинается аналогично шаблону из прошлого примера:

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

Однако в этом примере заголовок может продолжаться любым количеством цифр и точек:

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

23. Поменять слово *Fortran* на фразу *FORTRAN* (*акроним FORmula TRANslation*):

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

Заметив, что слова *FORmula* и *TRANslation* используют части первоначального слова, мы сначала сохраняем шаблон поиска в двух частях: `\(For\)` и `\(tran\)`. При первом восстановлении собираем эти кусочки вместе, переводя содержащиеся в них символы в верхний регистр: `\U\1\2`. Затем отменяем прописные буквы командой `\E`, иначе оставшаяся часть текста замены тоже будет в верхнем регистре. Замена продолжается явно указанным словом, после чего мы восстанавливаем первый временный буфер. Он все еще содержит *For*, поэтому мы снова переводим его буквы в прописные: `\U\1`. Сразу после этого прописываем остаток слова в нижнем регистре: `\Emula`. Наконец, восстанавливаем второй временный буфер. В нем находится фрагмент *tran*, так что перед «воспроизведением» мы переводим его в верхний регистр, затем снова в нижний, а потом вводим остаток слова: `U\2\Eslation)`.

# Финальный взгляд на шаблоны

Закончим эту главу несколькими задачами, включающими сложные концепции поиска по шаблону. Мы не будем сразу записывать их решение, а придем к нему шаг за шагом.

## Удаление текстового блока неизвестного размера

Пусть у нас есть несколько строк следующего общего вида:

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

Все рассматриваемые строки заканчиваются словом *moving*, но мы не знаем, какими будут первые два слова. Нужно поменять каждую строку, заканчивающуюся на *moving*, на следующую:

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

Поскольку изменения должны произойти в определенных строках, необходимо определить глобальную замену, зависящую от контекста. С по-

мощью команды :g/moving$/ можно распознать строки, заканчивающиеся на *moving*. Затем мы видим, что в строке поиска может оказаться сколько угодно произвольных символов, и на ум приходят метасимволы .*. Однако они обозначат всю строку, поэтому нужно как-то ограничить строку поиска. Первая попытка будет такой:

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

Как можно догадаться, данная команда соответствует куску строки от начала до слова *of*. Поскольку это слово требуется для ограничения поиска, мы просто повторим его в строке замены. В результате получится следующая строка:

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

Что-то мы сделали неправильно. Замена проглотила строку вплоть до второго *of* вместо первого. И вот почему: если будет выбор, то действие «соответствовать любому числу произвольных символов» возьмет *как можно больше текста*. В нашем случае, поскольку слово *of* встречается дважды, строка поиска находит:

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

а не

```
the best of
```

Наша строка поиска должна стать более строгой:

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

Команда .* включяет в себя все символы, вплоть до фразы *of times;*. Эта фраза встречается только один раз, поэтому она должна быть первой.

Тем не менее бывают ситуации, когда использование метасимволов .* неудобно и даже неправильно. Например, иногда для ограничения поиска приходится вводить много слов или строку поиска вообще невозможно задать конкретными словами (если текст в строках сильно отличается). Аналогичный случай представлен в следующем разделе.

## Смена записей в текстовой базе данных

Предположим, вы хотите поменять порядок следования имени/фамилии в (текстовой) базе данных. Строки имеют следующий вид:

```
Name: Feld, Ray; Areas: PC, UNIX; Phone: 123-4567
Name: Joy, Susan S.; Areas: Graphics; Phone: 999-3333
```

Имя каждого поля заканчивается двоеточием, а сами поля отделены друг от друга точкой с запятой. Например, в первой строке нужно заменить *Feld, Ray* на *Ray Feld*. Мы приведем несколько команд, которые выглядят многообещающе, но не работают. После каждой из них мы покажем, как выглядела строка до и после ее выполнения:

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

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

**Полужирным** выделено содержимое первого временного буфера, а *курсивом* – второго. Обратите внимание, что первый временный буфер содержит больше требуемого. Из-за недостаточной строгости шаблона в первый буфер занеслось все содержимое строки до второй запятой. Попробуем сильнее ограничить содержимое первого буфера:

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

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

Здесь у нас получилось сохранить фамилию в первый временный буфер, но во второй поместятся все символы, стоящие до второй точки с запятой в строке. Ограничим также и второй буфер:

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

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

Это даст нужный результат, но только в тех случаях, когда фамилия состоит из четырех букв, а имя – из трех (в предыдущем варианте была та же ошибка). Почему бы не вернуться к первой попытке, но теперь внимательнее подойти к выбору конца шаблона поиска?

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

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

Это работает, но мы продолжим обсуждение и рассмотрим еще один момент. Предположим, поле *Area* не всегда присутствует или необязательно является вторым. В подобных строках команда работать не будет.

Мы поставили эту проблему, чтобы высказать следующую мысль: всякий раз когда приходится заново придумывать шаблон, лучше поработать над переменными (или метасимволами), чем нагромождать еще больше текста для ограничения шаблона. Чем больше переменных вы используете, тем более гибкими будут ваши команды.

В нашем примере нужно снова подумать над шаблонами. Каждое слово начинается с прописной буквы, а после нее следует произвольное количество строчных, поэтому имена можно отслеживать следующим образом:

```
[[:upper:]][[:lower:]]*
```

В фамилии также могут присутствовать прописные буквы (например, *McFly*), поэтому необходимо отследить эту возможность во второй и последующих буквах:

```
[[:upper:]][[:alpha:]]*
```

Ничто не мешает использовать эту последовательность и для имен (кто знает, вдруг существует какой-нибудь *McGeorge Bundy*), поэтому команда примет следующий вид:

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

Довольно заумно, не так ли? Эта команда все еще не может обработать имена/фамилии вроде *Joy* и *Susan S*. Поскольку поле имени может включать инициал из одной буквы, вам придется поставить пробел с точкой во вторую пару скобок. Но хватит. Иногда определить что-то нужное сложнее, чем указать то, что *не* нужно. В базе данных из нашего примера после фамилий стоит запятая, так что поле фамилии может рассматриваться как строка символов, отличных от запятой:

```
[^,]*
```

Этот шаблон соответствует символам до первой запятой. Аналогично поле имени – это строка символов, *отличных от* точки с запятой:

```
[^;]*
```

Если разместить эти более эффективные шаблоны в предыдущей команде, то можно получить:

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

Похожая команда может использоваться для контекстной замены. Если все строки начинаются с *Name*, то целесообразно ввести:

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

или добавить звездочку после первого пробела на тот случай, если после двоеточия стоят дополнительные пробелы (или ни одного пробела):

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

## Использование :g для повторения команды

Как мы уже видели, обычно `:g` используется для отбора строк файла, к которым затем применяются команды, следующие за `g` в командной строке. Например, с помощью `g` мы выбираем строки, а затем делаем в них замены, выбираем их или удаляем:

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

Однако в своей статье из двух частей в «*UNIX World*»[1] Вальтер Зинтц (Walter Zintz) предлагает интересный взгляд на эту команду. В частности, она выбирает строки, но связанная с ней команда редактирования не обязательно должна действовать именно на эту строку.

---

[1] Часть 1, «`vi` Tips for Power Users», вышла в апрельском номере 1990 года *UNIX World*. Часть 2, «Using `vi` to Automate Complex Edits», была опубликована в майском номере того же года. Представленные примеры взяты из второй части.

Вместо этого он продемонстрировал технику, согласно которой вы сможете повторять команды ex произвольное число раз. Допустим, вы хотите поместить 10 копий строк с 12-й по 17-ю в самый конец файла. Можно ввести:

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

Это весьма неожиданное применение g, но оно работает! Команда g выбирает строку 1, выполняет заданную команду t, затем переходит к строке 2 и выполняет следующую копию команды. Когда будет достигнута строка 10, ex выполнит 10 копирований.

## Сбор строк

Далее будет показано еще одно продвинутое применение g, также основанное на идеях, изложенных в статье Зинтца (Zintz). Предположим, вы редактируете документ, состоящий из нескольких частей. Часть 2 этого файла приведена здесь, пропущенный текст показан многоточиями, а для удобства выведены номера строк:

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

400 ... and a complete index at the end.
401 .LP
402 Chapter 8
403 Screen Dimensions
404 Before you can do anything useful
405 on the screen, you need to know ...
555 .LP
556 Chapter 9
557 Editing the Screen
558 This chapter discusses …

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

Номера глав стоят в одной строке, а их заголовки – строчкой ниже. При этом текст главы (выделен **жирным**) начинается в следующей строке. Сперва нужно скопировать первую строку каждой главы и записать ее в уже существующий файл под именем begin.

Для этого используется следующая команда:

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

Перед ее выполнением необходимо перейти в самое начало файла. Прежде всего, здесь ищется *Chapter* в начале строки, но затем следует ввести команду для обработки первой строки каждой главы (которая следует за строкой с *Chapter*). Поскольку строка, начинающаяся с *Chapter*, теперь помечена как текущая, команда `.+2` укажет на две строки ниже. Эквивалентные адреса строк `+2` или `++` тоже будут работать. Нужно записать эти строки в существующий файл `begin`, для чего вызывается команда `w` с оператором перенаправления вывода `>>`.

Предположим, вам нужно переслать начала только глав, входящих в часть 2. Здесь необходимо ограничить строки, выбираемые командой `g`. После этого она изменяет свой вид на следующий:

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

Команда `g` отбирает строки, начинающиеся с *Chapter*, но ищет только во фрагменте файла, который начинается с *Part 2* и заканчивается *Part 3*. Если выполнить приведенную команду, то последними строками в файле `begin` станут:

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

С этих строк начинаются главы 7, 8 и 9.

Вдобавок к выделенным строкам можно скопировать в конец документа заголовки глав, чтобы сделать заготовку для оглавления. Для объединения второй команды с первой можно использовать знак вертикальной черты:

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

Помните, что в любой последующей команде адреса строк будут вычисляться относительно предыдущей команды. Первая команда отобрала строки (внутри части 2), которые начинаются с *Chapter*, а заголовки глав появляются на одну строку ниже этих строк. Следовательно, чтобы получить доступ к заголовкам глав в следующей команде, нужно использовать адрес `+` (либо, что эквивалентно, `+1` или `.+1`). Затем надо ввести `t$`, чтобы скопировать заголовки глав в конец файла.

Как показывают эти примеры, если думать и экспериментировать, то можно прийти к совершенно неожиданному решению задачи. Не бойтесь пробовать новое. Просто убедитесь, что ваш файл сохранен! (Конечно, располагая возможностью бесконечной «отмены», доступной в модификациях, вам даже не нужно заботиться о сохранении копии файла.)

# 7

# Продвинутое редактирование

В этой главе рассматриваются некоторые более продвинутые возможности редакторов vi и ex. Перед началом работы с представленными здесь концепциями стоит ознакомиться с материалом предыдущих глав.

Мы разделили эту главу на пять частей. В первой рассказывается о некоторых способах настройки окружения редактора. Вы узнаете, как использовать команду set и как с помощью файлов .exrc создавать несколько различных окружений редактирования.

Вторая часть излагает, как выполнять команды UNIX непосредственно из vi и использовать его для фильтрации текста через команды UNIX.

В третьей говорится о разных способах сохранения длинных последовательностей команд, сводя их к аббревиатурам или даже к командам, выполняющимся по нажатию одной клавиши клавиатуры (это называется *отображение* (*mapping*) клавиш). Также в ней содержится раздел, посвященный @-функциям, позволяющим хранить последовательности команд в буфере.

Четвертая часть рассказывает об использовании скриптов ex из командной строки UNIX или из скриптов оболочки. Такие скрипты являются мощными инструментами для многократного внесения однородных правок.

В пятой части говорится о некоторых функциях vi, которые будут полезны программистам. В этом редакторе есть опции для управления отступами строк и опция отображения невидимых символов (например, табуляций и переносов строк). Некоторые команды поиска особенно полезны в блоках программного кода и функциях языка C.

# Настройка vi

На разных терминалах `vi` ведет себя по-разному. В современных системах UNIX он получает рабочие инструкции о типе вашего терминала из базы данных терминалов `terminfo` (в старых системах `vi` использовал оригинальную базу данных `termcap`)[1].

Многие аспекты поведения `vi` регулируются опциями, которые задаются непосредственно в программе. Например, можно устанавливать правое поле, таким образом заставляя `vi` автоматически переносить строки, что позволит не нажимать на ENTER.

Вы можете менять эти опции из `vi` с помощью `ex`-команды `:set`. Кроме того, при запуске `vi` считывает файл `.exrc`, находящийся в домашнем каталоге, где могут содержаться различные инструкции. Прописав команды `:set` в этом файле, вы настроите стандартное поведение редактора под свои нужды.

Файлы `.exrc` можно располагать в разных каталогах, чтобы устанавливать разные опции для разных окружений. Например, для редактирования английских текстов вам потребуются одни установки, а для программного кода – совсем другие. Сначала будет обрабатываться файл `.exrc` из домашнего каталога, а затем – из текущего.

Наконец, при запуске `vi` выполняются все команды, хранящиеся в переменной окружения `EXINIT`. Установки в `EXINIT` имеют более высокий приоритет, чем установки, прописанные в файле `.exrc`.

## Команда :set

Команда `:set` может менять опции двух типов: переключатели, принимающие значения on (включен) или off (выключен), и опции с числовыми и строковыми значениями (например, положение правого поля или имя файла).

По умолчанию опция-переключатель может быть в состоянии «on» или «off». Команда ее включения имеет вид:

```
:set option
```

Для выключения опции следует выполнить команду:

```
:set nooption
```

Например, чтобы включить игнорирование регистра при поиске по шаблонам, введите:

```
:set ic
```

---

[1] Расположение этих двух баз данных отличается у разных производителей. Попробуйте ввести команды `man terminfo` и `man termcap`, чтобы получить информацию, касающуюся вашей системы.

Чтобы снова сделать vi регистрозависимым при поиске, наберите команду:

```
:set noic
```

Некоторые опции требуют присвоения им определенных параметров. Например, опция window устанавливает количество строк, отображаемых в «окне» экрана. Значения этих параметров устанавливаются знаком равенства (=):

```
:set window=20
```

Во время работы в vi можно проверить, какие именно установки он использует. Команда:

```
:set all
```

отображает полный список опций, как заданных вами, так и «выбранных» редактором по умолчанию.

Список должен выглядеть примерно так:[1]

```
autoindent          nomodelines                     noshowmode
autoprint           nonumber                        noslowopen
noautowrite         nonovice                        tabstop=8
beautify            nooptimize                      taglength=0
directory=/var/tmp  paragraphs=IPLPPPQPP LIpplpipnpbp  tags=tags /usr/lib/tags
noedcompatible      prompt                          tagstack
errorbells          noreadonly                      term=vt102
noexrc              redraw                          noterse
flash               remap                           timeout
hardtabs=8          report=5                        ttytype=vt102
noignorecase        scroll=11                       warn
nolisp              sections=NHSHH HUuhsh+c         window=23
nolist              shell=/bin/ksh                  wrapscan
magic               shiftwidth=8                    wrapmargin=0
nomesg              showmatch                       nowriteany
```

С помощью команды:

```
:set option?
```

можно вывести текущее значение любого параметра, а команда:

```
:set
```

показывает опции, заданные или измененные как посредством файла .exrc, так и во время текущего сеанса работы.

[1] Результат :set all сильно зависит от версии vi. Приведенный пример типичен только для vi в UNIX. Порядок следования опций — алфавитный по колонкам без учета стоящих в начале no.

Например, ее вывод может иметь следующий вид:

```
number sect=AhBhChDh window=20 wrapmargin=10
```

## Файл .exrc

Файл .exrc, управляющий окружением vi, расположен в домашнем каталоге (в который вы попадаете при входе в систему). Этот файл можно менять в редакторе, как и любой другой текстовый файл.

Если у вас нет такого файла, просто создайте его с помощью vi. Введите туда команды set, ab и map, которые вы хотите задействовать при работе в vi или ex (про команды ab и map рассказывается далее). Приведем пример файла .exrc:

```
set nowrapscan wrapmargin=7
set sections=SeAhBhChDh nomesg
map q :w^M:n^M
map v dwElp
ab ORA O'Reilly Media, Inc.
```

Поскольку он обрабатывается редактором ex перед переходом в визуальный режим (vi), начинать содержащиеся в .exrc команды с двоеточия не нужно.

## Различные окружения

Вдобавок к автоматическому считыванию файла .exrc, расположенного в домашнем каталоге, можно разрешить vi считывать файл с именем .exrc из текущего каталога. Это позволяет задавать опции, подходящие для конкретного проекта.

Во всех современных версиях vi для считывания файла .exrc из текущего каталога нужно установить опцию exrc в файле .exrc домашнего каталога:

```
set exrc
```

Этот механизм предотвращает возможность помещения в рабочий каталог файлов .exrc других пользователей, команды в которых могут нарушить безопасность системы[1].

Например, в каталоге, предназначенном в основном для программирования, можно иметь следующий набор опций:

```
set number autoindent sw=4 terse
set tags=/usr/lib/tags
```

а в каталоге, используемом для редактирования текстов, — другой набор:

```
set wrapmargin=15 ignorecase
```

---

[1] Первоначальные версии vi автоматически считывали оба файла при их обнаружении. Опция exrc устранила эту потенциальную брешь в безопасности.

Обратите внимание, что в файле .exrc из домашнего каталога можно включить определенные опции, а в файле из локального каталога – отключить их.

Также можно определить альтернативные окружения vi, сохранив настройки в файле, отличном от .exrc, и считать этот файл при помощи команды :so (so – это сокращение слова source).

Например:

```
:so .progoptions
```

Локальные файлы .exrc также используются при определении аббревиатур и отображений клавиш (про которые рассказывается далее в этой главе). Например, когда мы пишем книгу, то сохраняем все используемые в ней аббревиатуры в файл .exrc каталога с этой книгой.

## Некоторые полезные опции

Как показывает вывод команды :set all, в vi существует огромное количество опций. Одни из них предназначены для внутреннего использования в редакторе, и их значения обычно не изменяют, другие полезны в одних ситуациях, а в других – нет (например, noredraw и window могут пригодиться во время межконтинентального сеанса ssh). В табл. B.1 в разделе «Опции Solaris vi» на стр. 458 содержится краткое описание каждой опции. Мы рекомендуем уделить некоторое время на эксперименты с ними. Если вас заинтересовала какая-нибудь опция, попробуйте ее включить (или выключить) и посмотреть, что изменилось в редакторе. Таким образом можно обнаружить много полезного.

В разделе «Перемещение в строке» на стр. 35 мы уже говорили о том, что одна из этих опций – wrapmargin – особенно важна при редактировании текста, не являющегося программой. Она определяет размер правого поля, используемого для автоматического переноса текста (при этом можно не вводить возврат каретки вручную). Обычно этот параметр устанавливают в диапазоне от 7 до 15:

```
:set wrapmargin=10
```

Три другие опции определяют поведение vi при выполнении поиска. Обычно поиск различает прописные и строчные буквы (*foo* не совпадает с *Foo*), циклически возобновляется с начала файла (то есть вы можете начать поиск из любого места файла, но отыскать при этом все вхождения) и распознает метасимволы при поиске по шаблону. Этими параметрами управляют опции noignorecase, wrapscan и magic соответственно. Чтобы изменить любую из этих установок по умолчанию, можно включить противоположные опции, такие как ignorecase, nowrapscan и nomagic.

Программистов могут заинтересовать опции autoindent, showmatch, tabstop, shiftwidth, number и list, а также их антиподы.

Наконец, рассмотрим использование опции autowrite. Когда она установлена, vi автоматически записывает содержимое измененного буфера

при вызове команды :n (next), переключающей на следующий редактируемый файл, или при запуске команды оболочки через :!.

# Вызов команд UNIX

Во время редактирования в vi можно отображать или считывать результаты выполнения любой команды UNIX. Восклицательный знак (!) предписывает ex создать оболочку и воспринимать все дальнейшие символы как команду UNIX:

```
:!command
```

Так что если во время редактирования вам захочется посмотреть время или дату, не выходя из vi, то можете ввести:

```
:!date
```

На экране появятся текущие время и дата. Нажмите ENTER для продолжения редактирования файла с того места, где вы остановились.

Если нужно выполнить несколько команд UNIX друг за другом, не возвращаясь к редактированию в vi между ними, то можно создать оболочку командой ex:

```
:sh
```

Для выхода из оболочки и возврата в редактор нажмите CTRL-D.

Вызов UNIX можно сочетать с :read, чтобы результаты выполнения команды UNIX считались в ваш файл. Следующий простой пример:

```
:r !date
```

считывает системную информацию о дате в текст файла. Если перед командой :r поставить адрес строки, то результат ее выполнения будет вставлен в файл по указанному адресу (по умолчанию он появляется в текущей строке).

Предположим, вы редактируете файл и хотите вставить туда четыре телефонных номера из файла phone в алфавитном порядке. Содержимое файла phone таково:

```
Willing, Sue 333-4444
Walsh, Linda 555-6666
Quercia, Valerie 777-8888
Dougherty, Nancy 999-0000
```

Команда:

```
:r !sort phone
```

считает это содержимое после применения к нему фильтра sort:

```
Dougherty, Nancy 999-0000
Quercia, Valerie 777-8888
Walsh, Linda 555-6666
Willing, Sue 333-4444
```

Допустим, вы редактируете файл и хотите вставить текст из другого файла в том же каталоге, но не можете вспомнить имя этого файла. *Можно* поступить неэффективно: выйти из `vi`, выполнить команду `ls`, увидеть правильное имя файла, снова зайти в файл и найти нужное место.

Но то же самое можно сделать за меньшее число шагов:

| Клавиши | Результат |
|---|---|
| `:!ls` | `file1 file2 letter`<br>`newfile practice`<br>Выводит список файлов в текущем каталоге и находит имя нужного файла. Нажмите ENTER для продолжения редактирования. |
| `:r newfile` | `"newfile" 35 lines, 949 characters`<br>Считывает новый файл. |

# Фильтрация текста с помощью команды UNIX

Можно послать фрагмент текста в качестве стандартного ввода в команду UNIX. Ее результат заменит соответствующий текстовый блок в буфере. Фильтровать текст таким образом можно как в `ex`, так и в `vi`. Главное отличие между ними состоит в том, что для `ex` вы выделяете текстовый блок с помощью адресов строк, а в `vi` — как текстовый объект (командами перемещения).

## Фильтрация текста в ex

Первый пример показывает, как фильтровать текст в `ex`. Предположим, список имен в предыдущем примере содержится не в отдельном файле `phone`, а уже в текущем файле в строках с 96-й по 99-ю. Пользователю просто нужно ввести желаемые адреса строк, после чего поставить восклицательный знак и нужную команду UNIX.

Например, команда:

```
:96,99!sort
```

Пропустит строки с 96-й по 99-ю через фильтр `sort` и заменит их на результат команды `sort`.

## Фильтрация текста в vi

В `vi` текст отфильтровывается через команду UNIX, если ввести восклицательный знак, после чего указать любые комбинации клавиш `vi`, задающие текстовый блок, а затем выполняемую команду UNIX. Например:

```
!)command
```

передаст *command* текст до конца предложения.

При использовании этой особенности vi ведет себя несколько необычно:

- Восклицательный знак не появляется на экране сразу же. Если вы вводите сочетания клавиш, определяющие текст для фильтрации, то внизу экрана появляется восклицательный знак, *но символы, задающие фрагмент текста, – нет*.
- Текстовые блоки должны содержать больше одной строки, так что можно использовать только клавиши, перемещающие курсор более чем на строку (G, { }, ( ), [[ ]], +, -). Для усиления эффекта либо перед восклицательным знаком, либо перед текстовым объектом можно указывать число (например, как !10+, так и 10!+ укажут на следующие 10 строк). Объекты, подобные w, не будут работать, пока их количество не станет достаточным для выхода за пределы одной строки. Объект можно задавать косой чертой (/), после которой указать *шаблон* и нажать ENTER. В этом случае в качестве входных данных для команды выступит текст до шаблона.
- Команда обрабатывает строки целиком. Например, если курсор находится на середине строки и вы задаете команде фрагмент до конца предложения, то изменятся все строки, содержащие начало и конец предложения, а не только это предложение[1].
- Существует специальный текстовый объект, который можно использовать с синтаксисом команды, когда текущая строка может определяться с помощью второго восклицательного знака:

  ```
  !!command
  ```

  Не забывайте, что для повторения эффекта число можно ставить либо перед всей последовательностью, либо перед текстовым объектом. Так, чтобы изменить строки с 96-й по 99-ю, как в предыдущем примере, можно поместить курсор на строку номер 96 и ввести:

  ```
  4!!sort
  ```

  или

  ```
  !4!sort
  ```

Другой пример: предположим, в файле есть фрагмент текста, в котором необходимо изменить строчные буквы на прописные. Для смены регистра обработайте этот фрагмент командой tr. В нашем примере через нее будет пропущено второе предложение в блоке:

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

[1] Конечно, всегда найдется исключение. Например, Vim изменит только текущую строку.

<table>
<tr><th>Клавиши</th><th>Результат</th></tr>
<tr><td><code>!)</code></td><td><pre>One sentence after.
~
~
~
!</pre>В последней строке появляется восклицательный знак, приглашая пользователя ввести команду UNIX. <code>)</code> (скобка) указывает, что фильтроваться будет предложение.</td></tr>
<tr><td><code>tr '[:lower:]' '[:upper:]'</code></td><td><pre>One sentence before.
WITH A SCREEN EDITOR YOU CAN SCROLL THE PAGE
MOVE THE CURSOR, DELETE LINES, INSERT CHARACTERS,
AND MORE, WHILE SEEING THE RESULTS OF YOUR EDITS
AS YOU MAKE THEM.
One sentence after.</pre>Введите команду UNIX и нажмите ENTER. Ввод будет заменен на вывод.</td></tr>
</table>

Чтобы повторить предыдущую команду, воспользуйтесь синтаксисом:

```
! object !
```

Иногда полезно послать фрагменты кодированного документа в `nroff`, чтобы их заменил отформатированный текст (также перед отправкой сообщения по электронной почте можно «украсить» текст с помощью программы `fmt`). Помните, что «первоначальный» текст заменяется выводом фильтра. К счастью, если, например, вместо результата вы получаете сообщение об ошибке, то всегда можно отменить команду и восстановить строки.

# Сохранение команд

Часто при редактировании файла приходится вводить одни и те же длинные фразы. В `vi` и `ex` есть несколько способов сохранения длинных цепочек команд как в командном режиме, так и в режиме вставки. Для вызова сохраненной последовательности потребуется лишь нажать несколько клавиш (или даже одну), после чего вся цепочка будет выполнена, как если бы вы вводили команды одну за другой.

## Сокращения слов

Можно определить аббревиатуры, которые будут автоматически расшифровываться в `vi` при вводе в режиме вставки. Для назначения аббревиатуры используется команда `ex`:

```
:ab abbr phrase
```

*abbr* – это аббревиатура для указанной фразы *phrase*. Последовательность символов, составляющих строку, раскрывается в режиме вставки только в случае, если ввести ее как полное слово; если вписать сочетание *abbr* как часть слова, то оно не раскроется.

Пусть в файле practice необходимо набрать текст, содержащий часто встречающуюся фразу, например труднопроизносимое название товара или компании. Команда:

```
:ab imrc International Materials Research Center
```

заменит *International Materials Research Center* на *imrc*. После этого всякий раз при вводе *imrc* в режиме вставки это сокращение будет меняться на полный текст.

| Клавиши | Результат |
|---|---|
| ithe imrc | the International Materials Research Center |

Аббревиатуры расшифровываются после нажатия небуквенно-цифрового символа (например, знака препинания), пробела, возврата каретки или ESC (то есть при возврате в командный режим). При выборе аббревиатур используйте сочетания букв, которые обычно не попадаются в тексте. Если аббревиатура расшифровывается в ненужном месте, ее можно отключить следующей командой:

```
:unab abbr
```

Для отображения всех определенных ранее аббревиатур введите:

```
:ab
```

Символы, составляющую аббревиатуру, не могут встречаться в конце фразы, которую они сокращают. Например, если вызвать команду:

```
:ab PG This movie is rated PG
```

то вы получите сообщение «No tail recursion», и аббревиатура не будет создана. Это сообщение говорит о попытке определить то, что будет многократно разворачивать само себя, создавая бесконечный цикл. Если ввести:

```
:ab PG the PG rating system
```

то можно получить бесконечный цикл, а можно и не получить, однако сообщение об ошибке вы все равно не увидите. Например, предыдущая команда тестировалась на версии UNIX System V, где данный пример работал. В 1990 году в версии для Berkeley это приводило к бесконечному разворачиванию аббревиатуры, то есть фраза

```
the the the the the ...
```

выдавалась до тех пор, пока не произошла ошибка памяти, после чего vi закрылся.

При тестировании мы получили следующие результаты для разных версий vi:

*Solaris* vi

Рекурсия с конца запрещена, а если аббревиатура находится в середине, то сокращение раскрывается однократно.

nvi *1.79*

Оба варианта приводят к превышению внутреннего предела на раскрывание, в результате чего оно прекращается, и nvi выводит сообщение об ошибке.

elvis, *Vim и* vile

Обе формы распознаются и раскрываются только один раз.

Если вы используете vi или nvi для UNIX, мы рекомендуем избегать аббревиатур внутри сокращаемых фраз.

## Использование команды отображения map

Иногда вы многократно используете определенную последовательность команд или применяете очень длинную и сложную последовательность. Чтобы сэкономить время, затраченное на набор команд и их цепочек, командой отображения map их можно закрепить за любой неиспользуемой клавишей.

Работа этой команды во многом аналогична ab, за исключением того, что здесь вы определяете макрос для командного режима vi, а не для режима вставки.

:map *x sequence*

Определить символ *x* как *последовательность* команд редактирования.

:unmap *x*

Отключить *последовательность*, приписанную к *x*.

:map

Вывести список символов, для которых есть отображения в текущий момент.

Перед созданием собственных отображений необходимо знать, какие клавиши не используются в командном режиме и, следовательно, доступны для присвоения им пользовательских команд:

*Буквы*:

g, K, q, V и v

*Управляющие клавиши*:

^A, ^K, ^O, ^W и ^X

*Символы*:

_, *, \, и =

Символ = используется в vi при условии установленного режима Lisp, а в некоторых модификациях – при форматировании текста. Во многих современных версиях vi символ _ эквивалентен команде ^, а в elvis и Vim есть «визуальный режим», где используются клавиши v, V и ^V. Короче говоря, тщательно проверьте свою версию редактора.

В некоторых типах терминалов можно связать последовательности со специальными функциональными клавишами.

С помощью отображений вы сможете создавать простые или сложные цепочки команд. В качестве простого примера определим команду, переставляющую слова в обратном порядке. В vi при следующем положении курсора:

```
you can the scroll page
```

последовательность, которая поставит *the* после *scroll*, – dwelp: удаление слова – dw; перемещение в конец следующего слова – e; перемещение на один пробел направо – l; вставка удаленного слова на этой позиции – p. Сохраним эту цепочку:

```
:map v dwelp
```

Теперь простое нажатие на клавишу v поменяет порядок двух слов в любой момент сеанса редактирования.

## Защита клавиш от интерпретации редактором ex

Обратите внимание, что при определении отображений невозможно использовать клавиши ENTER, ESC, BACKSPACE и DELETE как часть отображаемой команды, поскольку в ex за ними уже закреплены определенные значения. Если вы хотите включить одну из этих кнопок в последовательность команд, то нужно экранировать их обычное значение, нажав CTRL-V. Сочетание ^V появится в отображении как символ ^. Следующие за ^V знаки также не появятся в ожидаемом виде. Например, возврат каретки отображается как ^M, ESC – как ^[, BACKSPACE – как ^H и т. д.

С другой стороны, если вы хотите использовать управляющий символ в качестве символа, на который происходит отображение, то в большинстве случаев нужно лишь нажать буквенную клавишу, удерживая CTRL. Например, для отображения последовательности на ^A достаточно ввести:

```
:map CTRL-A sequence
```

Однако существуют три управляющих символа, которые необходимо экранировать с помощью ^V: ^T,^W и ^X. То есть для отображения последовательности на ^T следует набрать:

```
:map CTRL-V CTRL-T sequence
```

Использование CTRL-V относится ко всем командам ex, а не только к map. Это означает, что в команде сокращения или замены можно использовать возврат каретки. Например, сокращение:

```
:ab 123 one^Mtwo^Mthree
```

развернется в:

```
one
two
three
```

(Здесь последовательность CTRL-V ENTER показана как ^M. Именно так она будет выглядеть на вашем экране.)

Используя команду g, можно добавить новые строки в различные места документа. Например, команда:

```
:g/^Section/s//As you recall, in^M&/
```

вставит фразу «As you recall...» в отдельной строке перед каждой строкой, начинающейся со слова *Section*. Символ & воспроизводит текст, соответствующий шаблону поиска.

К сожалению, один символ всегда имеет специальное значение в командах ex, даже если его «спрятать» при помощи CTRL-V. Напомним, что вертикальная черта (|) имеет специальное значение – разделителя нескольких команд ex. Вы не сможете использовать ее для отображений в режиме вставки.

Теперь, когда вы научились использовать CTRL-V для защиты отдельных клавиш внутри команд ex, вы сможете определить эффективные отображения для последовательностей команд.

## Пример сложного отображения

Предположим, у нас есть глоссарий с пунктами следующего вида:

```
map - an ex command which allows you to associate
a complex command sequence with a single key.
```

Мы хотим преобразовать этот список в формат troff, чтобы он выглядел примерно так:

```
.IP "map" 10n
An ex command...
```

Лучший способ определить сложное отображение – проделать одно исправление вручную, выписывая каждое нажатие клавиши. Затем надо воссоздать эти нажатия в качестве отображения. Нам нужно:

1. Вставить макрос MS для абзаца с отступом в начале строки. Также следует вставить первую кавычку (I.IP ").
2. Нажать ESC, чтобы выйти из режима вставки.
3. Переместиться в конец первого слова (e) и добавить вторую кавычку, после которой ввести пробел и размер отступа (a" 10n).

4. Нажать ENTER для перехода на новую строку.
5. Нажать ESC, чтобы завершить режим вставки.
6. Удалить дефис и два окружающих его пробела (3x), после чего сделать следующее слово прописным (~).

Если эти действия нужно повторить несколько раз, то это довольно нудная работа.

С помощью :map можно сохранить всю последовательность, чтобы затем вызывать ее одним нажатием клавиши:

```
:map g I.IP "^[ea" 10n^M^[3x~
```

Обратите внимание, что как ESC, так и ENTER необходимо «спрятать» с помощью CTRL-V. Последовательность ^[ возникает после ввода CTRL-V и ESC. Если ввести CTRL-V и ENTER, то появится последовательность ^M.

После этого простое нажатие на g произведет целую серию исправлений. Если связь медленная, то вы сможете увидеть, как происходят эти правки, одна за другой. Если же канал широкий (или вы редактируете файл локально), то это будет выглядеть как волшебство.

Не расстраивайтесь, если ваша первая попытка задать отображение для клавиши не удастся. Даже небольшая ошибка в определении отображения может привести к результатам, далеким от ожидаемых. Введите u для отмены правки и попробуйте все сначала.

## Другие примеры отображения клавиш

Следующие примеры показывают некоторые разумные сочетания клавиш, которые можно использовать в отображениях:

1. Добавление текста при каждом переходе в конец слова:

   ```
   :map e ea
   ```

   В большинстве случаев единственная причина для перехода на конец слова – добавление текста. Эта отображенная последовательность автоматически переводит пользователя в режим вставки. Обратите внимание, что отображаемая клавиша e уже имеет значение в vi. Можно переопределять используемую в редакторе клавишу, но пока работает отображение, обычное действие этой клавиши недоступно. В нашем случае это не страшно, так как E часто равносильна e.

2. Перестановка двух слов:

   ```
   :map K dwElp
   ```

   Ранее в главе мы уже обсуждали аналогичную последовательность, однако сейчас нужно использовать E (здесь, как и во всех примерах ниже, мы полагаем, что команда e отображена на ea). Не забывайте, что курсор должен стоять в начале первого из двух переставляемых слов. К сожалению, из-за команды l эта последовательность (как

и приведенная ранее) не будет работать, если слова стоят в конце строки, так как во время выполнения последовательности курсор попадет на конец строки и не может стоять правее. Так что приведем решение получше:

```
:map K dwwP
```

Вместо `w` можно использовать `W`.

3. Сохранение файла и переход к следующему из набора:

```
:map q :w^M:n^M
```

Обратите внимание, что можно отображать на клавиши команды `ex`, но убедитесь, что после каждой команды `ex` стоит возврат каретки. Подобная последовательность облегчит перемещение между файлами и пригодится, если вы открыли множество небольших файлов одной командой `vi`. Отображение на букву `q` поможет запомнить, что действие последовательности похоже на выход «quit».

4. Поместить код `troff` для жирного шрифта вокруг слова:

```
:map v i\fB^[e\fP^[
```

Здесь предполагается, что курсор стоит в начале слова. Сначала вы переходите в режим вставки, затем вводите код для жирного шрифта. Следует отметить, что в командах отображения не нужно вписывать две обратные косые черты для получения одной. После этого посредством ввода «спрятанного» ESC происходит переход в командный режим. Наконец, к концу слова приписывается завершающий код `troff`, после чего вы возвращаетесь в командный режим. Обратите внимание, что при добавлении кода к концу слова последовательность `ea` не нужна, так как она уже отображена на одну букву `e`. Это показывает, что отображенные последовательности могут содержать другие отображения (возможность использовать вложенные последовательности отображений управляется опцией `vi remap`, которая обычно включена).

5. Поместить вокруг слова HTML-код жирного шрифта, даже если курсор расположен не в начале слова:

```
:map V lbi<B>^[e</B>^[
```

Эта последовательность напоминает предыдущую: помимо того, что здесь присутствует HTML, а не `troff`, она использует `lb` для выполнения дополнительной задачи – помещения курсора в начало слова. Если курсор находится в середине слова, то команда `b` переместит его на начало. Однако если курсор уже находится в начале слова, команда `b` перенесет его на начало предыдущего слова. Чтобы исключить эту возможность, перед `b` стоит `l`. Можно определить варианты этой последовательности, заменяя `b` на `B` и `e` на `Ea`. Во всяком случае, команда `l` помешает выполнению последовательности, если курсор стоит в конце строки (это можно обойти, добавив пробел).

6. Найти и удалить скобки вокруг слова или фразы[1] по всему документу:

```
:map = xf)xn
```

Здесь предполагается, что сначала командой /( вы найдете открывающую скобку, а затем нажмете ENTER.

Если необходимо удалить скобки, используйте команду map следующим образом: удалите открывающую скобку с помощью x, найдите закрывающую с помощью f), удалите ее с помощью x и повторите поиск открывающей скобки командой n.

Если вы не хотите удалять скобки (например, когда они стоят правильно), не используйте отображенную команду, а вместо этого нажмите n для поиска новой открывающей скобки.

Последовательность отображения из этого примера можно изменить так, чтобы она обрабатывала не скобки, а кавычки.

7. Сделать всю строку комментарием C/C++:

```
:map g I/* ^[A */^[
```

Эта последовательность добавляет сочетание /* в начало строки, а к концу приписывает */. То же самое можно сделать, отобразив команду подстановки:

```
:map g :s;.*;/* & */;^M
```

Здесь с помощью .* мы ищем соответствие всей строке, затем при ее воспроизведении (используя &) помещаем ее между символами комментария. Обратите внимание, что в качестве разделителей используются точки с запятой, чтобы не пришлось экранировать / в комментариях.

8. Безопасно повторить большую вставку:

```
:map ^J :set wm=0^M.:set wm=10^M
```

Как уже говорилось в главе 2, в vi иногда возникает проблема, связанная с повторением больших вставок текста при включенной опции wrapmargin. Команда map поможет с этим справиться. Она временно отключает wrapmargin (установив ее равной нулю), выполняет команду повторения, а затем восстанавливает значение опции. Обратите внимание, что в этой последовательности отображения присутствуют одновременно команды vi и ex.

В предыдущем примере, несмотря на то, что ^J является командой vi (перемещает курсор вниз на одну строку), эта клавиша безопасно отображается, поскольку ее действие аналогично команде j. Существует много клавиш, которые либо выполняют аналогичные другим действия, либо редко используются. Однако все-таки следует ознакомиться с командами vi, перед тем как смело использовать клавиши в качестве отображаемых, отказываясь тем самым от их обычных значений.

---

[1] Взято из статьи Вальтера Зинтца (Walter Zintz) в *UNIX World*, апрель 1990.

## Отображение клавиш в режиме вставки

Как правило, отображения работают только в командном режиме. Действительно, в режиме вставки клавиши имеют свои обычные значения и им нельзя приписать команды. Тем не менее, если добавить в команду `map` восклицательный знак, то можно отменить обычное значение клавиши и сделать отображение в режиме вставки. Эта функция полезна при работе в режиме вставки, когда нужно быстро оттуда выйти, выполнить команду и вернуться обратно.

Предположим, вы набрали слово и забыли выделить его курсивом (поместить в кавычки и т. п.). Можете задать отображение типа:

```
:map! + ^[bi<I>^[ea</I>
```

Теперь при вводе + в конце слова оно выделится HTML-кодом курсива. Сам знак плюса в тексте не появится.

Определенная подобным образом последовательность выходит из режима вставки (`^[`), возвращает курсор в начало слова, вставляет первую часть кода (`bi<I>`), снова выходит из режима вставки (`^[`) и двигает курсор вперед на место вставки второй части кода (`ea</I>`). Поскольку отображенная последовательность начинается и заканчивается в режиме вставки, после такого выделения слова можно продолжать ввод текста.

А вот другой пример: допустим, нужно поставить двоеточие в конце предыдущей строки. Это может сделать следующая отображенная последовательность[1]:

```
:map! % ^[kA:^[jA
```

Теперь, введя `%` в любом месте текущей строки, вы припишите двоеточие к концу предыдущей. Эта команда выходит в командный режим, перемещает курсор вверх на строку и добавляет к ней двоеточие (`^[kA:`), затем снова выходит из режима вставки, перемещает курсор на строку ниже и включает режим вставки (`^[jA`).

Обратите внимание, что в предыдущих командах отображения мы использовали редко встречающиеся символы (`%` и `+`). Если символ отображен для режима вставки, вы уже не сможете вводить его как текст.

Чтобы вернуть знаку его прямое значение, используйте команду:

```
:unmap! x
```

где *x* – это символ, который прежде был отображен для режима вставки (хотя `vi` раскроет символ *x* в командной строке при его вводе, и это будет похоже на снятие отображения раскрытого текста, он произведет корректную отмену отображения символа).

Отображения в режиме вставки больше подходят для привязки символьных строк к специальным клавишам, не используемым для других

---

[1] Из статьи Walter Zintz в журнале *UNIX World*, апрель 1990.

целей. Оно особенно полезно при работе с программируемыми функциональными клавишами.

## Отображения функциональных клавиш

Многие терминалы имеют программируемые функциональные клавиши[1] (которые удачно эмулируются на эмуляторах терминалов на современных растровых рабочих станциях). В специальном установочном режиме (setup) терминала эти клавиши можно настроить на печать одного или нескольких любых символов. Однако клавиши, запрограммированные в установочном режиме терминала, будут работать только на этом терминале. Кроме того, при этом ограничиваются действия программ, использующих функциональные клавиши в своих целях.

`ex` позволяет отобразить эти клавиши в виде чисел с помощью:

```
:map #1 commands
```

для функциональной клавиши номер 1 и так далее (это возможно, потому что редактор имеет доступ к той записи в базах данных `terminfo` или `termcap`, где есть информация об обнаруженном терминале, и знает, какая Escape-последовательность обычно соответствует определенной функциональной клавише).

Как и в случае обычных кнопок, по умолчанию отображения относятся к командному режиму, но с помощью команды `map!` функциональной клавише можно задать два различных значения: одно для командного режима, а другое – для режима вставки. Например, при верстке HTML можно присвоить функциональным клавишам коды изменения начертания шрифта:

```
:map #1 i<I>^[
:map! #1 <I>
```

Если вы находитесь в командном режиме, то первая функциональная клавиша переведет вас в режим вставки, введет символы `<I>` и вернет в командный режим. Если пользователь уже находится в режиме вставки, клавиша просто добавит HTML-код из трех символов.

Если последовательность содержит `^M` ( возврат каретки), то нажмите CTRL-M. Например, чтобы функциональная клавиша 1 была доступна для отображения, запись в терминальной базе данных для вашего терминала должна содержать примерно следующее определение для `k1`:

```
k1=^A@^M
```

В свою очередь, определение:

```
^A@^M
```

должно появляться в выводе при нажатии этой кнопки.

[1] На современной клавиатуре PC им соответствуют функциональные клавиши F1–F12. – *Прим. науч. ред.*

Чтобы увидеть значение функциональной клавиши, воспользуйтесь командой od (octal dump) с опцией -c (показывать каждый символ). Теперь после нажатия этой функциональной клавиши введите ENTER, а затем CTRL-D, чтобы команда od вывела информацию. Например:

```
$ od -c
^[[[A
^D
0000000 033 [ [ A \n
0000005
```

Здесь функциональная клавиша посылает компьютеру Escape, две левых квадратных скобки и A.

## Отображения для других специальных клавиш

Многие клавиатуры содержат специальные кнопки, дублирующие команды vi, такие как HOME, END, PAGE UP и PAGE DOWN. Если описание терминала в terminfo или termcap является полным, то vi сможет распознать эти клавиши. Однако если это не так, то можно использовать команду map, чтобы сделать их доступными в vi. Обычно эти клавиши посылают компьютеру Escape-последовательность – символ Escape, за которым следует строка из одного или нескольких символов. Чтобы предотвратить обработку Escape в ex, нужно нажать ^V перед специальной клавишей в команде отображения. Например, чтобы кнопке HOME на клавиатуре IBM PC поставить в соответствие подходящий эквивалент из vi, можно определить следующее отображение[1]:

```
:map CTRL-V HOME 1G
```

На экране это будет выглядеть так:

```
:map ^[[H 1G
```

Похожие команды отображения выглядят следующим образом:

| | | |
|---|---|---|
| :map CTRL-V END G | *покажет* | :map ^[[Y G |
| :map CTRL-V PAGE UP ^F | *покажет* | :map ^[[V ^F |
| :map CTRL-V PAGE DOWN ^B | *покажет* | :map ^[[U ^B |

Возможно, вам захочется поместить эти отображения в файл .exrc. Обратите внимание, что если специальная клавиша вырабатывает длинную экранирующую последовательность (содержащую несколько непечатаемых символов), ^V защитит от интерпретации только первый Escape-символ, и отображение не будет работать. Вместо того чтобы просто нажимать ^V, а затем нужную клавишу, вам придется отыскать полную Escape-последовательность (возможно, в руководстве по вашему терминалу) и ввести ее вручную, используя ^V там, где это необходимо для защиты Escape-символов от интерпретации.

---

[1] Скорее следовало бы написать :map ^[[H ^, поскольку авторский вариант команды переносит курсор в начало файла, а не строки. – *Прим. науч. ред.*

Если вы пользуетесь различными типами терминалов (например, консолью на PC и `xterm`), то не надо ожидать, что отображения, подобные представленным выше, всегда будут работать. Для таких случаев Vim предоставляет переносимый способ для описания таких отображений клавиш:

```
:map <Home> 1G
```
*Введите шесть символов: < H o m e > (Vim)*

## Отображение нескольких клавиш ввода

Отображение нажатий на несколько клавиш не ограничивается функциональными кнопками. Можно также отображать последовательности нажатий обычных клавиш. Это поможет при вводе определенных типов текста, например XML или HTML.

Опираясь на Джерри Пика (Jerry Peek), соавтора изданной O'Reilly книги *Learning the Unix Operating System*, приводим некоторые из команд `:map`, облегчающие ввод разметки XML. (Строки, начинающиеся с двойной кавычки, являются комментариями. Позже об этом будет рассказано в разделе «Комментарии в скриптах ex» на стр. 142.)

```
" ADR: need this
:set noremap
" bold:
map! =b </emphasis>^[F<i<emphasis role="bold">
map =B i<emphasis role="bold">^[
map =b a</emphasis>^[
" Move to end of next tag:
map! =e ^[f>a
map =e f>
" footnote (tacks opening tag directly after cursor in text-input mode):
map! =f <footnote>^M<para>^M</para>^M</footnote>^[kO
" Italics ("emphasis"):
map! =i </emphasis>^[F<i<emphasis>
map =I i<emphasis>^[
map =i a</emphasis>^[
" paragraphs:
map! =p ^[jo<para>^M</para>^[O
map =P O<para>^[
map =p o</para>^[
" less-than:
map! *l &lt;
...
```

Если пользоваться этими командами, то для ввода сноски вам нужно перейти в режим вставки и набрать `=f`. При этом `vi` поставит открывающие и закрывающие теги и поместит курсор между ними в режиме вставки:

```
All the world's a stage.<footnote>
<para>
█
</para>
</footnote>
```

Стоит ли говорить, что при работе над книгой подобные макросы оказались чрезвычайно полезны.

## @-функции

Именованные буферы предоставляют еще один способ создания «макросов» – сложных последовательностей команд, которые можно выполнять всего несколькими нажатиями клавиш.

Если в тексте написать команду (это может быть последовательность `vi` или команда `ex`, *перед которой стоит двоеточие*), а затем удалить ее в именованный буфер, то содержимое этого буфера можно выполнить с помощью команды `@`. Например, в новой строке введите следующее:

```
cwgadfly CTRL-V ESC
```

На экране это будет выглядеть так:

```
cwgadfly^[
```

Снова нажмите ESC для выхода из режима вставки и удалите строку в буфер `g`, введя `"gdd`. Теперь всякий раз при помещении курсора в начало слова и вводе `@g` это слово будет заменено на *gadfly*.

Поскольку `@` интерпретируется как команда `vi`, точка (`.`) повторит всю последовательность, даже если в буфере содержится команда `ex`. Сочетание `@@` повторит последнюю `@`, а `u` или `U` можно использовать для отмены действия `@`.

Это был простой пример. @-функции полезны, поскольку их можно применять для самых необычных команд. Особенно это удобно при одновременном редактировании нескольких файлов, так как команды содержатся в своих именованных буферах и их можно вызывать из любого редактируемого файла. @-функции также полезны в сочетании с командами глобальной замены, про которые рассказывалось в главе 6.

## Выполнение содержимого буферов в ex

В режиме `ex` также можно выполнить текст, сохраненный в буфере. В этом случае необходимо ввести команду `ex`, удалить ее в именованный буфер, после чего использовать команду `@` из приглашения с двоеточием. Например, введем следующие строки:

```
ORA publishes great books.
ORA is my favorite publisher.
1,$s/ORA/O'Reilly Media/g
```

Поместив курсор в последнюю строку, удалите ее в буфер `g` командой `"gdd`, затем поставьте курсор на первую строку (`kk`) и выполните содержащиеся в буфере команды из командной строки с двоеточием (`:@g` ENTER). На экране вы должны увидеть следующее:

```
O'Reilly Media publishes great books.
O'Reilly Media is my favorite publisher.
```

В некоторых версиях vi символ * в командной строке ex трактуется так же, как и @. Кроме того, если после команд @ или * стоит *, то команда берется из буфера (неименованного) по умолчанию.

# Использование скриптов ex

Отдельные команды ex, такие как отображения, аббревиатуры и прочие, можно использовать только из vi. Если эти команды сохранить в файле .exrc, они будут автоматически выполняться при запуске редактора. Файл, содержащий выполняемые команды, называется *скриптом*.

Команды, содержащиеся в обычном скрипте .exrc, нельзя использовать вне vi, но можно сохранять в скрипт другие команды ex, а затем применять его к одному или нескольким файлам. Во внешних скриптах в основном используются команды подстановки.

При создании текстовых файлов скрипты ex могут использоваться при проверке правильности используемых терминов и даже орфографии по всему набору документов. Предположим, вы выполнили UNIX-команду spell с двумя файлами, и она выдала следующий список опечаток:

```
$ spell sect1 sect2
chmod
ditroff
myfile
thier
writeable
```

Как часто бывает, проверка орфографии выявила несколько технических терминов и специальных случаев, которые не смогла распознать. Однако она нашла и две настоящих ошибки в написании.

Поскольку мы проверили два файла сразу, нам неизвестно, в каком из них содержатся эти ошибки и где они расположены. Хотя есть способы их определить, да и для двух ошибок в двух файлах такая задача необременительна, но легко представить, насколько возрастет объем работы у бедного корректора или наборщика при проверке большого количества файлов.

Чтобы облегчить задачу, можно написать скрипт ex со следующими командами:

```
%s/thier/their/g
%s/writeable/writable/g
wq
```

Пусть мы сохранили эти строки в файл exscript. Этот скрипт можно запустить из vi с помощью команды:

```
:so exscript
```

Его также можно применить к файлу прямо в командной строке и отредактировать файлы sect1 и sect2 следующим образом:

```
$ ex -s sect1 < exscript
$ ex -s sect2 < exscript
```

Ключ -s, следующий за вызовом ex, – это POSIX-способ приказать редактору подавлять обычный вывод на терминал[1].

Если бы скрипт был больше, чем в этом простом примере, то мы бы уже сэкономили некоторое время. Однако вы можете спросить, можно ли как-то избежать повторения этого процесса для каждого из редактируемых файлов. Конечно да, мы можем написать скрипт оболочки, который не только включает, но и обобщает вызов ex, так что его можно использовать для любого числа файлов.

## Циклы в скриптах командной строки

Вы, наверное, уже знаете, что оболочка – это не только интерпретатор командной строки, но и язык программирования. Чтобы вызвать ex с несколькими файлами, мы используем простую команду оболочки, называемую циклом for. Он позволяет выполнять последовательность команд для каждого аргумента скрипта. (Возможно, цикл for является наиболее часто используемым куском кода для начинающих осваивать программирование в оболочке. Его стоит научиться использовать, даже если вы не собираетесь писать другие скрипты.)

Цикл for имеет следующий синтаксис:

```
for variable in list
do
   command(s)
done
```

Например:

```
for file in $*
do
   ex - $file < exscript
done
```

(Отступ в команде можно не делать; у нас он показан для удобства чтения.) После создания этого скрипта сохраним его в файле с именем correct и сделаем его исполняемым командой chmod (если вы не знакомы с командой chmod и действиями, необходимыми для ее попадания в пути поиска UNIX, почитайте книгу *Learning the UNIX Operating System*, опубликованную в O'Reilly). Теперь введем:

```
$ correct sect1 sect2
```

---

[1] Обычно для этой цели ex использует одиночный знак минуса. Для обратной совместимости поддерживаются оба способа.

Цикл `for` в `correct` присвоит переменной `file` каждый аргумент (каждый файл в списке, определенном как `$*`, что означает *все аргументы*) и выполнит скрипт `ex` с содержимым каждой из этих переменных.

Понять принцип работы циклов `for` будет проще, если взять пример с более очевидным выводом. Посмотрим на скрипт для переименования файлов:

```
for file in $*
do
   mv $file $file.x
done
```

Предположим, что этот скрипт расположен в исполняемом файле с именем `move`. Мы можем сделать следующее:

```
$ ls
ch01 ch02 ch03 move
$ move ch??                    Только файлы глав (ch...)
$ ls                           Проверим результаты
ch01.x ch02.x ch03.x move
```

Если подойти к делу творчески, то можно по-другому переписать скрипт, переименовывающий файлы:

```
for nn in $*
do
   mv ch$nn sect$nn
done
```

В таком скрипте следует указывать в командной строке не имена, а номера файлов:

```
$ ls
ch01 ch02 ch03 move
$ move 01 02 03
$ ls
sect01 sect02 sect03 move
```

Циклу `for` не обязательно требуется `$*` (с любыми аргументами) в качестве списка значений. Можно указать список в явном виде. Например:

```
for variable in a b c d
```

присваивает *variable* значения *a*, *b*, *c* и *d* по очереди. Также можно подставлять вывод другой команды.

Например:

```
for variable in `grep -l "Alcuin" *`
```

присвоит *variable* по очереди имя каждого файла, в котором `grep` найдет строку *Alcuin*. (`grep -l` выводит имена тех файлов, содержимое которых отвечает шаблону, без вывода самих строк, где есть совпадение.)

Если список не указан:

```
for variable
```

переменной *variable* будет присвоен каждый из аргументов командной строки, как было в нашем первом примере. Вообще говоря, это равносильно не такой команде:

```
for variable in $*
```

а такой:

```
for variable in "$@"
```

Последнее выражение имеет немного другое значение. Символ `$*` раскрывается в `$1`, `$2`, `$3` и т. д., а последовательность из четырех символов `"$@"` – в `"$1"`, `"$2"`, `"$3"` и т. д. Кавычки защищают от последующей интерпретации специальных символов[1].

Давайте вернемся к основной идее и нашему первоначальному скрипту:

```
for file in $*
do
   ex - $file < exscript
done
```

Использование двух скриптов – скрипта оболочки и команды `ex` – может показаться немного безвкусным. Ведь фактически оболочка сама дает способ включить скрипт редактирования в скрипт оболочки.

## Документы heredoc

В скриптах оболочки оператор `<<` означает взять в качестве ввода команды следующие далее строки вплоть до указанного маркера. (Часто это называют *документом heredoc*.) С помощью этого синтаксиса мы можем включить в `correct` собственные команды редактирования следующим образом:

```
for file in $*
do
ex - $file << end-of-script
g/thier/s//their/g
g/writeable/s//writable/g
wq
end-of-script
done
```

Строка-маркер `end-of-script` совершенно произволен. Единственное требование состоит в том, что это должен быть набор символов, не содержащихся во вводе, чтобы оболочка могла распознать конец документа heredoc. Также его *нужно* поместить на начало строки. По соглашению многие пользователи указывают конец документа heredoc строкой `EOF` или `E_O_F`, обозначающей конец файла.

[1] Например, если значение какой-либо из переменных `$1`, `$2`, `$3` (имя файла-аргумента в примере выше) содержит пробел, то в цикле по `"$@"` оно будет представлено одним значением (что правильно), а в цикле по `$*` – несколькими разными значениями (что неправильно). – *Прим. науч. ред.*

Каждый из представленных способов имеет свои достоинства и недостатки. Если вы хотите проделать серию однократных правок и готовы переписывать скрипт каждый раз, то документ heredoc дает эффективное решение этой задачи.

Однако более гибкий способ состоит в записи команд редактирования в отдельном от скрипта оболочки файле. Например, можно договориться, что вы всегда размещаете команды редактирования в файле с именем `exscript`. В этом случае скрипт `correct` нужно написать всего один раз. Его можно хранить в специальном «утилитном» каталоге (который содержится в пути поиска) и использовать в любое время.

## Сортировка текстовых блоков: пример скрипта ex

Предположим, вам нужно расположить в алфавитном порядке определения глоссария, содержащиеся в размеченном `troff`-файле. Каждый термин начинается с макроса `.IP`. Вдобавок, каждый пункт заключен в паре макросов `.KS/.KE` (чтобы каждый термин выступал в блоке со своим определением, то есть чтобы они не оказались на разных страницах). Файл глоссария имеет примерно следующий вид:

```
.KS
.IP "TTY_ARGV" 2n
The command, specified as an argument vector,
that the TTY subwindow executes.
.KE
.KS
.IP "ICON_IMAGE" 2n
Sets or gets the remote image for icon's image.
.KE
.KS
.IP "XV_LABEL" 2n
Specifies a frame's header or an icon's label.
.KE
.KS
.IP "SERVER_SYNC" 2n
Synchronizes with the server once.
Does not set synchronous mode.
.KE
```

Если прогнать его строки через UNIX-команду `sort`, то можно привести файл в алфавитный порядок. Однако нам не нужно сортировать все строки подряд. Отсортированы должны быть только термины глоссария, а каждое определение должно остаться нетронутым и следовать за соответствующим термином. Оказывается, каждый текстовый блок можно рассматривать как целое, если объединить блок в одной строке. Вот первая версия нашего скрипта `ex`:

```
g/^\.KS/,/^\.KE/j
%!sort
```

Каждый пункт глоссария расположен между макросами .KS и .KE. Команда ex j объединяет строки (ее эквивалент в vi называется J), поэтому первая команда объединяет каждую запись глоссария в одну «строку». Следующая команда отсортирует файл, и на выходе получится что-то вроде этого:

```
.KS .IP "ICON_IMAGE" 2n Sets or gets ... image. .KE
.KS .IP "SERVER_SYNC" 2n Synchronizes with ... mode. .KE
.KS .IP "TTY_ARGV" 2n The command, ... executes. .KE
.KS .IP "XV_LABEL" 2n Specifies a ... icon's label. .KE
```

Сейчас строки отсортированы по записям. К сожалению, макросы и текст в них перемешались (многоточия [...] обозначают пропущенный текст). Теперь надо как-то вставить переносы, чтобы «разъединить» строки. Это можно сделать, изменив скрипт ex следующим образом: *перед* объединением нужно пометить точки объединения, а затем заменить эти метки на перевод строки. Расширенный скрипт ex имеет вид:

```
g/^\.KS/,/^\.KE/-1s/$/@@/
g/^\.KS/,/^\.KE/j
%!sort
%s/@@ /^M/g
```

Первые три команды дадут следующий результат:

```
.KS@@ .IP "ICON_IMAGE" 2nn@@ Sets or gets ... image. @@ .KE
.KS@@ .IP "SERVER_SYNC" 2nn@@ Synchronizes with ... mode. @@ .KE
.KS@@ .IP "TTY_ARGV" 2nn@@ The ... vector, @@ that ... .@@ .KE
.KS@@ .IP "XV_LABEL" 2nn@@ Specifies a ... icon's label. @@ .KE
```

Обратите внимание на дополнительный пробел после @@. Эти пробелы появляются от команды j, поскольку она преобразует каждый перевод строки в пробел.

Первая команда помечает первоначальные переносы строк символами @@. Конец блока (после .KE) помечать не надо, поэтому в первой последовательности стоит команда -1, перемещающая курсор от конца блока вверх на одну строку. Четвертая команда восстанавливает переносы строк, заменяя метки (и один пробел) на новые строки. После этого файл будет рассортирован по блокам.

## Комментарии в скриптах ex

Возможно, вы будете использовать этот скрипт неоднократно в разных ситуациях. В случае сложных скриптов, таких как в нашем примере, разумно добавлять комментарии, чтобы другим пользователям (или даже вам) было легче его реконструировать. В скриптах ex все, что следует за двойной кавычкой, будет игнорироваться во время выполнения, поэтому двойная кавычка отмечает начало комментария. Комментарии можно писать в отдельных строках. Они также могут располагаться в конце любой команды, не считающей кавычки своей частью. (Напри-

мер, кавычка имеет значение в командах отображения и оболочки. Подобные строки не могут заканчиваться комментарием.)

Кроме использования комментариев можно указывать полное имя команды, что в vi обычно не делается из-за экономии времени. Наконец, после добавления пробелов наш ex-скрипт станет гораздо более удобочитаемым:

```
" Mark lines between each KS/KE block
global /^\.KS/,/^\.KE/-1 s /$/@@/
" Now join the blocks into one line
global /^\.KS/,/^\.KE/ join
" Sort each block--now really one line each
%!sort
" Restore the joined lines to original blocks
% s /@@ /^M/g
```

Удивительно, но команда substitute в ex не работает, хотя полные имена других команд работают.

## За пределами ex

Если изложенный материал пробудил ваш аппетит к более продвинутому редактированию, то необходимо знать, что в UNIX есть гораздо более мощные редакторы по сравнению с ex, такие как потоковый редактор sed и язык обработки данных awk. Также существует чрезвычайно популярный язык программирования Perl. Если хотите узнать больше об этих программах, обратитесь к книгам O'Reilly: *sed & awk*, *Effective awk Programming*, *Learning Perl*[1] и *Programming Perl*[2].

# Редактирование исходного кода программы

Все рассмотренные выше функции не отличали обычный текст от программного кода. Однако существует ряд дополнительных функций, которые будут интересны именно программистам. Таковыми являются контроль отступов, поиск начала и конца процедуры и использование ctags.

Дальнейшее изложение основано на документации, предоставленной Mortice Kern Systems для их прекрасной реализации vi для систем DOS и Windows, доступной в виде части MKS Toolkit или отдельной программы MKS Vi. Воспроизводится с разрешения Mortice Kern Systems.

---

[1] Рэндал Шварц, Том Феникс и Брайан Д. Фой «Изучаем Perl», 5-е издание. – Пер. с англ. – Символ-Плюс, 2009.

[2] Ларри Уолл, Том Кристиансен, Джон Орвант «Программирование на Perl», 3-е издание. – Пер. с англ. – Символ-Плюс, 2002.

## Контроль за отступами

Программный код несколько отличается от обычного текста. Один из аспектов касается использования отступов в коде, где они показывают логическую структуру программы, то есть способ объединения строк в блоки. vi дает возможность автоматически контролировать отступы. Чтобы использовать это, введите команду:

```
:set autoindent
```

Теперь при установке отступа в строке пробелами или табуляцией такие же отступы установятся во всех последующих строках. После нажатия на ENTER в конце первой строки с отступом курсор переместится на новую строку и автоматически отступит на то же расстояние, что и в предыдущей.

Если вы программист, то сможете сэкономить много времени на установке отступов, особенно если у вас есть несколько уровней отступов.

При вводе текста с включенной опцией autoindent нажатие CTRL-T в начале строки переведет курсор на следующий уровень отступа, а CTRL-D вернет его обратно.

Отметим, что CTRL-T и CTRL-D должны вводиться в режиме вставки, в отличие от многих других команд, которые набираются в командном режиме.

Есть два других варианта применения команды CTRL-D[1]:

`^ ^D`

Если ввести `^ ^D` (^ CTRL-D), то vi переместит курсор назад на начало строки, но только для текущей строки. Следующая строка начнется на текущем установленном автоматически уровне отступа. Это может быть полезно при вводе команд препроцессора C в программах на C/C++.

`0 ^D`

При нажатии `0 ^D` vi переместит курсор обратно на начало строки. Вдобавок к этому текущий уровень отступа станет нулевым, и следующая строка будет уже без отступа[2].

Попробуйте опцию autoindent при вводе программного кода. Она упростит работу по упорядочиванию отступов. А в некоторых случаях поможет избежать ошибок, например в программе на C, где требуется одна закрывающая фигурная скобка (}) для каждого возврата на предыдущий уровень отступа.

При установке отступов в программе также могут быть полезными команды >> и <<. По умолчанию >> смещает строку на восемь пробелов

---

1 В elvis это работать не будет.

2 В документация nvi 1.79 эти две команды перепутаны, но программа ведет себя именно так, как мы описали.

вправо (то есть добавляет к отступу эти восемь пробелов), а `<<` – на восемь пробелов влево. Например, переместив курсор на начало строки и набрав `>>`, вы увидите, как строка сместится вправо. Если ввести `<<`, то строка вернется на прежнее место.

Можно сместить несколько строк, указав их количество перед `>>` или `<<`. Например, поместите курсор на первую строку большого абзаца и введите `5>>`. После этого первые пять строк в этом абзаце сдвинутся вправо.

По умолчанию величина сдвига равна восьми пробелам (влево или вправо). Это значение можно изменить следующей командой:

```
:set shiftwidth=4
```

Вы увидите, что целесообразно установить `shiftwidth` равным ширине табуляции.

При установке отступов `vi` пытается умничать. Как правило, при наличии текста с отступом из восьми пробелов он вставит в файл символы табуляции, потому что обычно табуляция заменяется восемью пробелами. Это принято в UNIX, однако легко заметить, что при вводе табуляции и печати файла на принтере UNIX заменит ее на восемь пробелов.

При желании можно изменить способ представления табуляции на экране, поменяв опцию `tabstop`. Например, если у вас большой уровень вложенности блоков, можно взять четыре символа для табуляции, чтобы строки помещались на строке. Такую замену произведет следующая команда:

```
:set tabstop=4
```

Менять ширину табуляции не рекомендуется. Хотя `vi` и будет отображать файл при любом значении `tabstop`, в остальных программах UNIX знаки табуляции будут[1] заменяться на восемь пробелов.

Хуже того: смесь пробелов, табуляций и необычных позиций табуляции сделает ваш файл совершенно нечитаемым в других программах, таких как программа постраничного вывода `more`, или при распечатке. Восьмисимвольная табуляция – это одна из данностей UNIX, и вам придется с этим смириться.

Иногда отступы не работают как надо, потому что символ табуляции на проверку оказывается несколькими пробелами. Как правило, на экране табуляция и пробел отображаются одинаково, и их не различить. Однако можно выполнить следующую команду:

```
:set list
```

---

[1] По крайней мере, могут: в некоторых приложениях UNIX, особенно современных, ширину табуляции также можно регулировать. Но в любом случае изменение значения `tabstop` в `vi` выльется в лишнюю работу. – *Прим. науч. ред.*

При этом отображение сменится таким образом, что табуляция будет показана в виде управляющего символа ^I, а конец строки – как $. В этом случае можно распознать реальные пробелы и увидеть лишние в конце строки. Временным эквивалентом этого является команда :l. Например, последовательность:

```
:5,20 l
```

отобразит символы табуляции и конца строки в строках с 5-й по 20-ю.

## Специальная команда поиска

Символы (, [, { и < могут быть открывающими скобками. Когда курсор находится на одном из этих символов, нажатие клавиши % переместит его с открытой скобки вперед на соответствующую закрывающую: ), ], } или > с учетом обычного правила для вложенных скобок[1]. Например, если поставить курсор на первую ( в выражении:

```
if ( cos(a[i]) == sin(b[i]+c[i]) )
{
   printf("cos and sin equal!\n");
}
```

и нажать %, то курсор перепрыгнет на скобку в конце этой строки. Это закрывающая скобка, соответствующая открывающей, на которой стоял курсор.

Аналогично, если курсор стоит на одном из символов закрывающих скобок, то нажатие % переместит его назад на соответствующую открывающую скобку. Например, поставьте курсор на закрывающую скобку после строки с printf и нажмите %.

vi достаточно умен и может сам найти символ скобки. Если курсор стоит не на символе скобки, то при вводе % редактор будет искать вперед по текущей строке до первого попавшегося символа открывающей или закрывающей скобки и переместит курсор на соответствующую ей скобку! Например, если в предыдущем примере курсор в первой строке стоит на символе >, то % отыщет открывающую круглую скобку и переместит курсор на закрывающую.

Символ поиска % позволяет не только быстро перемещаться вперед и назад по программе, но и проверять вложенность скобок в программном коде. Например, если поставить курсор на первую { в начале функции C, то ввод % переместит курсор на ту }, которая, по идее, завершает функцию. Если это не так, значит, где-то содержится ошибка. Когда соответствующая } не найдена, vi выдаст звуковой сигнал.

[1] Из всех протестированных версий соответствие < и > при использовании % наблюдалось только в nvi. В vile (и Vim. – *Прим. науч. ред.*) можно задать опцию, где определяется, какие пары символов будут соответствовать друг другу при нажатии %.

Другой способ искать скобки, соответствующие друг дружке, состоит во включении следующей опции:

```
:set showmatch
```

Включение showmatch (или ее аббревиатуры sm) отличается от % тем, что работает в режиме вставки. При вводе ) или }[1] курсор на короткое время переместится на соответствующую ( или {, а затем вернется на исходную позицию. Если соответствие не будет найдено, вы услышите звуковой сигнал. Если соответствующая скобка не попадает на текущий экран, vi продолжит работу без звуковых оповещений. Vim 7.0 и его более поздние версии могут подсвечивать соответствующие друг другу скобки, используя загружаемый по умолчанию плагин matchparen.

## Использование тегов

Обычно исходный код больших программ на C или C++ разбит на несколько файлов. Иногда очень сложно отследить, какой из них содержит описание нужной функции. Чтобы упростить дело, команда UNIX ctags может использоваться совместно с командой :tag из vi.

UNIX-версии ctags поддерживают языки C и, чаще всего, Pascal и Fortran 77. Иногда они даже поддерживают язык ассемблера. C++, как правило, не поддерживается. Доступны другие версии, которые могут вырабатывать файлы tags для C++ и других языков и типов файлов. За более подробной информацией обратитесь к разделу «Улучшенные теги» на стр. 154.

Сначала нужно вызвать команду ctags в командной строке UNIX. Ее цель – создать информационный файл, который в дальнейшем будет использоваться в vi для определения, в каком файле какие функции содержатся. По умолчанию этот файл называется tags. В vi команда следующего вида:

```
:!ctags file.c
```

создаст в текущем каталоге файл с именем tags, содержащий информацию о функциях, определенных в file.c. Следующая команда

```
:!ctags *.c
```

создаст файл tags, описывающий все файлы исходного кода C в текущем каталоге.

Теперь предположим, что файл tags содержит информацию обо всех файлах исходного кода, из которых собрана программа на C. Также допустим, что нужно найти и отредактировать функцию из этой программы, но вы не знаете, где располагается эта функция. Команда в vi:

```
:tag name
```

---

[1] В elvis, Vim и vile showmatch также показывает соответствие для квадратных скобок [ и ].

ищет в файле `tags` файл, содержащий определение функции *name*. Затем эта команда считывает этот файл и помещает курсор на строку с определением *name*. При этом необязательно знать, какой файл вы редактируете. Нужно лишь решить, какую функцию следует изменить.

Возможностями тегов можно пользоваться и в командном режиме `vi`. Поместите курсор на идентификатор, который вы хотите отыскать, и введите `^]`. `vi` произведет поиск по тегам и переместит вас в файл, где этот идентификатор определен. Будьте осторожны с положением курсора – `vi` считает «словом» то, что начинается с текущей позиции курсора, а не все слово, где стоит курсор.

Если вы пытаетесь использовать команду `:tag` для считывания файла, но не сохранили текущий файл со времени его последнего изменения, `vi` не позволит перейти на новый файл. Нужно либо сохранить текущий файл командой `:w` и только после этого вызвать `:tag`, либо выполнить следующую команду:

```
:tag! name
```

которая пересилит нежелание `vi` отбрасывать изменения.

Версия `vi` для Solaris тоже поддерживает *стеки* тегов, однако эта возможность не отражена в документации к Solaris. Поскольку многие, если не большинство, версии `vi` не поддерживают стеки тегов, мы переместили обсуждение этой функции в раздел «Стеки тегов» на стр. 157, где подробнее познакомим вас с ними.

# 8

# Представляем клоны vi

## Знакомьтесь: Даррелл, Даррелл и Даррелл

Существует несколько свободно распространяемых модификаций редактора `vi`. Приложение D дает ссылку на веб-сайт, где перечислены все его известные разновидности, а в части II подробно рассказывается о Vim. Часть III касается трех других популярных модификаций `vi`:

- `nvi` версия 1.79 Кейта Бостича (Keith Bostic) (глава 16)
- `elvis` версия 2.2.0 Стива Киркендалла (Steve Kirkendall) (глава 17)
- `vile` версия 9.6.4 Кевина Буттнера (Kevin Buettner), Тома Дики (Tom Dickey), Пола Фокса (Paul Fox) и Кларка Моргана (Clark Morgan) (глава 18)

Все они были написаны либо из-за ограничений на свободное распространение исходного кода `vi`, что делало невозможным перенос редактора в не-UNIX окружение и изучение его кода, либо из-за того, что `vi` (или другой клон!) для UNIX не имел требуемой функциональности. Например, `vi` для UNIX часто имеет предел максимальной длины строки и в нем нельзя редактировать бинарные файлы. (Главы, посвященные соответствующим программам, расскажут больше об их истории.)

В каждой программе есть множество расширений по сравнению с `vi` для UNIX. Зачастую одни и те же дополнения, возможно, реализованные по-разному, есть у нескольких модификаций. Чтобы не повторять в каждой главе описание одной и той же общей для всех программ функции, мы вывели их обсуждение сюда. Можете рассматривать эту главу как рассказ о функциональности модификаций редактора. При этом глава, посвященная каждому клону, рассказывает, как они работают.

Темы этой главы представлены далее в том же порядке, но более подробно в части II, посвященной Vim, и гораздо более компактно – в главах части III. В данной главе мы рассмотрим следующие темы:

*Многооконное редактирование*

Это возможность разделить экран (терминала) на несколько «окон» и/или использовать их в графическом окружении. При этом в окнах можно редактировать разные файлы или держать разные виды одного файла. Возможно, это самое главное улучшение оригинального `vi`.

*Графические интерфейсы*

Все клоны, за исключением `nvi`, могут быть скомпилированы с поддержкой интерфейса X Window. Если в вашей системе работает X, то использование GUI-версии может быть предпочтительнее разделения экрана `xterm` (или другого эмулятора терминала). Как правило, GUI-версии обеспечены и другими полезными функциями, такими как полосы прокрутки и разные шрифты. Также могут поддерживаться GUI из других операционных систем.

*Расширенные регулярные выражения*

Все модификации `vi` позволяют искать текст при помощи регулярных выражений, которые похожи или совпадают с используемыми в команде `egrep` UNIX.

*Улучшенные теги*

Как уже описывалось в разделе «Использование тегов» на стр. 147, при построении базы данных для поиска в файлах можно использовать программу `ctags`. Модификации `vi` также позволяют создавать «стеки тегов» (tag stack), последовательно сохраняя ваше текущее местоположение во время поиска по тегам. Впоследствии можно вернуться к сохраненному месту. Последовательность местоположений сохраняется в порядке «первым пришел, последним ушел» (LIFO), формируя стек положений.

Некоторые авторы модификаций `vi` и автор по крайней мере одной разновидности `ctags` пришли к соглашению, определяющему стандартную форму улучшенной версии формата `ctags`. В частности, сейчас проще использовать функциональность тегов в программах, написанных на C++, где разрешена перегрузка имен функций.

*Улучшенные возможности редактирования*

Во всех модификациях `vi` есть возможность редактирования командной строки `ex`, функция «бесконечной отмены», строки произвольной длины и содержимого (поддержка любых 8-разрядных символов), инкрементный поиск, опция прокрутки экрана слева направо вместо переноса при чтении длинных строк, индикаторы режима и другие функции.

*Помощь программисту*

Некоторые редакторы предоставляют возможность не выходить из программы при выполнении типичного цикла разработки ПО «редактирование-компиляция-отладка».

*Подсветка синтаксиса*

В elvis, Vim и vile можно сделать так, чтобы отдельные части файла отображались в разных цветах и шрифтах. Это особенно удобно при редактировании программного кода.

# Многооконное редактирование

Возможно, самое важное улучшение стандартного vi, предоставляемое его модификациями, – это способность редактировать файлы в нескольких окнах. Оно позволяет легко править сразу несколько файлов и «вырезать и вставлять» текст из одного файла в другой посредством копирования и вставки.

В клонах не обязательно разделять экран, чтобы копировать и вставлять текст между файлами. Только оригинальный vi очищает буферы для хранения фрагментов текста при переключении между файлами.

В каждом редакторе существуют две основные концепции, лежащие в основе реализации многооконности: буферы и окна.

В *буфере* содержится редактируемый текст. Он может быть взят из файла или быть совершенно новым, который нужно будет записать в файл. Каждому файлу соответствует только один связанный с ним буфер.

*Окно* обеспечивает просмотр буфера, позволяя видеть и изменять содержащийся в нем текст. С одним буфером может быть связано несколько окон. Изменения, сделанные в буфере в одном из окон, отражаются и в других окнах, в которых открыт этот буфер. С буфером может не быть связано ни одного окна. В этом случае с ним ничего нельзя делать, разве что открыть его в окне позже. Если закрыть последнее из окон, связанных с некоторым буфером, то «закроется» и файл. Если буфер был изменен, но не сохранен на диск, то редактор может запретить (или не запретить) закрыть последнее связанное с ним окно.

При создании нового окна редактор разделяет текущий экран. В большинстве программ в новом окне вы увидите редактируемый файл. После этого можно переключиться в окно, где вы хотите редактировать другой файл, и дать редактору команду о переходе к редактированию файла в этом окне. Каждый редактор предоставляет доступ к командам vi и ex для перемещения по окнам вперед и назад, а также позволяет менять размер окон, скрывать и показывать их.

О многооконном редактировании в Vim рассказывается в главе 11. В каждой из глав части III, посвященной очередному редактору, мы покажем примеры разделения экрана (на примере одних и тех же двух файлов) и расскажем, как разбить экран и перемещаться между окнами.

# Графические интерфейсы

У elvis, Vim и vile есть версии с графическими пользовательскими интерфейсами (GUI), дающие все преимущества растровых экранов и мыши. Кроме поддержки X Window в UNIX поддерживаются также Microsoft Windows и другие оконные системы. В табл. 8.1 перечислены GUI, доступные для разных модификаций vi.

*Таблица 8.1. Доступные GUI*

| Редактор | Терминал | X11 | Microsoft Windows | OS/2 | BeOS | Macintosh | Amiga | QNX | OpenVMS |
|---|---|---|---|---|---|---|---|---|---|
| Vim | • | • | • | • | • | • | • | | |
| nvi | • | | | | | | | | |
| elvis | • | • | • | • | | | | | |
| vile | • | • | • | • | • | | | • | • |

# Расширенные регулярные выражения

Метасимволы, доступные в vi при поиске и замене посредством регулярных выражений, описаны ранее в главе 6 в разделе «Метасимволы, используемые при поиске по шаблону» на стр. 95. Во всех модификациях vi есть разновидности расширенных регулярных выражений, которые либо опциональны, либо доступны всегда. Как правило, они идентичны (или почти идентичны) аналогам из программы egrep. К сожалению, разновидности расширенных выражений в разных модификациях vi несколько отличаются.

Чтобы показать возможности расширенных регулярных выражений, представим их на примере nvi. Раздел «Расширенные регулярные выражения» на стр. 198 описывает расширенные регулярные выражения Vim, а главы части III, посвященные модификациям vi, также описывают синтаксисы расширенных регулярных выражений соответствующих редакторов, причем примеры не повторяются.

Согласно стандарту POSIX, расширенные регулярные выражения nvi называются Extended Regular Expressions (ERE). Чтобы включить их, воспользуйтесь командой `set extended` либо в файле .nexrc, либо из приглашения ex.

Кроме стандартных метасимволов, описанных в главе 6, и скобковых выражений POSIX, про которые мы упоминали в разделе «Выражения в скобках в стандарте POSIX» на стр. 97 той же главы, здесь доступны следующие метасимволы:

|

Разделяет несколько возможных вариантов. Например, `a|b` соответствует или *a*, или *b*. Однако этот метасимвол не ограничивается одиночными знаками: `house|home` соответствует либо слову *house*, либо *home*.

(...)

Используется для создания группы, к которой потом можно применять другие операторы регулярных выражений. Например, house|home можно сократить (если не упростить) до ho(use|me). Оператор * можно применять к тексту в скобках: (house|home)* соответствует *home*, *homehouse*, *househomehousehouse* и т. д.

При включенной опции extended текст, сгруппированный скобками, ведет себя аналогично тексту, заключенному между \(...\) в обычном vi: строка, соответствующая совпадению, может быть воспроизведена в части замены в команде подстановки с помощью /1, /2 и т. д. В этом случае для ввода обычной открывающей скобки без специального значения используются символы \(.

+

Соответствует *одному* или нескольким предшествующим регулярным выражениям. Это может быть либо одиночный символ, либо группа символов, заключенных в скобки. Обратите внимание на различия между + и *. Символ * может означать пустое место, а у + должно быть хотя бы одно совпадение. Например, ho(use|me)* соответствует *ho*, равно как и *home*, и *house*, но ho(use|me)+ не соответствует *ho*.

?

Обозначает ноль или одно вхождение предшествующего регулярного выражения. Так можно указать на «необязательный» текст, который может присутствовать, а может и нет. Например, free?d соответствует и *fred*, и *freed*.

{...}

Определяет *интервальное выражение*. Интервальное выражение описывает счетное число повторений. В следующих описаниях *n* и *m* обозначают целые числа:

{*n*}

Обозначает ровно *n* повторений предыдущего регулярного выражения. Например, (home|house){2} соответствует *homehome*, *homehouse*, *househome*, *househouse* и больше ничему.

{*n*,}

Соответствует *n* или более повторениям предыдущего регулярного выражения. Можно считать, что эта команда задает «как минимум *n*» повторений.

{*n*,*m*}

Соответствует числу повторений от *n* до *m*. Ограничение важно, так как оно определяет, сколько именно текста заменится командой подстановки[1].

---

[1] Операторы *, + и ? можно свести к {0,}, {1,} и {0,1} соответственно, но использовать первые гораздо удобнее. Кроме того, с точки зрения истории UNIX интервальные выражения были введены в обращение позже.

Если опция extended не установлена, то эта же функция реализуется в nvi при помощи \{ и \}.

# Улучшенные теги

Программа «Exuberant ctags» – это разновидность ctags, способная сделать намного больше, чем ctags в UNIX. Она создает файл tags расширенного формата, который делает процесс поиска по тегам более гибким и мощным. Мы опишем версию Exuberant первой, так как она поддерживается большинством модификаций vi.

В этом разделе также описаны стеки тегов, позволяющие сохранять несколько посещенных с помощью команд :tag или ^] позиций в файлах. Стеки тегов поддерживаются во всех разновидностях vi.

## Exuberant ctags

Программа Exuberant ctags была создана Дарреном Хибертом (Darren Hiebert) и на момент написания этой книги вышла в версии 5.7[1]. Ее домашняя страница – *http://ctags.sourceforge.net/*. Следующий список возможностей программы взят из файла README дистрибутива ctags:

- Генерация тегов для *всех* типов синтаксических элементов языков C и C++, включая названия классов, определения макросов, названия перечислений (enum), перечисляемые значения (то, что находится внутри enum), определения функций (методов), объявления/прототипы функций (методов), члены структур и поля классов, названия структур (struct), определения типов (typedef), имена объединений (union) и переменных. Уфф!
- Поддержка кода C и C++.
- Также поддерживаются 29 других языков, включая C# и Java.
- Программа очень устойчива при анализе кода, и ее гораздо сложнее одурачить кодом, содержащим директивы препроцессора #if.
- Возможность ее использования для распечатывания читабельного списка выбранных объектов, обнаруженных в исходном коде.
- Поддержка генерации файлов тегов в стиле GNU Emacs (etags).
- Работа под Amiga, Cray, MS-DOS, Macintosh, OS/2, QDOS, QNX, RISC OS, UNIX, VMS и Windows версий от 95 до XP. На веб-сайте доступны некоторые скомпилированные бинарные файлы.

Exuberant ctags генерирует файлы tags в формате, который мы опишем ниже.

[1] На момент подготовки русскоязычного издания – 5.8. – *Прим. науч. ред.*

## Новый формат тегов

Как правило, файл tags имеет три поля, отделенные друг от друга табуляцией: имя тега (обычно это идентификатор), исходный файл, содержащий тег, и указание, где искать идентификатор. Это указание может быть либо простым номером строки, либо поисковым шаблоном, соответствующим установленной опции vi nomagic и заключенным между обратными косыми чертами или знаками вопроса. Кроме того, файл tags всегда отсортирован.

Этот формат генерируется программой UNIX ctags. Фактически многие версии vi принимали в поле поискового шаблона *любую* команду (зияющая дыра в безопасности). Более того, имелась недокументированная особенность: если строка заканчивалась точкой с запятой, после которой стояли двойные кавычки (;"), то весь текст после этих символов игнорировался (двойными кавычками открывается комментарий, как в файлах .exrc).

Новый формат имеет обратную совместимость с традиционным. Первые три поля те же самые: тег, имя файла и шаблон поиска. Exuberant ctags генерирует только шаблоны поиска, а не произвольные команды. После разделителя ;" ставятся расширенные атрибуты. Каждый атрибут отделен от следующего символом табуляции и содержит два подполя, разделенных двоеточием. Первое подполе содержит ключевое слово, описывающее атрибут, второе – его значение. В табл. 8.2 перечислены поддерживаемые ключевые слова.

*Таблица 8.2. Ключевые слова в расширенном ctags*

| Ключевое слово | Значение |
|---|---|
| kind | Значение – одна буква, указывающая на лексический тип тега. Это может быть f для функции, v для переменной и т. д. Поскольку именем атрибута по умолчанию является kind, любая отдельная буква может обозначать тип тега (например, f для функции). |
| file | Для «статических» (то есть локальных для файла) тегов. Значением должно быть имя файла. |
| | Если значению присвоена пустая строка (просто file:), то считается, что оно совпадает со вторым полем (имя файла). Эта особенность была добавлена отчасти для компактности и для обеспечения простого способа поддержки файлов tags, расположенных не в текущем каталоге. Значение поля, содержащего имя файла, всегда задается относительно того каталога, где расположен сам файл tags. |
| function | Для локальных синтаксических конструкций. Значение – имя функции, где они определены. |
| struct | Для полей в struct. Значением служит имя структуры. |

*Таблица 8.2 (продолжение)*

| Ключевое слово | Значение |
|---|---|
| enum | Для элементов перечислений enum, где значение – имя перечисления. |
| class | Для членов-функций и членов-переменных в C++. Значение – имя класса. |
| scope | Предназначено в основном для членов-функций класса C++. Обычно оно имеет значение private для частных закрытых членов или опускается для публичных членов, чтобы пользователь мог ограничить поиск тегов только публичными членами. |
| arity | Для функций. Определяет число ее аргументов. |

Если поле не содержит двоеточия, то считается, что оно относится к типу kind. Приведем примеры:

```
ARRAYMAXED      awk.h     427;"   d
AVG_CHAIN_MAX   array.c   38;"    d    file:
array.c         array.c    1;"    F
```

ARRAYMAXED – это макрос C #define, определенный в awk.h. AVG_CHAIN_MAX – тоже макрос C, но он используется только в array.c. Третья строка немного другая: это тег для самого файла исходного кода! Он генерируется с помощью опции -i F Exuberant ctags и позволяет задать команду :tag array.c. Кроме того, удобно поместить курсор над именем файла и перейти к этому файлу с помощью команды ^] (например, если вы редактируете Makefile и хотите перейти к определенному файлу).

В той части атрибута, где задается значение, обратная косая черта, табуляция, возврат каретки и символы перевода строки должны быть представлены как \\, \t, \r и \n соответственно.

Расширенные файлы tags могут содержать некоторое количество инициализирующих тегов, начинающихся с !_TAG_. Эти теги обычно отсортированы в начале файла и полезны при определении программы, создавшей файл. Exuberant ctags генерирует следующее:

```
!_TAG_FILE_FORMAT    2 /extended format; --format=1 will not append ;" to lines/
!_TAG_FILE_SORTED        1 /0=unsorted, 1=sorted, 2=foldcase/
!_TAG_PROGRAM_AUTHOR     Darren Hiebert /dhiebert@users.sourceforge.net/
!_TAG_PROGRAM_NAME       Exuberant Ctags //
!_TAG_PROGRAM_URL        http://ctags.sourceforge.net   /official site/
!_TAG_PROGRAM_VERSION    5.7     //
```

Некоторые редакторы могут использовать эти специальные теги для реализации различных функций. Например, Vim уделяет внимание тегу !_TAG_FILE_SORTED и при поиске в файле tags будет использовать бинарный поиск вместо линейного, используемого для неотсортированных файлов.

Если вы пользуетесь файлами tags, рекомендуем установить Exuberant ctags.

## Стеки тегов

Команды ex :tag и ^] в режиме vi предоставляют ограниченные возможности для поиска идентификаторов на основе информации, содержащейся в файле tags. Однако во всех модификациях редактора есть расширение этих возможностей, а именно – способность поддерживать *стек* положений тегов. Всякий раз при вызове команд ex :tag или ^] в режиме vi перед поиском указанного тега редактор сохраняет текущее положение в файле. После этого можно вернуться к сохраненному положению, используя (как правило) команду vi ^T или команду ex.

Ниже мы расскажем о стеке тегов Solaris vi и приведем некоторые примеры. Стеки тегов Vim описываются в разделе «Стеки тегов» на стр. 302. О методах работы со стеками тегов, применяемых в других редакторах, рассказывается в соответствующих главах части III.

## Solaris vi

Как ни удивительно, версия vi для Solaris поддерживает стеки тегов. Возможно, не столь удивительно то, что эта функция совершенно не документирована на страницах *ex*(1) и *vi*(1) руководства по Solaris. Мы свели сведения о работе со стеком тегов в Solaris vi в табл. 8.3, 8.4 и 8.5. Стек тегов в Solaris vi весьма прост[1].

*Таблица 8.3. Команды тегов в vi для Solaris*

| Команда | Действие |
|---|---|
| ta[g][!] *tagstring* | Редактирует файл, содержащий *tagstring*, согласно файлу tags. Восклицательный знак заставляет vi переключиться в новый файл, даже если текущий буфер изменен, но не сохранен. |
| po[p][!] | Вытаскивает один элемент из стека тегов. |

*Таблица 8.4. Команды тегов для командного режима vi для Solaris*

| Команда | Действие |
|---|---|
| ^] | Отыскивает в файле tags местоположение идентификатора, на котором стоит курсор, и переходит на это место. Если включен стек тегов, то текущее положение автоматически помещается в стек. |
| ^T | Возвращает к предыдущему месту в стеке тегов, то есть вытаскивает оттуда один элемент. |

[1] Эта информация была получена экспериментально. Ваши результаты могут отличаться.

*Таблица 8.5. Опции для управления тегами в vi для Solaris*

| Опция | Действие |
|---|---|
| taglength, tl | Управляет количеством значимых символов в теге при его поиске. Значение по умолчанию, равное нулю, указывает, что значимы все символы. |
| tags, tagpath | Значением является список файлов, где нужно искать теги. По умолчанию это "tags/ usr/lib/tags". |
| tagstack | Когда эта опция установлена равной true (истина), vi посылает в стек каждое местоположение. Чтобы ее выключить, введите :set notagstack. |

## Exuberant ctags и Vim

Чтобы было понятно, как использовать стеки тегов, представим небольшой пример с использованием Exuberant ctags и Vim.

Предположим, вы работаете с программой, использующей функцию GNU getopt_long, про которую вам необходимо узнать больше.

GNU getopt состоит из трех файлов: getopt.h, getopt.c и getopt1.c.

Сначала создадим файл tags, а затем начнем редактировать основную программу, расположенную в main.c:

```
$ ctags *.[ch]
$ ls
Makefile   getopt.c   getopt.h   getopt1.c   main.c   tags
$ vim main.c
```

<table>
<tr><th>Клавиши</th><th>Результат</th></tr>
<tr><td>/getopt_</td><td><pre>/* option processing. ready, set, go! */
for (optopt = 0, old_optind = 1;
     (c = getopt_long(argc, argv, optlist, optab, NULL)) != EOF;
     optopt = 0, old_optind =    optind) {
       if (do_posix)
               opterr = TRUE;</pre>
Редактируем main.c и переходим к вызову getopt_long.</td></tr>
<tr><td>^]</td><td><pre>int
getopt_long (int argc, char *const *argv, const char *options,
             const struct option *long_options, int *opt_index)
{
 return _getopt_internal (argc, argv, options, long_options,
                          opt_index, 0);
}

"getopt1.c" 192L, 4781C</pre>
Выполняем поиск тегов по getopt_long. Vim переместит вас в getopt1.c и поставит курсор на определение getopt_long.</td></tr>
</table>

Оказывается, getopt_long является функцией-«оберткой» для _getopt_internal. Вы помещаете курсор на _getopt_internal и снова выполняете поиск тегов.

<table>
<tr><th>Клавиши</th><th>Результат</th></tr>
<tr><td>3jf_ ^]</td><td>

```
int
_getopt_internal (int argc, char *const *argv, const char *optstring,
          const struct option *longopts, int *longind, int long_only)
{
  int result;

  getopt_data.optind = optind;
  getopt_data.opterr = opterr;

  result = _getopt_internal_r (argc, argv, optstring, longopts,
                               longind, long_only, &getopt_data);

  optind = getopt_data.optind;
"getopt.c" 1225L, 33298C
```

Вы перешли к getopt.c. Чтобы отыскать больше информации о struct option, поместите курсор на option и опять выполните поиск по тегам.</td></tr>
<tr><td>jfo; ^]</td><td>

```
   one). For long options that have a zero `flag' field, `getopt'
   returns the contents of the `val' field. */

struct option
{
  const char *name;
  /* has_arg can't be an enum because some compilers complain about
     type mismatches in all the code that assumes it is an int. */
  int has_arg;
  int *flag;
  int val;
};

/* Names for the values of the `has_arg' field of `struct option'. */

"getopt.h" 177L, 6130C
```

Редактор перешел на определение struct option в getopt.h. Теперь можно почитать комментарии, где объясняется использование данной структуры.</td></tr>
<tr><td>:tags</td><td>

```
# TO tag         FROM line  in file/text
1 1 getopt_long        310  main.c
2 1 _getopt_internal    67  getopt1.c
3 1 option            1129  getopt.c
```

Команда :tags в Vim показывает стек тегов.</td></tr>
</table>

Нажатие ^T три раза переместит вас обратно в main.c, откуда вы начали. С помощью тегов очень удобно перемещаться во время редактирования исходного кода.

## Улучшенные возможности

Во всех разновидностях vi есть дополнительные возможности, позволяющие редактировать текст проще и быстрее:

*Редактирование командной строки ex*

Способность редактировать команды в режиме ex во время их ввода, включая сохранение истории команд ex. Также сюда входят функции завершения имен файлов и прочие, например завершение команд или опций.

*Отсутствие ограничений на длину строки*

Редактирование строки произвольной длины. Также сюда входит возможность редактировать файлы, содержащие любые 8-разрядные символы.

*Бесконечные отмены*

Последовательная отмена всех сделанных в файле изменений.

*Инкрементный поиск*

Поиск текста прямо во время ввода шаблона поиска.

*Правая/левая прокрутка*

Возможность не переносить длинные строки, позволяя им вылезать за пределы экрана.

*Визуальный режим*

Способность выбирать произвольные смежные фрагменты текста, над которыми будет совершена какая-либо операция.

*Индикаторы режима*

Видимый указатель активного режима (командного или режима вставки), а также указатели текущих строки и столбца.

## История командной строки, автозавершение

Пользователи оболочек csh, tcsh, ksh, zsh и bash знают, что если есть возможность вызывать предыдущие команды и легко редактировать их и выполнять заново, то работа становится намного более эффективной.

Для пользователей редакторов это так же верно, как и для пользователей командной строки. К сожалению, в vi для UNIX невозможно сохранять и восстанавливать команды ex.

Этот недостаток исправлен во всех модификациях vi. Сохранение истории команд и их повторный вызов реализованы в них разными способами, но все эти механизмы работают и добавляют удобства в работе.

Вдобавок к истории команд во всех этих редакторах есть некоторое подобие автозавершения команды. Например, после ввода начала имени файла и нажатия определенной клавиши (например, табуляции) редактор завершает имя файла за вас. Кроме автозавершения имени файла в некоторых редакторах есть автозавершение и для других объектов. Подробности, касающиеся Vim, можно найти в разделе «Ключевые слова и завершение слов по словарю» на стр. 293. Про другие редакторы подробно рассказывается в соответствующих главах части III.

## Строки произвольной длины и двоичные данные

Во всех модификациях vi есть поддержка строк произвольной длины[1]. Старые версии часто имели предел, равный примерно 1000 символам в строке, а более длинные строки обрезались.

Кроме того, все они прозрачны для 8-разрядных символов, то есть в них можно редактировать файлы, содержащие любые 8-разрядные символы. При необходимости можно редактировать даже двоичные и исполняемые файлы. Иногда это может пригодиться. Редактору можно сообщить о том, что файл двоичный, а можно и не сообщать.

nvi

Автоматически поддерживает двоичные данные. Не требуется ни специальных опций командной строки, ни ex.

elvis

Под UNIX не делает отличий между двоичными и прочими файлами. В других системах использует файл elvis.brf для задания опции binary, чтобы избежать проблем с восприятием переноса строки (файл elvis.brf и режим отображения hex описаны в разделе «Интересные особенности» на стр. 374).

*Vim*

Не ограничивает длину строки. Если не установлен режим binary, Vim ведет себя как nvi и автоматически поддерживает двоичные данные. Однако при редактировании двоичного файла лучше воспользоваться опцией -b командной строки или :set binary. При этом устанавливаются некоторые другие опции Vim, облегчающие редактирование двоичных файлов.

vile

Автоматически поддерживает двоичные данные и не требует ни специальных опций командной строки, ни ex.

Наконец, одна важная деталь. Традиционный vi всегда сохраняет файл с добавленным в конец символом переноса строки. Таким образом, при редактировании двоичного файла к нему может добавиться лишний

[1] Вернее, до максимального значения типа long в C – 2147483647 (на 32-разрядном компьютере).

символ, что приведет к неприятностям. По умолчанию nvi и Vim совместимы с vi и добавляют перенос строки. В Vim этого можно избежать, если установить опцию binary. elvis и vile никогда не добавляют перенос строки.

## Бесконечная отмена

vi для UNIX позволяет отменить только последнее изменение или восстановить строку в состоянии до внесения в нее каких-либо изменений. Во всех разновидностях редактора есть «бесконечная отмена» – способность откатывать действия пользователя одно за другим вплоть до состояния, когда файл не подвергался *никаким* правкам.

## Инкрементный поиск

При использовании *инкрементного поиска* редактор перемещает курсор по всему файлу, отыскивая соответствия в тексте *по мере ввода* шаблона поиска. После нажатия на ENTER поиск завершается[1]. Если раньше вы подобного не встречали, то поначалу это может привести в замешательство. Однако когда вы привыкните к этому, то удивитесь, как раньше обходились без такого поиска.

В nvi, Vim и elvis инкрементный поиск присутствует в качестве опции, а vile использует даже две специальные команды режима vi. vile можно скомпилировать с отключенным инкрементным поиском, но по умолчанию он включен. В табл. 8.6 показаны опции, предоставляемые каждым из редакторов.

*Таблица 8.6. Инкрементный поиск*

| Редактор | Опция | Команда | Действие |
|---|---|---|---|
| nvi | searchincr | | Во время ввода курсор перемещается по файлу, всегда попадая на первый символ текста, где обнаружилось соответствие. |
| Vim | incsearch | | Во время ввода курсор перемещается по файлу. Vim подсвечивает текст, соответствующий уже введенному. |
| elvis | incsearch | | При вводе курсор перемещается по файлу. elvis подсвечивает текст, который соответствует введенному. |
| vile | | ^X S, ^X R | При вводе курсор перемещается по файлу, всегда попадая на первый символ текста, где обнаружилось соответствие. Команда ^X S выполняет инкрементный поиск вперед по файлу, а ^X R – в обратном направлении. |

[1] В Emacs инкрементный поиск был всегда.

## Прокрутка влево-вправо

По умолчанию vi и большая часть его модификаций переносят длинные строки. Таким образом, одна логическая строка файла может занимать несколько видимых строк на экране.

Иногда предпочтительнее сделать так, чтобы длинная строка не переносилась, а выходила за правый край экрана. Если поставить курсор на такую строку и переместить его направо, то весь экран «прокрутится» в сторону. Эта функция доступна во всех разновидностях vi. Как правило, числовая опция задает, на сколько нужно прокручивать экран, а булевская определяет, переносятся строки или исчезают за границами экрана. В vile также есть командные клавиши, позволяющие смещать в сторону весь экран. В табл. 8.7 приведены примеры использования горизонтальной прокрутки в разных редакторах.

*Таблица 8.7. Прокрутка в стороны*

| Редактор | Величина прокрутки | Опция | Действие |
|---|---|---|---|
| nvi | sidescroll = 16 | leftright | По умолчанию выключена. Если включить эту опцию, то длинные строки выходят за границы экрана. За один раз экран прокручивается на 16 символов влево или вправо. |
| elvis | sidescroll = 8 | wrap | По умолчанию выключена. Если включить опцию, то длинные строки выходят за границы экрана. За один раз экран прокручивается на 8 символов влево или вправо. |
| Vim | sidescroll = 0 | wrap | По умолчанию выключена. Если включить эту опцию, то длинные строки выходят за границы экрана. Если sidescroll равен нулю, то каждая прокрутка помещает курсор в середину экрана. В остальных случаях экран прокручивается на установленное количество символов. |
| vile | sideways = 0 | linewrap | По умолчанию выключена. Если включить опцию, то длинные строки будут переноситься. По умолчанию они уходят за границы экрана. Длинные строки отмечаются слева и справа символами < и >. Если sideways равен нулю, то каждая прокрутка перемещает экран на ⅓. В остальных случаях экран прокручивается на установленное количество символов. |
| | | horizscroll | По умолчанию включена. Если опция включена, то курсор, переместившись за пределы экрана, сместит весь экран. Если опция не установлена, то смещается только текущая строка. Это удобно на медленных дисплеях. |

В vile есть две дополнительные команды: ^X ^R и ^X ^L. Они смещают экран вправо и влево соответственно, при этом оставляя курсор на своем месте. Сместить экран так, чтобы позиция курсора вышла за его пределы, нельзя.

## Визуальный режим

Как правило, операции в vi затрагивают единицы текста, такие как строки, слова или символы, или разделы текста от текущего положения курсора до позиции, определенной командой поиска. Например, d/^} удалит символы до строки, начинающейся с правой фигурной скобки. В elvis и vile есть механизм, позволяющий явно указать область текста, к которой будет применена операция. В частности, можно выделить прямоугольный блок текста и применить операцию ко всему тексту в этой области. Детали, касающиеся этого режима в Vim, см. в разделе «Движение в визуальном режиме» на стр. 197. Информацию о других редакторах см. в соответствующих главах части III.

## Индикаторы режима

Как вы уже знаете, в vi есть два режима – режим вставки и командный режим. При этом при взгляде на экран нельзя определить, какой режим активен. Кроме того, часто бывает полезно знать, в какой части файла вы находитесь, не обращаясь к ^G или команде ex :=.

Для решения этих проблем выделены две опции: showmode и ruler. Во всех разновидностях vi имена и значения этих опций одинаковы, а опция showmode есть даже в версии редактора для Solaris.

В табл. 8.8 перечислены особенности каждого из редакторов.

*Таблица 8.8. Индикаторы позиции и режима*

| Редактор | ruler показывает | showmode показывает |
|---|---|---|
| nvi | Строку и столбец | Индикаторы режимов вставки, изменения и замены, а также командного режима |
| elvis | Строку и столбец | Индикаторы режимов вставки и командного режима |
| Vim | Строку и столбец | Индикаторы режимов вставки и замены, а также визуального режима |
| vile | Строку, столбец и процент от всего файла | Индикаторы режимов вставки, замены и перезаписи |
| vi | Н/Д | Отдельные индикаторы для режимов открытия, ввода, вставки, добавления, изменения, замещения, односимвольного замещения и подстановки |

Графическая версия elvis меняет форму курсора в зависимости от текущего режима.

# Помощь программисту

Изначально vi разрабатывался как редактор для программистов. У него были функции, особенно полезные для традиционных UNIX-программистов, писавших программы на C и документацию в troff («настоящие» программисты пишут «настоящую» документацию в troff). Некоторые модификации этого редактора являются гордыми продолжателями традиций, и в них добавлены дополнительные функции, позволяющие опытным пользователям применять их еще эффективнее[1].

Две функции (среди множества) достойны отдельного рассмотрения:

*Ускорение цикла «редактирование-компиляция»*

elvis, Vim и vile позволяют быстро вызывать make, перехватывать сообщения об ошибках от компилятора и автоматически перемещаться на строки, содержащие эти ошибки. После исправления ошибок можно заново вызывать make. Все это делается, не выходя из редактора.

*Подсветка синтаксиса*

В elvis, Vim и vile есть возможность подсветки и/или изменения цвета различных синтаксических элементов в различных типах файлов.

## Ускорение цикла «редактирование–компиляция»

Часто программирование состоит из циклов «редактирование-компиляция-отладка», то есть вы делаете изменения, компилируете новый код, затем тестируете и отлаживаете его. При изучении нового языка ошибки совершаются довольно часто, а постоянно выходить и запускать редактор (или приостанавливать/возобновлять его) между компиляциями довольно утомительно.

В elvis, Vim и vile есть возможность оставаться в редакторе при компиляции программы. Более того, эти редакторы обрабатывают вывод компилятора и используют его для перехода на строки с ошибками[2]. Правильное применение этой возможности сэкономит ваше время и повысит производительность.

Приведем пример использования elvis. Пусть вы новичок в C++ и начинаете с обязательной первой программы:

| Клавиши | Результат |
|---|---|
| :w hello.C | `#include <iostream>`<br><br>`int main()`<br>`{` |

[1] В отличие от философии What You See Is What You Get (WYSIWYG, «что видишь, то и получишь»), философия UNIX – это You Asked For It, You Got It («о чем просил, то и получил»). (Спасибо Скотту Ли (Scott Lee).)

[2] Так в vi пришла еще одна функция, привычная пользователям Emacs.

<table>
<tr><th>Клавиши</th><th>Результат</th></tr>
<tr><td></td><td>

```
        std::cout << "hello, world!\n ;
        return 0;
}
```

Вы набрали текст программы, забыв поставить закрывающую кавычку, после чего записали программу в файл `hello.C`.</td></tr>
<tr><td><code>:make hello</code></td><td>

```
g++     hello.C   -o hello
hello.C:5: error: missing terminating " character
hello.C: In function 'int main()':
hello.C:6: error: expected primary-expression before 'return'
hello.C:6: error: expected `;' before 'return'
make: *** [hello] Error 1
```

Вводится команда `:make`, чтобы запустить `make`, который, в свою очередь, запускает компилятор C++ (в нашем случае это `g++`). В выводе `g++` описана каждая ошибка.

```
#include <iostream>

int main()
{
        std::cout << "hello, world\n ;
        return 0;
}
~
line 5: missing terminating " character      5,8    Command
```

Вывод от `make` быстро пропадает, и `elvis` заменяет строку состояния первым сообщением об ошибке, помещая курсор на строку, которую нужно изменить.</td></tr>
</table>

Вы можете исправить ошибки, снова сохранить файл, снова запустить `:make` и, наконец, скомпилировать файл, не содержащий ошибок.

Все разновидности `vi` предлагают схожие функции. Они учитывают изменения в файле, корректно перемещая пользователя между строками, в которых содержатся ошибки. За подробностями, касающимися Vim, обратитесь к разделу «Компиляция и поиск ошибок в Vim» на стр. 314. Другим редакторам посвящены соответствующие главы в части III.

## Подсветка синтаксиса

`elvis`, Vim и `vile` в том или ином виде предоставляют подсветку синтаксиса. В них также есть «раскраска» синтаксиса, при которой меняется цвет различных частей файла на тех дисплеях, которые позволяют это делать (например, под X11 или в консоли Linux). Чтобы узнать больше о подсветке синтаксиса в Vim, обратитесь к разделу «Подсветка синтаксиса» на стр. 305. Про другие редакторы рассказывается в соответствующих главах части III.

# Итоги: сравним редакторы

Большинство модификаций vi поддерживают все функции, о которых рассказывалось в этой главе. В табл. 8.9 подведен итог, показывающий возможности каждого из редакторов. Конечно, эта таблица не представит полной картины; подробностям посвящена оставшаяся часть книги.

*Таблица 8.9. Суммарный список функций*

| Функция | nvi | elvis | vim | vile |
|---|---|---|---|---|
| Многооконное редактирование | • | • | • | • |
| GUI | | • | • | • |
| Расширенные регулярные выражения | • | • | • | • |
| Улучшенные теги | | • | • | • |
| Стеки тегов | • | • | • | • |
| Строки произвольной длины | • | • | • | • |
| 8-разрядные данные | • | • | • | • |
| Бесконечные отмены | • | • | • | • |
| Инкрементный поиск | • | • | • | • |
| Прокрутка влево/вправо | • | • | • | • |
| Индикаторы режима | • | • | • | • |
| Визуальный режим | | • | • | • |
| Ускорение цикла «редактирование-компиляция» | | • | • | • |
| Подсветка синтаксиса | | • | • | • |
| Поддержка в нескольких ОС | | • | • | • |

# Ничто не сравнится с оригиналом

На протяжении многих лет исходный код первоначального vi оставался недоступным, если у пользователя не было лицензии на исходные тексты UNIX. Хотя образовательные учреждения могли достать эти лицензии по сравнительно низкой цене, коммерческие варианты были довольно дорогими. Этот факт дал толчок к созданию модификаций vi, про которые рассказывается в этой книге.

В январе 2002 года исходный код для V7 и 32V UNIX стал доступным по открытой[1] лицензии. После этого открылся доступ почти ко всему коду, разработанному для BSD UNIX, включая ex и vi.

В современных системах, таких как GNU/Linux, оригинальный код не компилируется «прямо из коробки», и его переносимость обеспечить трудно[2]. К счастью, эта работа уже проделана. Если вам хочется исполь-

[1] За подробностями зайдите на веб-сайт Общества Истории UNIX (UNIX Historical Society) *http://www.tuhs.org*.

[2] Мы знаем. Сами пробовали.

зовать оригинальный, «настоящий» vi, то можно скачать исходный код и собрать его самостоятельно. За более подробной информацией обратитесь на сайт *http://ex-vi.sourceforge.net/*.

# Перспектива

Часть II описывает Vim наиподробнейшим образом. В семи полноценных главах рассказывается о том, что упомянуто в этой главе, а также освещена важная тема написания скриптов в Vim, которые делают редактор мощнее и полезнее.

Три главы в части III посвящены nvi, elvis и vile. Каждая из них повествует о следующем:

1. Кто написал редактор и зачем.
2. Важные аргументы командной строки.
3. Онлайн-справка и другая документация.
4. Инициализация – какие файлы и переменные окружения используются программой и в каком порядке.
5. Многооконное редактирование.
6. Графические интерфейсы, если таковые есть.
7. Расширенные регулярные выражения.
8. Улучшенные возможности редактирования (стеки тегов, бесконечная отмена и т. д.).
9. Помощь для программиста (ускорение цикла «редактирование-компиляция», подсветка синтаксиса).
10. Интересные особенности, уникальные для данной программы.
11. Где взять исходный код и на каких операционных системах работает данный редактор.

    Все дистрибутивы сжаты с помощью gzip, GNU zip. Если у вас его еще нет, его можно взять с *ftp://ftp.gnu.org//gnu/gzip/gzip-1.3.12.tar*[1]. Программа untar.c, доступная с FTP-сайта elvis, очень проста и портируется программой для распаковки сжатых gzip архивов tar на не-UNIX системах.

Поскольку все программы, которым посвящена часть III, находятся в процессе разработки, мы даже не пытаемся охватить все их функции, поскольку эта информация быстро устаревает. Вместо этого мы «пройдемся по верхам», затронув только те функции, которые, скорее всего, вам понадобятся и у которых меньше всего шансов измениться при развитии программы. Если вы хотите узнать об использовании самых последних функций какой-либо программы, то чтение этой книги лучше сопровождать изучением онлайн-документации по каждой программе.

[1] На момент написания. Вы можете найти и более свежую версию.

# II

# Vim

В части II описывается наиболее популярная модификация `vi` под названием Vim (от «`vi` improved» – улучшенный `vi`). Эта часть содержит следующие главы:

- Глава 9 «Vim (vi Improved): введение»
- Глава 10 «Главные улучшения Vim по сравнению с vi»
- Глава 11 «Многооконность в Vim»
- Глава 12 «Скрипты Vim»
- Глава 13 «Графический Vim (gvim)»
- Глава 14 «Улучшения Vim для программистов»
- Глава 15 «Другие полезности в Vim»

# 9

# Vim (vi Improved): введение

В этой части книги описывается Vim – одна из разновидностей vi. Мы кратко познакомим вас с этим редактором и его самыми заметными техническими усовершенствованиями по сравнению с vi, а также немного с его историей. В конце мы рассмотрим специальные режимы Vim и средства обучения для новых пользователей. В последующих главах обсуждаются:

- Усовершенствования в редактировании по сравнению с vi
- Многооконное редактирование
- Скрипты Vim
- Графический пользовательский интерфейс (GUI) Vim
- Усовершенствования для программистов
- Редактирование шаблонов
- Другие полезные возможности

Vim означает «vi improved – улучшенный vi». Он был написан и поддерживается Брамом Моленаром (Bram Moolenaar). На сегодняшний день Vim, вероятно, является самой популярной разновидностью vi. Ему даже посвящен специальный интернет-домен (*vim.org*). Текущая версия редактора имеет номер 7.1[1].

Будучи не связанным стандартами и комитетами, Vim продолжает развивать свою функциональность. Вокруг него выросло целое сообщество, которое при помощи номинаций и голосования во время циклов разработки коллективно решает, какие новые функции нужно добавлять, а какие из существующих – изменить.

---

[1] На момент подготовки русскоязычного издания книги – 7.3. – *Прим. науч. ред.*

Благодаря самоотверженной энергии Брама Моленара и системе голосования Vim находится на существенном подъеме. Он держит планку, развиваясь и изменяясь вместе с компьютерным миром и, следовательно, с потребностями пользователей в области редактирования. Например, его контекстно-ориентированное редактирование исходных текстов программ началось с языка C, но охватило и C++, и Java, а теперь и C#.

Vim содержит множество новых функций, облегчающих редактирование кода на многих новых языках. Фактически большинство функций, которые были обещаны в прошлом издании этой книги, уже полностью реализованы. За последние 10 лет компьютерный ландшафт сильно изменился, но Vim шаг за шагом следует за ним.

На сегодняшний день этот редактор настолько популярен, особенно среди UNIX и его разновидностей (например, BSD и GNU/Linux), что для многих пользователей он стал синонимом `vi`. Действительно, во многих дистрибутивах GNU/Linux исполняемый файл Vim по умолчанию устанавливается в `/bin/vi`!

Vim предоставляет отсутствующие в `vi`, но необходимые при работе с современным редактором возможности, например простоту в использовании, поддержку графических терминалов, цвета, подсветку синтаксиса и форматирование, а также расширенную настройку.

# Обзор

## Автор и история[1]

Брам (Bram) начал работать над Vim после того, как купил компьютер Amiga. Будучи пользователем UNIX, он применял `vi`-подобный редактор под названием `stevie`, который показался ему далеким от совершенства. К счастью, у того был открытый исходный код, и Брам начал исправлять ошибки и делать редактор более совместимым с `vi`. Через некоторое время программа стала вполне пригодной к использованию, и Vim версии 1.14 был опубликован на Fred Fish disk 591 (коллекция свободного ПО для Amiga).

Программой стали пользоваться другие, она им понравилась, и люди начали помогать в разработке. За переносом в UNIX последовал перенос на MS-DOS и другие системы, вследствие чего Vim стал одной из наиболее распространенных модификаций `vi`. Многие функции добавлялись постепенно: многоуровневая отмена, многооконность и т. д. Некоторые особенности уникальны для Vim, но большинство было позаимствовано из других разновидностей `vi`. Целью всегда оставалось предоставить пользователю лучшие возможности.

[1] Этот раздел был написан на основании материалов, которые прислал нам Брам Моленар (Bram Moolenaar), автор Vim. Мы выражаем ему благодарность.

На сегодняшний день Vim является самым полнофункциональным из всех редакторов в стиле vi, а его онлайн-справка является исчерпывающей.

Одной из самых необычных функций Vim является поддержка набора текста справа налево, что полезно при использовании таких языков, как иврит и фарси, и иллюстрирует универсальность Vim. Другая его цель – быть надежным как скала, чтобы на него могли положиться разработчики ПО. Сбои в работе редактора очень редки, а если и случаются, то можно восстановить изменения.

Разработка Vim продолжается. Группа людей, добавляющих в него новые возможности и портирующая его на другие платформы, растет. Растет и качество портов на различные операционные системы. Версия для Microsoft Windows предлагает диалоговые окна, включая стандартное окно для открытия файла, что делает труднозапоминаемые команды vi доступными для более широкого круга пользователей.

## Почему Vim?

Vim расширил традиционную функциональность vi настолько сильно, что легче спросить «А почему *не* Vim?». vi ввел стандарт, который позаимствовали другие редакторы (vile, elvis, nvi), а Vim подхватил эстафетную палочку и побежал дальше. В нем разработчики осмелились кардинально изменить функции, порой заставляя процессоры работать на пике своих возможностей, чтобы выполнить задачи Vim за приемлемое время. Неизвестно, предполагал ли Брам (Bram), что скорости процессоров и памяти вырастут настолько, чтобы успевать за этим редактором, но, к счастью, современные компьютеры хорошо справляются даже с самыми трудными его задачами.

## Сравнение и отличие от vi

Vim более широко доступен по сравнению с vi. Существует как минимум несколько версий Vim, доступных практически на всех операционных системах, в то время как vi работает только в UNIX и UNIX-подобных системах.

vi – это оригинал, слабо меняющийся с течением времени. Он скрупулезно следует стандартам POSIX и хорошо выполняет свою роль. Там, где кончается vi, начинается Vim, который предоставляет те же возможности и расширяет их, добавляя графический интерфейс и такие функции, как комплексные опции и поддержка скриптов, выходя далеко за пределы изначальных способностей vi.

Vim поставляется с собственной встроенной документацией в виде каталога специальных текстовых файлов. Простой анализ этого каталога (с использованием стандартного средства подсчета слов UNIX, wc -c *.txt) покажет 129 файлов, в которых содержатся почти 122000 строк документации! Это первое указание на широту возможностей Vim. Он имеет

доступ к этим файлам посредством внутренней команды «help» – еще одной функции, отсутствующей в vi. Чуть позже мы рассмотрим систему справки Vim, а также дадим советы и подсказки о том, как изучать его с максимальной пользой.

Один из способов противопоставить функции Vim возможностям vi – пристально взглянуть на его каталог файлов справки. Многие опции, команды и функции Vim в этих файлах имеют аннотацию «not in vi» или «not available in vi». Простое сканирование файлов справки (командой `grep -i 'not *in vi'`) выводит около 700 совпадений. Даже если уменьшить это количество вдвое, то все равно понятно, что в Vim есть множество функций, которых нет в vi.

Следующие главы охватывают наиболее интересные функции Vim. Мы расскажем о самых лучших и самых популярных способах повышения производительности, от расширений его ранних версий до новых возможностей, а также рассмотрим общепризнанные полезные улучшения, например подсветку синтаксиса, после чего взглянем на менее очевидные функции, эффективные для достижения еще большей производительности. Например, мы расскажем, как настроить Vim таким образом, чтобы его строка состояния показывала изменение даты и времени в реальном времени при каждом перемещении курсора.

## Категории функций

Функции Vim охватывают диапазон действий, отвечающих любой мыслимой задаче редактирования текста. Одни функции являются простым расширением требований пользователей оригинального vi, другие же совершенно новые, и в vi их нет. А если вам нужно нечто, чего нет в Vim, у вас в распоряжении есть встроенные скрипты, возможности которых по расширению функциональности Vim просто безграничны. Вот некоторые категории его функций:

*Синтаксические расширения*

Vim помогает управлять отступами и выделением текста цветом на основе синтаксиса и предоставляет множество опций для этого. Например, если вам не понравилась цветовая раскраска, то ее можно поменять, а если требуются отступы в определенном стиле, Vim его предоставит. Если необходим особенный стиль, то также можно настроить свое окружение.

*Помощь программисту*

Хотя Vim не пытается обеспечить все потребности программиста, у него есть множество функций, обычно присутствующих в интегрированных средах разработки (IDE). В нем есть специализированные функции, позволяющие править код быстрее, начиная от ускорения цикла «редактирование-компиляция-отладка» и заканчивая автозавершением ключевых слов. Как видите, Vim не только ускоряет процесс редактирования, он помогает программировать.

*Графический пользовательский интерфейс (GUI)*

Vim, подобно многим современным удобным в использовании редакторам, позволяет править текст в стиле «указал и щелкнул», что делает его доступным для более широкого круга пользователей. Всем возможностям, предназначенным для опытных пользователей, придается новый импульс, потому что GUI значительно облегчает их применение в задачах редактирования.

*Скрипты и плагины*

Vim позволяет написать для него пользовательское расширение или скачать плагин из Интернета. Таким образом, вы можете сами сделать вклад в развитие редактора, опубликовав свои расширения для всеобщего использования.

*Инициализация*

Для определения сеансов при старте Vim, как и `vi`, использует конфигурационные файлы, но в Vim репертуар определяемых поведений намного шире. Можно свести его к простой установке нескольких опций, как в `vi`, либо написать целый комплект настроек, которые определят сеанс работы в зависимости от заданного контекста. Например, можно написать сценарий, по которому файлы инициализации делали бы предкомпиляцию кода в зависимости от каталога, в котором вы редактируете файлы, или запросить информацию из какого-нибудь источника в реальном времени и включить ее в ваш текст при загрузке.

*Контекст сеанса*

Vim хранит информацию о сеансе в файле `.viminfo`. Вы когда-нибудь спрашивали себя: «на чем я остановился?» при повторном открытии уже редактировавшегося файла? В Vim пользователь вправе сам задавать количество и вид информации, сохраняющейся от сеанса к сеансу. Например, можно определять, сколько нужно запоминать «недавних документов», то есть последних редактировавшихся файлов, сколько исправлений (удалений, изменений) помнить для данного файла, сколько команд хранить в истории команд и сколько буферов и строк хранить из предыдущих правок («вставок», «удалений» и т. д.). Vim не только запоминает правки вашего последнего сеанса работы с файлом, но и помнит основные действия *между* разными файлами. Это полезно при таких правках, как извлечение последовательности строк из одного файла (с помощью `y` [yank] или `d` [delete]) и их «вставка» в другой файл. Все, что попадало в безымянный буфер, запоминается и будет доступно как из одного, так и из другого файла. Кроме того, этот редактор запоминает последний шаблон поиска, так что для поиска по последнему шаблону можно просто использовать команду `n` (найти следующее вхождение) сразу после начала сеанса работы.

Также Vim запоминает, на какой строке находился пользователь в каждом из недавно редактировавшихся файлов. Если при выходе

из сеанса редактирования курсор стоял на строке 25, то в следующий раз при открытии этого файла он попадет на эту строку.

*Постобработка*

Вдобавок к выполнению функций перед сеансом работы, Vim позволяет определить, что нужно сделать *после* редактирования файла. Например, можно написать процедуры очистки, удаляющие временные файлы, созданные во время компиляций, или проделать исправления в файле в реальном времени перед тем, как записать его обратно на диск. У вас есть полный контроль над действиями после редактирования.

*Переходы*

Vim поддерживает переходы между состояниями. При перемещении от буфера к буферу или от окна к окну (обычно это одно и то же) во время сеанса редактирования Vim автоматически делает пред- и постобработку.

*Прозрачное редактирование*

Редактор распознает и автоматически распаковывает заархивированные или сжатые файлы. Например, можно напрямую редактировать gzip-файл, такой как `myfile.tar.gz`, и даже каталоги. Vim позволит перейти в каталог и выбрать файлы для правки, используя знакомые вам команды перемещения.

*Метаинформация*

Vim предлагает четыре удобных регистра, из которых пользователь может извлекать метаинформацию для «вставок»: имя текущего файла (`%`), имя альтернативного файла (`#`), последняя команда, выполненная в командной строке (`:`), и последний вставленный текст (`.`, точка).

*Регистр «черная дыра»*

Это малоизвестное, но полезное расширение регистров редактирования. Как правило, удаление текста помещает его в буферы по схеме ротации, полезной при циклическом проходе по предыдущим удалениям для извлечения ранее удаленного текста. В Vim есть регистр «черная дыра», куда можно выбрасывать текст без влияния на ротацию удаленного текста в обычных регистрах. Если вы – пользователь UNIX, то можете рассматривать этот регистр как версию `/dev/null` для Vim.

*Автозавершение ключевых слов*

Vim позволяет завершать частично набранные слова при помощи правил автозавершения, зависящих от контекста. Например, он может искать слово либо в словаре, либо в файле, содержащем ключевые слова для данного языка.

Также в Vim можно перейти обратно в режим совместимости с `vi` с помощью опции `compatible` (`:set compatible`). Возможно, большую часть

времени вам будут нужны дополнительные функции Vim, однако разумно обеспечить обратную совместимость на случай, когда это может понадобиться.

## Философия

Философия Vim близка к vi. Оба делают редактирование мощным и элегантным, опираются на модальность (командный режим и режим вставки), переносят редактирование на клавиатуру, то есть пользователи могу проделать всю работу быстро и эффективно, не притрагиваясь к мыши (и не нажимая всякие ^X^C). Этот подход можно расценить как «слепое редактирование», аналогично «слепой печати», отразив получаемое повышение скорости и эффективности в соответствующих задачах.

Vim дополняет эту концепцию, разрешая и предоставляя функции для менее опытных пользователей (GUI, визуальный режим) и продвинутые опции для более опытных (скрипты, расширенные регулярные выражения, настраиваемые синтаксис и отступы).

А для суперопытных пользователей, любящих программировать, Vim предоставляет свой исходный код. Пользователям разрешается (и даже поощряется) «усовершенствовать усовершенствования». С философской точки зрения Vim предлагает компромисс, удовлетворяющий потребности *всех* пользователей.

# Где взять Vim

Если ваше окружение – один из вариантов UNIX, в том числе и Mac OS X, то вам повезло, и Vim у вас, скорее всего, уже установлен. Если он доступен в предопределенной переменной окружения PATH, то для открытия окна редактора введите vim в командной строке оболочки. При возникновении типичного для UNIX сообщения об ошибке:

```
sh: command not found: vim
```

попробуйте ввести vi и посмотрите, не появилось ли приветственное сообщение Vim. Возможно, в вашей установке этот редактор заменяет собой vi.

На многих системах можно встретить старую версию Vim. Этот раздел книги поможет установить самую свежую версию, даже если редактор у вас уже установлен. Находясь в нем, с помощью команды :version вы не только убедитесь в том, что запущен именно Vim, но и узнаете номер его версии. Программа выдаст экран, аналогичный следующему:

```
:version
VIM - Vi IMproved 7.0 (2006 May 7, compiled Aug 30 2006 21:54:03)
Included patches: 1-76
Compiled by corinna@cathi
Huge version without GUI. Features included (+) or not (-):
+arabic +autocmd -balloon_eval -browse ++builtin_terms +byte_offset +cindent
```

```
-clientserver -clipboard +cmdline_compl +cmdline_hist +cmdline_info +comments
+cryptv +cscope +cursorshape
 ...
 +profile -python +quickfix +reltime +rightleft -ruby +scrollbind +signs
+smartindent -sniff +statusline -sun_workshop +syntax +tag_binary +tag_old_static
-tag_any_white -tcl +terminfo +termresponse +textobjects +title -toolbar
+user_commands +vertsplit +virtualedit +visual +visualextra +viminfo +vreplace
+wildignore +wildmenu +windows +writebackup -X11 -xfontset -xim -xsmp
-xterm_clipboard -xterm_save
   system vimrc file: "$VIM/vimrc"
     user vimrc file: "$HOME/.vimrc"
      user exrc file: "$HOME/.exrc"
  fall-back for $VIM: "/usr/share/vim"
Compilation: gcc -c -I. -Iproto -DHAVE_CONFIG_H -g -O2
 Linking: gcc -L/usr/local/lib -o vim.exe -lncurses -liconv -lint
```

Некоторые пункты этого вывода будут обсуждаться в главе 10, когда мы научимся компилировать Vim со своими опциями.

Интересно, что на Mac Mini с OS X версии 10.4.10 одного из авторов этой книги не только команда vi запускает Vim, но и документация (man-страница) соответствует Vim!

Если вы все еще не нашли Vim, поищите его в некоторых типовых каталогах, прежде чем скачивать и устанавливать. Если найдете исполняемый файл, добавьте этот каталог как часть значения переменной PATH, после чего вы сможете продолжить:

/usr/bin (этот каталог должен содержаться в PATH)
/bin (и этот тоже)
/opt/local/bin
/usr/local/bin

Если ни один из способов не сработал, то, возможно, Vim у вас не установлен. К счастью, он доступен во многих видах для самых разных платформ и (обычно относительно) прост в скачивании и установке. Ниже мы изложим, как достать Vim для вашей платформы, а затем обсудим, как установить Vim на следующие платформы:

- UNIX и его вариации, включая GNU/Linux
- Windows XP, 2000, Vista
- Macintosh

# Как установить Vim в UNIX и GNU/Linux

Многие современные разновидности UNIX уже поставляются с Vim какой-нибудь версии. Во многих дистрибутивах GNU/Linux vi по умолчанию расположен в каталоге /bin/vi; это просто ссылка на Vim. Большинству пользователей UNIX устанавливать Vim не нужно.

Поскольку существует множество вариантов UNIX, а также множество разновидностей некоторых вариантов (например, Sun Solaris, HP-UX, *BSD, различные дистрибутивы GNU/Linux), самый простой и рекомендуемый способ получить Vim – скачать его исходный код, скомпилировать и установить его[1].

Описанная ниже процедура установки требует среду разработки, способную скомпилировать исходный код. Хотя многие варианты UNIX предоставляют компиляторы и другие инструменты, некоторые (особенно текущий релиз дистрибутива Ubuntu GNU/Linux) потребуют скачивания и установки дополнительных пакетов перед тем, как получить удовольствие от компиляции.

На домашней странице Vim есть ссылка и краткая инструкция на новую рекомендованную процедуру установки, которая называется `aap`. Поскольку `aap` – это новинка, а старый метод установки путем скачивания и компиляции все еще хорошо работает, мы не будем рекомендовать `aap` как предпочтительную процедуру установки. К тому времени, как вы станете читать эти строки, использование `aap` может уже стать устоявшимся.

Также на сайте можно найти готовые пакеты Vim для простой установки стандартным способом на GNU/Linux (RPM для Red Hat, DEB для Debian), IRIX (SoftwareManager), Sun Solaris (Companion Software) и HP-UX. На домашней странице Vim есть ссылки для всех этих систем.

Исходный код Vim можно взять с домашней страницы Vim *http://www.vim.org*. Он упакован в архив tar, сжатый либо GZip (`.gz`), либо BZip2 (`.bz2`). Сейчас практически все операционные системы распознают и поддерживают файлы GZip, а многие варианты UNIX также содержат утилиты, работающие с BZip2. Скачайте исходный код и распакуйте сжатый файл, как показано ниже. Если у вас другая версия, то следует заменить имя скачанного файла:

*файл* `.gz`

```
$ gunzip vim-7.1.tar.gz
```

*файл* `.bz2`

```
$ bunzip2 vim-7.1.tar.gz
```

После завершения работы команды останется файл `vim-7.1.tar` (или похожий, в зависимости от номера скачанной версии). Теперь распакуйте файл `tar`:

```
$ tar xvf vim-7.1.tar
vim71
vim71/README.txt
```

[1] Если вы пользуетесь GNU/Linux, то Vim наверняка отыщется в репозиториях дистрибутива. В этом случае установка из репозитория будет предпочтительнее сборки из исходных текстов. – *Прим. науч. ред.*

```
vim71/runtime
vim71/README_unix.txt
vim71/README_lang.txt
vim71/src
vim71/Makefile
vim71/Filelist
vim71/README_src.txt
 ...
vim71/runtime/doc/vimtutor-ru.1
vim71/runtime/doc/xxd-ru.1
vim71/runtime/doc/evim-ru.UTF-8.1
vim71/runtime/doc/vim-ru.UTF-8.1
vim71/runtime/doc/vimdiff-ru.UTF-8.1
vim71/runtime/doc/vimtutor-ru.UTF-8.1
vim71/runtime/doc/xxd-ru.UTF-8.1
```

Теперь файл vim-7.1.tar можно удалить[1]. Перейдите в каталог Vim, созданный командой tar:

```
$ cd vim71
```

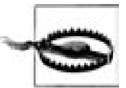
Файл configure – это скрипт, настраивающий параметры установки. На него выпадает большая часть работы по конфигурации. Он проверяет окружение хоста и включает или выключает опции в зависимости от программного обеспечения, установленного в системе.

На этом этапе вы можете выбрать, оставить параметры по умолчанию или включить (выключить) определенные функции. Например, можно скомпилировать Vim с включенным интерфейсом Perl (по умолчанию скрипт configure этого не делает) с учетом, что в будущем вы установите полезные Perl-скрипты для Vim:

```
$ ./configure --enable-perlinterp
```

Если в интерфейсе Perl нет надобности, то выключите этот параметр опцией configure:

```
$ ./configure --disable-perlinterp
```



Текущие версии Vim предлагают слегка различные способы настройки установки. Вместо того чтобы прописывать все --disable-XXX и --enable-XXX в опциях configure, файл INSTALL предлагает делать изменения напрямую в файле feature.h. Если у вас нет веских причин править этот файл, мы рекомендуем компилировать Vim с доступными опциями (они описаны в README.txt) и подгонять редактор под себя посредством правки конфигурационных файлов Vim.

Типичный вывод configure (по умолчанию без опций) выглядит примерно так:

---

[1] Если в вашей системе используется достаточно современная версия GNU tar, то команда tar xvf распознает и распакует сжатый архив; удалять «промежуточный» файл вручную не потребуется. – *Прим. науч. ред.*

```
$ configure
/home/ehannah/Desktop/vim/vim71/src
configure: loading cache auto/config.cache
checking whether make sets $(MAKE)... (cached) yes
checking for gcc... (cached) gcc
checking for C compiler default output file name... a.out
checking whether the C compiler works... yes
checking whether we are cross compiling... no
checking for suffix of executables...
...
checking for NLS... no "po/Makefile" - disabled
checking for dlfcn.h... (cached) yes
checking for dlopen()... no
checking for dlopen() in -ldl... yes
checking for dlsym()... yes
checking for setjmp.h... (cached) yes
checking for GCC 3 or later... yes
configure: creating auto/config.status
config.status: creating auto/config.mk
config.status: creating auto/config.h
config.status: auto/config.h is unchanged
```

Далее соберите Vim с помощью утилиты `make`:

```
$ make
Starting make in the src directory.
If there are problems, cd to the src directory and run make there
cd src && /usr/local/lib/cw/make first
/home/ehannah/Desktop/vim/vim71/src
make[1]: Entering directory `/home/ehannah/Desktop/vim/vim71/src'
gcc -c -I. -Iproto -DHAVE_CONFIG_H     -g -O2        -o objects/
    charset.o charset.c
gcc -c -I. -Iproto -DHAVE_CONFIG_H     -g -O2        -o objects/
    diff.o diff.c
gcc -c -I. -Iproto -DHAVE_CONFIG_H     -g -O2        -o objects/
    digraph.o digraph.c
gcc -c -I. -Iproto -DHAVE_CONFIG_H     -g -O2        -o objects/
    edit.o edit.c

...

make[2]: Entering directory `/home/ehannah/Desktop/vim/vim71/src'
creating auto/pathdef.c
gcc -c -I. -Iproto -DHAVE_CONFIG_H     -g -O2        -o objects/
    pathdef.o auto/
    pathdef.c
make[2]: Leaving directory `/home/ehannah/Desktop/vim/vim71/src'
link.sh: Using auto/link.sed file to remove a few libraries
  gcc     -o vim objects/buffer.o objects/charset.o objects/diff.o
  objects/digraph.o objects/edit.o objects/eval.o objects/ex_cmds.o
  objects/ex_cmds2.o objects/ex_docmd.o objects/ex_eval.o
  objects/ex_getln.o objects/fileio.o objects/fold.o objects/getchar.o
```

```
  objects/hardcopy.o objects/hashtab.o objects/if_cscope.o
  objects/if_xcmdsrv.o objects/main.o objects/mark.o objects/memfile.o
  objects/memline.o objects/menu.o objects/message.o objects/misc1.o
  objects/misc2.o objects/move.o objects/mbyte.o objects/normal.o
  objects/ops.o objects/option.o objects/os_unix.o objects/pathdef.o
  objects/popupmnu.o objects/quickfix.o objects/regexp.o objects/screen.o
  objects/search.o objects/spell.o objects/syntax.o objects/tag.o
  objects/term.o objects/ui.o objects/undo.o objects/window.o
  objects/netbeans.o objects/version.o        -lncurses -lgpm -ldl
link.sh: Linked fine with a few libraries removed
cd xxd; CC="gcc" CFLAGS=" -g -O2" \
    /usr/local/lib/cw/make -f Makefile
/home/ehannah/Desktop/vim/vim71/src/xxd
make[2]: Entering directory `/home/ehannah/Desktop/vim/vim71/src/xxd'
gcc -g -O2   -DUNIX -o xxd xxd.c
make[2]: Leaving directory `/home/ehannah/Desktop/vim/vim71/src/xxd'
make[1]: Leaving directory `/home/ehannah/Desktop/vim/vim71/src'
```

Если все прошло хорошо, то в каталоге `src` появится исполняемый двоичный файл Vim. Редактор готов к использованию, но вызвать его можно либо прописав в командной строке полный путь, либо добавив каталог с Vim в пользовательский путь поиска исполняемых файлов. Если вы не можете устанавливать программы как администратор, то это все, что вам остается.

Чтобы завершить установку Vim как общего ресурса для всех пользователей машины, необходимо иметь права администратора (`root`). Если они у вас есть, зайдите как `root` и введите:

```
# make install
Starting make in the src directory.
If there are problems, cd to the src directory and run make there
cd src && make install
/home/ehannah/Desktop/vim/vim71/src
make[1]: Entering directory `/home/ehannah/Desktop/vim/vim71/src'
if test -f /usr/local/bin/vim; then \
  mv -f /usr/local/bin/vim /usr/local/bin/vim.rm; \
  rm -f /usr/local/bin/vim.rm; \
fi
cp vim /usr/local/bin
strip /usr/local/bin/vim
chmod 755 /usr/local/bin/vim
cp vimtutor /usr/local/bin/vimtutor
chmod 755 /usr/local/bin/vimtutor
/bin/sh ./installman.sh install /usr/local/man/man1 "" /usr/local/
    share/vim /usr/local/share/vim/vim71 /usr/local/share/vim ../
    runtime/doc 644 vim vimdiff evim
installing /usr/local/man/man1/vim.1
installing /usr/local/man/man1/vimtutor.1
installing /usr/local/man/man1/vimdiff.1

...
```

```
if test -d /usr/local/share/icons/hicolor/48x48/apps -a -w /usr/
    local/share/icons/hicolor/48x48/apps \
    -a ! -f /usr/local/share/icons/hicolor/48x48/apps/gvim.png; then \
   cp ../runtime/vim48x48.png /usr/local/share/icons/hicolor/48x48/
      apps/gvim.png; \
fi
if test -d /usr/local/share/icons/locolor/32x32/apps -a -w /usr/
    local/share/icons/locolor/32x32/apps \
    -a ! -f /usr/local/share/icons/locolor/32x32/apps/gvim.png; then \
   cp ../runtime/vim32x32.png /usr/local/share/icons/locolor/32x32/
      apps/gvim.png; \
fi
if test -d /usr/local/share/icons/locolor/16x16/apps -a -w /usr/
    local/share/icons/locolor/16x16/apps \
    -a ! -f /usr/local/share/icons/locolor/16x16/apps/gvim.png; then \
   cp ../runtime/vim16x16.png /usr/local/share/icons/locolor/16x16/
      apps/gvim.png; \
fi
make[1]: Leaving directory `/home/ehannah/Desktop/vim/vim71/src'
```

Установка завершена; если переменные `PATH` у пользователей заданы правильно, они должны получить доступ к Vim.

# Установка Vim в окружении Windows

В системе Microsoft Windows есть две возможности. Первая – это установочный файл `gvim.exe`, который можно взять с домашней страницы Vim. Скачайте и запустите его, а остальное он сделает сам. Мы устанавливали Vim с помощью этого exe-файла на разные машины с Windows, и он прекрасно выполнял свою работу. Этот файл производит установку на Windows XP, 2000, NT, ME, 98 и 95.

В определенный момент установки появится окно DOS, где вы увидите предупреждение о том, что что-то не может быть проверено. У нас это не вызвало проблем.

Другой способ для пользователей Windows состоит в установке Cygwin (*http://www.cygwin.com/*) – комплекта типовых утилит GNU, портированных на платформу Windows. Это удивительно полная реализация почти всех основных программ, используемых на платформе UNIX. Vim является частью стандартной установки Cygwin, поэтому его можно запускать из окна оболочки Cygwin.

# Установка Vim в окружении Macintosh

Mac OS X поставляется с Vim 6.2, но без какого-либо графического интерфейса. Пользователи могут скачать файлы `.tar.bz2`, чтобы скомпилировать версии 6.4 и 7.3, где есть GUI.

Однако при скачивании исходного кода ответственный за него (maintainer) рекомендует брать программу из Mercurial (система управления исходным кодом), чтобы гарантировать, что исходный код обновлен и содержит самые свежие изменения. Это несложно, но скачивание посредством командной строки может быть незнакомо новичкам.

После завершения закачки файлов процедура установки очень похожа на компиляцию и установку в UNIX, которая описана в разделе «Как установить Vim для UNIX и GNU/Linux» на стр. 178.

### Использование Vim в Cygwin

Текстовая консоль Vim хорошо работает в Cygwin, однако `gvim` для Cygwin предполагает, что работает сервер X Window System, так что если запустить `gvim` без этого сервера, то он изящно запустится как текстовый Vim.

Чтобы `gvim` из Cygwin заработал (полагаем, что вы хотите запустить его на локальном экране), запустите в Cygwin X-сервер, введя следующую команду в оболочке Cygwin:

```
$ X -multiwindow &
```

Опция `-multiwindow` указывает серверу X, что Windows может управлять программами Cygwin. Есть много других способов использовать сервер X из Cygwin, но это выходит за рамки нашей книги, равно как и установка этого сервера для Cygwin. Если у вас он не установлен, обратитесь к домашней странице Cygwin за подробностями. В системном лотке Windows должен появиться значок «X». Это подтверждает, что сервер X действительно запущен.

Одновременная установка Vim из Cygwin и с сайта *www.vim.org* может привести к недоразумениям. Некоторые конфигурационные файлы, упоминаемые в настройках Vim, могут располагаться в разных местах, таким образом приводя к практически одинаковым версиям Vim, работающим с совершенно разными опциями. Например, Vim из Cygwin и Vim из Windows могут иметь различные понятия о том, что является домашним каталогом.

## Другие операционные системы

На домашней странице Vim перечислены другие окружения, в которых Vim как будто бы работает, но сообщается, что вы пользуетесь ими на свой страх и риск. Это версии Vim для:

- QNX – операционная система реального времени (RTOS)
- Agenda

- Sharp Zaurus – наладонник с системой на основе Linux
- HP Jornada – наладонник с системой на основе Linux
- Windows CE – версия Windows для наладонников
- Compaq Tru64 Unix на Alpha
- Open VMS, VMS от Digital с POSIX
- Amiga
- OS/2
- RISC OS – ОС, основанная на ЦПУ с ограниченным набором команд (reduced instruction set CPU, RISC)
- MorphOS – ОС на ядре Quark, основанная на Amiga

## Помощь и упрощения для новичков

Признавая, что как `vi`, так и Vim требуют от пользователя определенного уровня подготовки, Vim предоставляет несколько вариантов своего упрощенного использования:

*Графический Vim* (`gvim`)

При вызове команды `gvim` пользователь получает графическое окно, где предлагается Vim с некоторыми добавочными функциями, действующими по принципу «указал и щелкнул», которые приобрели популярность в современных графических программах. Во многих окружениях `gvim` – это отдельный двоичный файл, который получается путем компилирования Vim с включением всех графических опций. Также его можно вызвать командой `vim -g`.

*«Простой» Vim* (`evim`)

Команда `evim` меняет поведение некоторых стандартных функций `vi` на более простое и интуитивно понятное для людей, не знакомых с этим редактором. Возможно, опытным пользователям данный режим не покажется легким, поскольку они уже привыкли к стандартному поведению `vi`. Также его можно вызвать командой `vim -e`.

`vimtutor`

Vim поставляется вместе с `vimtutor` – отдельной командой, запускающей редактор со специальным файлом справки. Такой вызов Vim дает пользователям еще одну отправную точку для изучения редактора. Для завершения `vimtutor` требуется примерно 30 минут.

## Итог

`vi` все еще является стандартным средством редактирования текста в UNIX. В свое время он был почти революционным благодаря двум режимам и философии «слепого редактирования». Vim начинается там,

где заканчивается `vi`. Это следующий виток в эволюции редактирования и управления текстом:

- Vim расширяет `vi`, основываясь на стандартах качества старого редактора. Хотя есть другие модификации редактора, основанные на оригинальном `vi`, именно Vim стал самым популярным и распространенным.
- Он предоставляет намного больше возможностей по сравнению с `vi`, достаточных для того, чтобы стать новым стандартом.
- Vim подходит *как* для новичков, *так и* для опытных пользователей. Новичкам он предложит различные средства обучения и «облегченные» режимы, а экспертам – мощные расширения `vi` наряду с платформой, на которой они смогут улучшать и настраивать Vim для своих потребностей.
- Этот редактор работает везде. Как мы уже говорили, добровольцы портировали Vim в среды, которые считали нужными и для которых он изначально не предназначался. Нельзя заявлять, что он есть буквально везде, но он близок к этому!
- Vim свободен. Более того, как уже упоминалось в прошлом издании этой книги, Vim является благотворительным ПО (charityware). Работа, проделанная Брамом Моленаром (Bram Moolenaar) по созданию, улучшению и поддержке Vim, – один из самых заметных подвигов в мире свободного ПО. Если вам понравилась *его* работа, Брам (Bram) приглашает узнать больше о его любимом деле – помощи детям Уганды. Больше информации об этом можно почерпнуть на сайте *http://iccf-holland.org/* либо обратиться к справке, введя встроенную справочную команду по теме «uganda» (`:help uganda`).

# 10

# Главные улучшения Vim по сравнению с vi

Vim содержит тысячи улучшений по сравнению с `vi`, от разнообразной расцветки синтаксиса до полнофункциональных скриптов. Если `vi` – это хорошо (а это так), то Vim – это просто замечательно. В данной главе мы рассмотрим, как Vim реализовал возможности, на отсутствие которых в `vi` жаловались многие пользователи. Вот некоторые из них:

- Встроенная справка
- Варианты запуска и инициализации
- Новые команды перемещения
- Расширенные регулярные выражения
- Расширенная отмена
- Сборка исполняемого файла под конкретные задачи

## Встроенная справка

Как уже говорилось в прошлой главе, Vim поставляется с документацией, содержащей более 100000 строк. Почти вся она доступна при обращении к встроенной в редактор функции справки и в самом простом виде вызывается командой `:help` (это интересно, поскольку знакомит пользователей с первым примером многооконного редактирования в Vim).

Несмотря на приятный вид, пользователь сталкивается с проблемой «курица-яйцо», поскольку встроенная справка требует небольших навыков владения техникой навигации: чтобы она была действительно эффективной, пользователь должен уметь перемещаться вперед и назад по тегам. Здесь мы представим обзор навигации по экрану справки.

Команда `:help` выдаст что-то похожее на:

```
*help.txt*      For Vim version 7.0.  Last change: 2006 May 07

                        VIM - main help file
                                                                         k
      Move around:  Use the cursor keys, or "h" to go left,          h   l
                    "j" to go down, "k" to go up, "l" to go right.       j
Close this window:  Use ":q[Enter]".
   Get out of Vim:  Use ":qa![Enter]" (careful, all changes are lost!).

Jump to a subject:  Position the cursor on a tag (e.g. |bars|) and hit CTRL-].
   With the mouse:  ":set mouse=a" to enable the mouse (in xterm or GUI).
                    Double-click the left mouse button on a tag, e.g. |bars|.
        Jump back:  Type CTRL-T or CTRL-O (repeat to go further back).

Get specific help:  It is possible to go directly to whatever you want help
                    on, by giving an argument to the |:help| command.
                    It is possible to further specify the context:
                                                       *help-context*
                         WHAT                  PREPEND    EXAMPLE      ~
                      Normal mode command      (nothing)   :help x
```

К счастью, разработчик Vim предвидел потенциальные проблемы в навигации у новичков, поэтому предусмотрительно прописал основные указания и даже подсказал, как закрыть экран справки. Мы рекомендуем начать с него и настаиваем, чтобы вы уделили побольше времени на изучение справки.

После знакомства с командой `help` попробуйте использовать завершение по Tab в командной строке Vim. Для любой команды в приглашении ввода (`:`) нажатие Tab приведет к завершению команды в зависимости от контекста. Например, следующее:

```
:e /etc/termc[TAB]
```

в любой системе UNIX дополнится до:

```
:e /etc/termcap
```

Команда `:e` требует, чтобы ее аргументом являлся файл, так что завершение команды ищет файлы, соответствующие набранному фрагменту, чтобы завершить ввод.

Однако в `:help` есть свой контекст, охватывающий темы справки. Введенный пользователем фрагмент строки темы выявит соответствующие подстроки во всех доступных темах справки Vim. Мы настоятельно рекомендуем изучить и использовать эту функцию. Она экономит время и открывает новые и интересные возможности, о которых вы, возможно, не знали.

Например, пусть вы хотите узнать, как разбить экран. Начните с:

```
:help split
```

и нажмите клавишу Tab. При таком запросе команда `help` начнет пролистывать: `split()`; `:split`; `:split_f`; `splitview`; `splitfind`; `'splitright'`; `'split-`

below'; g:netrw_browse_split; :dsplit; :vsplit; :isplit; :diffsplit; +vertsplit и т. д. Чтобы получить справку по какой-то из тем, нажмите ENTER при появлении интересующего названия. Вы увидите не только то, что искали (например, :split), но познакомитесь с аспектами, о существовании которых, возможно, и не предполагали, например команда :vsplit (вертикальное разбиение окна).

# Варианты запуска и инициализации

Vim использует разные механизмы для настройки окружения при старте. Он проверяет параметры командной строки и самого себя (как его вызвали, по какому имени). Для разных целей (графическое или текстовое окно) существуют по-разному скомпилированные двоичные файлы. Vim также использует целую серию файлов инициализации, в которых может задаваться и меняться бесчисленное количество комбинаций режимов. Опций слишком много, чтобы охватить их все целиком, поэтому мы коснемся только самых интересных. В следующих разделах мы рассмотрим стартовую последовательность по следующей схеме:

- Параметры командной строки
- Поведение, связанное с именем команды
- Конфигурационные файлы (для системы в целом и для каждого пользователя)
- Переменные окружения

Этот раздел знакомит вас с *некоторыми* способами запуска Vim. Для детального изучения других опций используйте команду справки:

```
:help startup
```

## Параметры командной строки

Параметры командной строки Vim обеспечивают мощность и гибкость программы. Одни опции вызывают дополнительные функции, другие подавляют или отменяют поведение, заданное по умолчанию. Мы обсудим синтаксис командной строки, который использовался бы в типичном окружении UNIX. Однобуквенные опции начинаются с - (один минус), например, как в -b, позволяющей редактировать двоичные файлы. Более длинные опции начинаются с -- (два минуса), как в --noplugin, которая отменяет загрузку плагинов (поведение по умолчанию). Аргумент командной строки, состоящий из двух знаков минуса, говорит Vim, что остальная часть командной строки не содержит опций (это стандартное поведение в UNIX).

После опций командной строки можно указать одно или более имен файлов, которые нужно отредактировать. (При этом есть интересный случай, когда именем файла выступает «-». Это говорит Vim, что ввод поступает из стандартного ввода *stdin*. Об этом мы расскажем позднее, но пока можете самостоятельно посмотреть, что получится в этом случае.)

Ниже дан выборочный список опций команды Vim, которые отсутствовали в `vi` (все опции `vi` можно использовать с Vim):

`-b`

Редактирование в двоичном режиме. Название говорит само за себя. Работа с двоичными файлами требует известных навыков, однако это действенный способ править файлы, не доступные для других утилит. За дополнительной информацией обратитесь к разделу справки Vim о редактировании двоичных файлов.

`-c` *`command`*

*command* будет выполнена как команда `ex`. В `vi` есть такая же опция, но Vim позволяет использовать до десяти опций `-c` в одной команде.

`-C`

Запускает Vim в совместимом (с `vi`) режиме. По понятным причинам в `vi` такой опции нет.

`-cmd` *`command`*

Перед исполнением файлов `vimrc` выполняется команда *command*. Это длинная форма опции `-c`.

`-d`

Старт в режиме поиска различий — `diff`. Vim сравнивает два, три или четыре файла и устанавливает опции, упрощающие просмотр различий этих файлов (scrollbind, foldcolumn и т. д.).

Vim использует команду сравнения, доступную в операционной системе, такую как `diff` в системах UNIX. Версия для Windows предлагает скачиваемый исполняемый файл, с помощью которого Vim может провести сравнение.

`-E`

Запуск в улучшенном режиме `ex`. Этот режим использует, среди прочего, расширенные регулярные выражения.

`-F` *или* `-A`

Режимы фарси и арабский соответственно. Они требуют специальную раскладку клавиатуры и рисуют экран справа налево.

`-g`

Запуск `gvim` (графический Vim).

`-m`

Отключает режим записи. Буферы нельзя будет изменить.

`-o`

Открывает все файлы в отдельных окнах. Опционально можно указать целое число, задав тем самым количество открываемых окон. Файлы, перечисленные в командной строке, заполнят только эти окна (остальные пойдут в буферы). Если указанное число окон будет

больше, чем количество файлов, то Vim откроет пустые окна, чтобы число запрашиваемых окон соблюдалось.

`-O`

То же, что и `-o`, но окна разделены вертикально.

`-y`

Запускает Vim в облегченном режиме. Опции устанавливаются, чтобы поведение программы было более понятно новичку. Хотя «упрощенный режим» может помочь несведущим, опытным пользователям он может показаться раздражающим и сбивающим с толку.

`-Z`

Запускается в ограниченном режиме. В основном при этом отключаются все внешние интерфейсы и предотвращается доступ к системным функциям. Например, пользователь не сможет использовать `!G!sort` для сортировки от текущей строки буфера до конца файла. Также будет недоступен фильтр `sort`.

Далее мы приведем набор взаимосвязанных опций, позволяющих использовать удаленный экземпляр сервера Vim. Команды `remote` прикажут Vim (который может выполняться на той же машине, а может и нет) редактировать файл или вычислять выражение на удаленном сервере, а команды `server` укажут редактору, на какой сервер посылать данные, и позволят объявить себя сервером. `serverlist` просто выводит список доступных серверов:

```
-remote file
-remote-silent file
-remote-wait file
-remote-send file
-servername name
-remote-expr expr
-remote-wait-silent file
-remote-tab
-remote-send keys
-remote-wait-silent file
-serverlist
```

Для получения более полной информации обо всех опциях командной строки, включая все команды `vi`, перейдите в раздел «Синтаксис командной строки» на стр. 421.

## Поведение, связанное с именем команды

Vim поставляется в двух основных разновидностях: графической (использующей X Window System в вариантах UNIX и родной GUI в других операционных системах) и текстовой. Каждая из них может быть запущена со своим подмножеством характеристик. Пользователи UNIX просто используют одну из команд следующего списка, чтобы получить желаемое поведение:

`vim`

Запуск текстового Vim.

`gvim`

Запуск Vim в графическом режиме. Во многих окружениях двоичный файл `gvim` — это отдельный файл, полученный при включении всех графических опций во время компиляции. Команда аналогична `vim -g` (в окружении UNIX `gvim` требует наличия X Window System).

`view`, `gview`

Запуск Vim или `gvim` в режиме «только для чтения». Команда аналогична `vim -R`.

`rvim`

Запуск Vim в ограниченном режиме. Выключены доступ ко всем внешним командам оболочки и возможность приостанавливать сеанс работы при помощи команды `^Z`.

`rgvim`

То же, что и `rvim`, но для графического режима.

`rview`

Похож на `view`, но запускается в ограниченном режиме, в котором пользователь не имеет доступа к фильтрам, внешнему окружению и функциям ОС. Команда аналогична `vim -Z` (опция `-R` включает режим «только для чтения», описанный выше).

`rgview`

То же, что и `rview`, но для графического режима.

`evim`, `eview`

Использует «облегченный» режим для редактирования или просмотра. Vim настраивает опции и функции таким образом, чтобы вести себя более привычно для тех, кто не знаком с его парадигмой. Команда аналогична `vim -y`. Возможно, опытным пользователям этот режим не покажется упрощенным, поскольку они уже привыкли к стандартному поведению `vi`.

Обратите внимание, что команды *gXXX*, аналогичной этим командам, не существует. Видимо, причина в том, что `gvim` считается изначально простым, по крайней мере интуитивно понятным в изучении, с предсказуемым поведением в стиле «наведи-и-щелкни».

`vimdiff`, `gvimdiff`

Запуск в режиме `diff` и сравнение файлов, переданных в качестве аргументов. Этот режим подробно рассмотрен в разделе «В чем разница?» на стр. 330.

`ex`, `gex`

Использование старого режима строкового редактирования `ex`. Полезен при использовании из скриптов. Команда аналогична `vim -e`.

Пользователи Windows могут получить доступ к этим версиям программы Vim в списке программ (в стартовом меню).

## Системные и пользовательские конфигурационные файлы

Vim ищет инструкции по инициализации особым образом. Он выполняет первый найденный набор инструкций (либо в переменной окружения, либо из файла) и начинает редактирование. Иными словами, первый встретившийся элемент из следующего списка будет единственным элементом из списка, который будет выполнен. Список таков:

1. `VIMINIT`. Это переменная окружения. Если она не пустая, Vim исполняет ее содержимое как команду `ex`.
2. Пользовательские файлы `vimrc`. Файл инициализации `vimrc` (Vim resource) кроссплатформенный, однако из-за некоторых отличий между операционными системами и платформами Vim ищет его в разных местах в следующем порядке:

   `$HOME/.vimrc` (Unix, OS/2 и Mac OS X)
   `s:.vimrc` (Amiga)
   `home:.vimrc` (Amiga)
   `$VIM/.vimrc` (OS/2 и Amiga)
   `$HOME/_vimrc` (DOS и Windows)
   `$VIM/_vimrc` (DOS и Windows)

3. Опция `exrc`. Если установлена опция Vim `exrc`, то редактор будет искать три дополнительных файла настроек: `[._]vimrc`; `[._]vimrc` и `[._]exrc`.

Файл `vimrc` – хорошее место для установки характеристик редактора Vim. Теоретически в этом файле можно включать или выключать любую опцию Vim, и он особенно хорошо подходит при установке глобальных переменных и определении функций, аббревиатур, отображений клавиш и т. д. Вот что следует знать о файле `vimrc`:

- Комментарии начинаются с двойной кавычки ("). Она может стоять в любом месте строки. Любой текст после двойной кавычки, включая ее саму, будет игнорироваться.
- Команды `ex` можно указывать с двоеточием или без него. Например, `set autoindent` равносильно `:set autoindent`.
- Файл будет гораздо удобнее в управлении, если вы разобьете большие наборы определений опций по разным строкам. Например:

  ```
  set terse sw=1 ai ic wm=15 sm nows ruler wc=<Tab> more
  ```

  эквивалентно следующему:

  ```
  set terse " short error and info messages
  set shiftwidth=1
  set autoindent
  set ignorecase
  ```

```
set wrapmargin=15
set nowrapscan " don't scan past end or top of file in searches
set ruler
set wildchar=<TAB>
set more
```

Обратите внимание, насколько более удобочитаем второй набор команд. Также его легче обрабатывать посредством удалений, вставок и временного комментирования строк при поиске ошибок в конфигурационном файле. Например, если вам захочется временно убрать нумерацию строк из стартовой настройки, просто поставьте двойную кавычку (") в начало строки `set number` конфигурационного файла.

## Переменные окружения

Множество переменных окружения влияет на стартовое поведение Vim, а некоторые – даже на поведение во время редактирования. В основном они понятны и заменяются установками по умолчанию, если не задать их специально.

### Как установить переменные окружения

Командное окружение (в UNIX называемое *оболочкой*), существующее при регистрации, устанавливает переменные, отражающие его поведение либо влияющие на него. Переменные окружения особенно полезны, поскольку они влияют на программы, вызванные из командного окружения. Следующие инструкции касаются не только Vim; их можно применять для установки переменных окружения, которые вы хотите видеть в своем командном окружении.

*Windows*

Чтобы установить переменную окружения:

1. Вызовите панель управления.
2. Дважды щелкните по System.
3. Перейдите на вкладку Advanced.
4. Нажмите кнопку Environment Variables.

В результате вы увидите окно, разделенное на две области переменных окружения, User и System. Новичкам не следует менять системные переменные окружения. В области User можно задать переменные окружения, связанные с Vim, и сделать их постоянными для всех сеансов работы.

*Unix/Linux Bash и другие оболочки Bourne*

Отредактируйте подходящий файл настройки оболочки (например, для пользователей Bash это `.bashrc`) и вставьте строки следующего вида:

```
VARABC=somevalue
VARXYZ=someothervalue
```

```
MYVIMRC=myfavoritevimrcfile
export VARABC VARXYZ MYVIMRC
```

Порядок строк роли не играет. Оператор `export` просто делает эти переменные видимыми для программ, работающих в оболочке, таким образом превращая их в переменные окружения. Их значения можно устанавливать как до экспорта, так и после него.

*Unix/Linux оболочки C*

Отредактируйте подходящий файл настройки оболочки (такой как `.cshrc`) и вставьте строки следующего вида:

```
setenv VARABC somevalue
setenv VARXYZ someothervalue
setenv MYVIMRC myfavoritevimrcfile
```

## Переменные окружения, связанные с Vim

Далее мы приводим список переменных окружения, чаще всего используемых в Vim, а также их действие.

Опция команды Vim `-u` отменяет действие переменных окружения Vim и переходит непосредственно к указанному конфигурационному файлу. `-u` *не* отменяет переменные окружения, не относящиеся к Vim:

`SHELL`

Указывает, какую оболочку или внешний интерпретатор команд использует Vim в командах оболочки (`!!`, `:!` и т. д.). В MS-DOS, если `SHELL` не установлена, вместо нее используется переменная окружения `COMSPEC`.

`TERM`

Устанавливает в Vim внутреннюю опцию `term`. Это необязательно, поскольку редактор настраивает свой терминал так, как считает нужным. Другими словами, Vim лучше знает, что такое терминал, нежели предопределенная переменная.

`MYVIMRC`

Отменяет поиск файлов инициализации в Vim. Если `MYVIMRC` при старте имеет значение, то редактор полагает, что оно и является именем файла инициализации и, если такой файл существует, берет оттуда все установки. Другие файлы при этом не проверяются (см. последовательность поиска в предыдущем разделе).

`VIMINIT`

Указывает, какие команды `ex` нужно выполнить при старте Vim. Можно задать несколько команд, отделяя их вертикальной чертой (`|`).

`EXINIT`

То же, что и `VIMINIT`.

`VIM`

Содержит путь к системному каталогу, где находится информация о стандартной установке Vim (только для сведения, в Vim не используется).

Если на компьютере установлено несколько версий Vim, то команда `VIM`, скорее всего, будет отражать разные значения в зависимости от запущенной пользователем версии. Например, на машине одного из авторов версия Cygwin установила переменной `VIM` значение `/usr/share/vim`, тогда как пакет с `vim.org` установил ее равной `C:\Program Files\Vim`.

Это важно иметь в виду во время правок в файлах Vim, поскольку при редактировании не тех файлов ваши изменения не дадут эффекта!

`VIMRUNTIME`

Указывает на файлы поддержки Vim, такие как онлайн-документация, определение синтаксиса и каталоги с плагинами. Обычно редактор сам об этом знает. Если пользователь задает эту переменную, например в файле `vimrc`, то это может вызвать ошибки при установке более новой версии Vim, так как пользовательская `VIMRUNTIME` может указывать на старое, несуществующее или неправильное место.

# Новые команды перемещения

В Vim присутствуют все команды перемещения и движения `vi`, большая часть которых перечислена в главе 3, а также есть новые, приведенные в табл. 10.1.

*Таблица 10.1. Команды перемещения Vim*

| Команда | Описание |
|---|---|
| `<C-End>` | Переход в конец файла, то есть на последний символ последней строки файла. Если задать *count*, то произойдет переход на последний символ строки *count*. |
| `<C-Home>` | Переход на первый непробельный символ первой строки файла. Это отличает его от `<C-End>`, поскольку `<C-Home>` не переместит курсор на пробельный символ. |
| *count*`%` | Переход на *count-й* процент файла; помещает курсор на первый печатаемый символ строки. Важно отметить, что Vim подсчитывает процент, основываясь на количестве строк, а не символов в файле. Это может показаться неважным, однако рассмотрим файл, содержащий 200 строк, первые 195 из которых состоят из пяти символов (например, `$4.98`), а последние четыре – из 1000 символов. В UNIX, с учетом символа переноса строки, файл содержит примерно<br><br>(195 * (5 + 1)) (*Число символов в первых 5-символьных строках*) |

| Команда | Описание |
|---|---|
| | + 2 + (4 * (1000 + 1)) (*Число символов в 1000-символьных строках*)<br>или 5200 символов. Реально 50% соответствует позиции на строке 96, тогда как 50%-е перемещение в Vim поместит курсор на сотую строку. |
| :go n<br>n go | Переход на n-й байт в буфере. В расчет берутся все символы, включая знак конца строки. |

## Движение в визуальном режиме

Vim позволяет пользователям визуально определять выделение и производить над ним редактирование. Это похоже на то, что можно увидеть в графических редакторах, когда пользователь выделяет области, щелкнув мышкой в одном месте и перетащив ее в другое. Визуальный режим Vim добавляет удобства в работе, отображая выделенный кусок текста, над которым производятся какие-либо действия, *а также* действие всех тех мощных команд Vim, которые производят изменения в визуально выделенном тексте. Это позволит проделывать намного более изощренную работу над выделенным текстом, нежели традиционные команды «вырезать» и «вставить» в менее продвинутых редакторах.

Выделять текст в Vim можно теми же способами, что и в других редакторах, то есть указывая область мышью. Однако кроме этого существуют полезные команды перемещения и некоторые специальные команды для визуального режима, позволяющие задать визуальное выделение.

Например, можно в обычном режиме ввести v, после чего запустится визуальный режим. При нахождении в нем любая команда перемещения передвигает курсор *и* подсвечивает текст по мере того, как курсор переходит на новую позицию. Так, команда «на следующее слово» (w) в визуальном режиме переместит курсор на следующее слово и подсветит выделенный текст. Другие перемещения также расширят выделенный фрагмент.

В визуальном режиме Vim использует специальные команды, с помощью которых удобно расширять выделенный текст, выбирая текстовый объект, окружающий курсор. Например, курсор может находиться внутри «слова», одновременно внутри «предложения» и, наконец, в «абзаце». С помощью команд, расширяющих подсвеченный текст до текстового объекта, Vim позволит увеличить визуальное выделение. Для визуального выделения слова можно использовать aw (в визуальном режиме).

Vim предлагает различные команды перемещения, использующие преимущества «визуального режима», который подсвечивает строки и символы в буфере, наглядно показывая, какой именно текст будет подвергаться дальнейшим действиям Vim. Визуальные области в буфере можно подсвечивать несколькими способами. В текстовом режиме просто введите v для перехода в визуальный режим и выхода из него. При

включенном визуальном режиме при перемещении курсора в буфере появляется и подсвечивается выделение. В gvim можно просто выделять текст мышью. Это установит визуальный флаг Vim.

В табл. 10.2 показаны некоторые команды Vim для перемещения в визуальном режиме.

*Таблица 10.2. Команды для перемещения в визуальном режиме Vim*

| Команда | Описание |
|---|---|
| *count*aw, *count*aW | Выделяет *count* слов, включая, если присутствует, пробельный символ. Это немного отличается от iw (см. следующий пункт). Строчная w ищет слова, разделенные знаками препинания, а прописная W ищет слова, разделенные пробельными символами. |
| *count*iw, *count*iW | Выбирает *count* слов. Добавляет слова без пробельных символов. Строчная w ищет слова, ограниченные знаками препинания, а прописная W – пробельными символами. |
| as, is | Добавляет предложение или внутреннее[a] предложение. |
| ap, ip | Добавляет абзац или внутренний абзац. |

[a] «Внутренним» является сам текстовый объект без окружающих его непечатаемых и т. п. символов. – *Прим. науч. ред.*

Для получения подробной информации о текстовых объектах и их использовании в визуальном режиме воспользуйтесь командой справки:

```
:help text-objects
```

# Расширенные регулярные выражения

Из всех модификаций vi Vim предоставляет самый богатый набор функций для работы с регулярными выражениями. Большая часть текста описания в нижеприведенном списке взята из документации Vim:

\|

Указывает на варианты (house\|home).

\+

Соответствие одному или более предшествующим регулярным выражениям.

\=

Соответствие одному или ни одному из предшествующих регулярных выражений.

\{*n*,*m*}

Соответствие максимальному количеству предшествующих регулярных выражений в диапазоне от $n$ до $m$. $n$ и $m$ – это числа от 0 до 32000.

Vim требует, чтобы обратная косая черта стояла только перед левой фигурной скобкой, а не перед правой.

`\{n}`

Соответствие *n* предшествующим регулярным выражениям.

`\{n,}`

Соответствие как можно большему количеству предшествующих регулярных выражений, но не меньше *n*.

`\{,m}`

Соответствие как можно большему количеству предшествующих регулярных выражений в диапазоне от 0 до *m*.

`\{}`

Соответствие как можно большему количеству предшествующих регулярных выражений, начиная от нуля (аналогично `*`).

`\{-n,m}`

Соответствие минимальному количеству предшествующих регулярных выражений в диапазоне от *n* до *m*.

`\{-n}`

Соответствие *n* предшествующим регулярным выражениям.

`\{-n,}`

Соответствие наименьшему количеству предшествующих регулярных выражений, но не меньше *n*.

`\{-,m}`

Соответствие наименьшему количеству предшествующих регулярных выражений в диапазоне от 0 до *m*.

`\i`

Соответствие любому символу идентификатора согласно опции `isident`.

`\I`

Подобна `\i`, но исключает цифры.

`\k`

Соответствие любому ключевому слову согласно опции `iskeyword`.

`\K`

Подобна `\k`, но исключает цифры.

`\f`

Соответствие любому символу имени файла согласно опции `isfname`.

`\F`

Подобна `\f`, но исключает цифры.

\p

Соответствие любому печатаемому символу согласно опции isprint.

\P

Подобна \p, но исключает цифры.

\s

Соответствие любому пробельному символу (то есть пробелу или табуляции).

\S

Соответствует всему, что не является пробелом или табуляцией.

\b

Символ забоя (Backspace).

\e

Escape.

\r

Возврат каретки.

\t

Табуляция.

\n

Зарезервирована для будущего использования[1]. В конце концов будет использоваться для многострочных шаблонов. За подробностями обращайтесь к документации по Vim.

~

Соответствие последней использовавшейся строке замены.

\(…\)

Обеспечивает группировку для *, \+ и \=, а также делает доступным подтекст в команде замены (\1, \2 и т. д.).

\1

Соответствует той же строке, которая соответствовала первому подвыражению в \( и \). Например, \([a-z]\).\1 соответствует *ata*, *ehe*, *tot* и т. д. \2, \3 и прочие могут использоваться для соответствия второму, третьему и последующим подвыражениям.

Опции isident, iskeyword, isfname и isprint определяют печатаемые символы, содержащиеся в идентификаторах, ключевых словах и именах файлов. Использование этих опций придает регулярным выражениям еще больше гибкости.

[1] В версии Vim 7.3 этот метасимвол уже работоспособен. – *Прим. науч. ред.*

## Сборка исполняемого файла под конкретные задачи

Установленный по умолчанию Vim удовлетворяет потребностям большинства пользователей. Современные компьютеры ввиду высокой производительности выполняют все расширенные функции редактора. Однако в некоторых случаях окружение или обстоятельства вынуждают использовать его облегченную версию.

Пользователям может понадобиться Vim с ограничениями, например для наладонных устройств под управлением Linux, у которых не так много памяти. Различным пользователям могут оказаться ненужными такие предкомпилированные функции, как проверка орфографии (например, программистам, которым не важны функции, пришедшие из текстовых процессоров) или поддержка `perl` (из-за того, что на их машинах `perl` не установлен).

Гораздо проще жить с доступными функциями, чем перенастраивать, перекомпилировать и заново устанавливать Vim с новыми опциями, только чтобы добавить новые возможности.

# 11

# Многооконность в Vim

По умолчанию Vim открывает все файлы в одном окне, отображая только один буфер при перемещении между несколькими файлами или частями одного файла. Однако Vim может предложить и многооконное редактирование, упрощающее сложные задачи. Это не то же самое, что запуск различных экземпляров Vim в графическом терминале. В данной главе рассказывается об использовании нескольких окон при запуске только одного процесса Vim (назовем это *сеансом*).

Несколько окон можно создать как при инициализации сеанса, так и после его начала. Количество окон, используемых в сеансе работы, ограничивается только вашим здравым смыслом. Их можно удалять, оставив только одно окно редактирования.

Ниже приведены некоторые примеры, когда несколько окон могут облегчить вашу жизнь:

- Редактирование нескольких файлов, которые нужно отформатировать одинаковым образом. Вы сможете зрительно сравнивать их при выполнении работы.
- Многократное и быстрое выполнение действий типа «вырезать» и «вставить» между несколькими файлами или разными частями одного файла.
- Отображение части одного файла в качестве образца, чтобы облегчить работу в других местах файла.
- Сравнение двух версий файла.

Vim предлагает множество функций, облегчающих управление окнами, включая:

- Разделение окна по горизонтали или по вертикали.
- Быстрый переход от одного окна к другому и обратно.

- Копирование и перемещение текста в несколько окон и из нескольких окон.
- Изменение расположения и размеров окон.
- Работа с буферами, включая скрытые (о них мы расскажем ниже).
- Использование внешних утилит, например команды `diff`, с несколькими окнами.

В этой главе мы познакомим вас с работой в нескольких окнах, расскажем, как запустить многооконный сеанс, обсудим имеющиеся возможности и дадим подсказки по сеансу редактирования, а также покажем, как закончить работу, гарантированно сохранив все изменения (или отказавшись от них, если хотите!). Следующие темы охватывают:

- Инициализация или запуск сеанса многооконного редактирования.
- Команды `:ex` для нескольких окон.
- Перемещение курсора от одного окна к другому.
- Перемещение окон по экрану.
- Изменение размера окон.
- Буферы и их взаимодействие с окнами.
- Работа с вкладками (подобно вкладкам в современных веб-браузерах и диалоговых окнах).
- Закрытие окон и выход из них.

# Инициализация многооконного сеанса

Многооконный сеанс можно инициализировать при запуске Vim, а можно разделить существующие окна в процессе редактирования. В Vim многооконность динамична, что позволяет открывать, закрывать окна и перемещаться между ними в любое время и почти при любых обстоятельствах.

## Инициализация многооконности из командной строки (оболочки)

По умолчанию Vim открывает только одно окно для сеанса, даже если вы запускаете его с несколькими файлами. Нельзя с уверенностью сказать, почему Vim не хочет открывать разные окна для различных файлов, но, возможно, причина в том, что использование одного окна соответствует поведению `vi`. Отдельные файлы занимают отдельные буферы, по одному буферу на файл (про буферы мы расскажем чуть ниже).

Чтобы открыть несколько окон из командной строки, воспользуйтесь опцией Vim `-o`. Например:

```
$ vim -o file1 file2
```

***Рис. 11.1.*** *Результат выполнения «vim -o file1 file2»*

При этом будет открыт сеанс редактирования с экраном, поделенным горизонтально на два одинаковых окна для каждого файла (рис. 11.1). Для каждого файла, указанного в командной строке, Vim попытается открыть свое окно. Если он не сможет разбить экран на требуемое число окон, то окна достанутся первым файлам, перечисленным в командной строке, а остальные будут загружены в буферы, скрытые от пользователя (но тем не менее доступные).

Если после опции `-o` поставить число *n*, то команда зарезервирует нужное количество окон:

```
$ vim -o5 file1 file2
```

Будет открыт сеанс с экраном, разделенным по горизонтали на пять окон одинакового размера, в самом верхнем из которых разместится `file1`, в следующем – `file2` (рис. 11.2).

Когда Vim создает несколько окон, по умолчанию он добавляет к каждому из них строку состояния (тогда как в однооконном сеансе по умолчанию она не отображается). Поведением редактора можно управлять с помощью опции Vim `laststatus`, например:

```
:set laststatus=1
```

Задайте `laststatus` равным 2, чтобы видеть строку состояния всегда, даже в однооконном режиме (лучше всего сделать это в файле `.vimrc`).

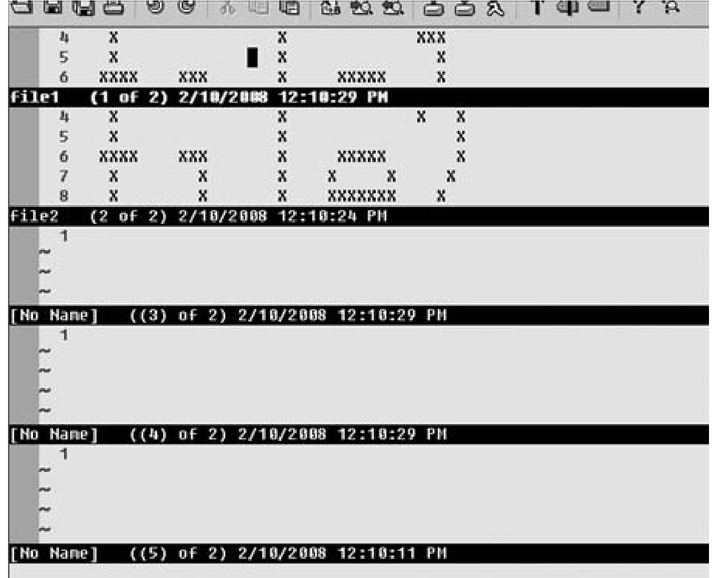



***Рис. 11.2.*** *Результат выполнения «vim -o5 file1 file2»*

Поскольку размер окна влияет на читаемость и удобство работы, вам, возможно, захочется контролировать ограничения, налагаемые Vim на размеры окон. Используйте опции Vim `winheight` и `winwidth` для задания разумных ограничений для текущего окна (размеры остальных окон также можно изменить, чтобы подогнать их под текущее окно).

## Многооконное редактирование в Vim

Можно инициализировать и менять конфигурацию окон непосредственно из Vim. Создайте новое окно командой `:split`. Она разделит пополам текущее окно и покажет один и тот же буфер в обеих половинах. После этого вы сможете быстро перемещаться по одному и тому же файлу в каждом из этих окон.

В этой главе представлены «горячие» сочетания клавиш для многих команд. Так, в данном случае `^Ws` также приведет к разделению окна (все команды Vim, связанные с окнами, начинаются на `^W`, где «W» – сокращение от «window»). В целях этого повествования мы будем приводить только методы с командной строкой, поскольку в этом случае можно изменить поведение команды по умолчанию, добавляя дополнительные опции. Если вы заметите, что постоянно используете одни и те же команды, то сможете легко отыскать соответствующие им сочетания клавиш в документации Vim, как описано в разделе «Встроенная справка» на стр. 187.

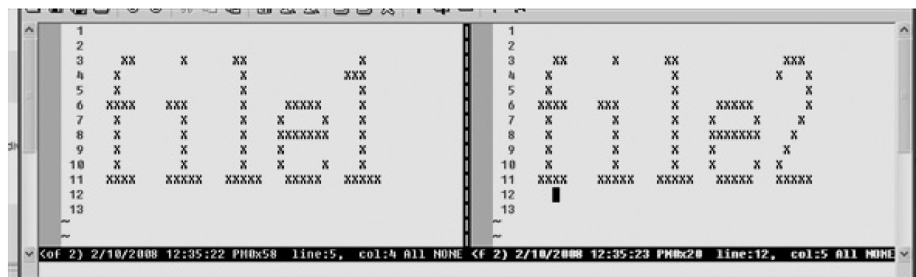


*Рис. 11.3. Вертикально разделенное окно*

Аналогично окно, разделенное по вертикали, можно создать командой `:vsplit` (рис. 11.3).

В каждом из этих способов Vim разделяет окно (вертикально или горизонтально), а поскольку в команде `:split` не было указано никакого файла, вы увидите две копии одного файла в двух окнах.

Не верите, что можно редактировать один и тот же файл одновременно в двух окнах? Разделите окно редактирования и при помощи прокрутки сделайте так, чтобы в обоих окнах видеть одну и ту же часть файла. Выполните изменения. Посмотрите на другое окно. Чудеса.

Как и зачем это использовать? Один из авторов постоянно пользуется следующим приемом: при написании скриптов оболочки или программ на C он создает блок текста, описывающий использование программы (как правило, программа отображает этот текст, если ее вызвать опцией `--help`). Экран разделяется так, чтобы в одном окне была справка по применению программы, после чего она используется как образец при редактировании кода в другом окне, где обрабатываются все опции и аргументы командной строки, описанные в тексте справки. Часто (почти всегда) этот код сложный и заканчивается довольно далеко от справки по применению, то есть в одном окне код и текст отобразить не получилось бы.

Если потребуется редактировать или просмотреть другой файл, не меняя своей позиции в текущем, задайте имя этого файла в качестве аргумента команды `:split`. Например:

```
:split otherfile
```

Следующий раздел детально описывает разделение и слияние окон.

# Открытие окон

В этом разделе мы углубленно изучим, как добиться нужного поведения при разделении окна.

## Новые окна

Как рассказывалось выше, самый простой способ открыть новое окно – выполнить команду :split (для горизонтального разделения) или :vsplit (для вертикального). Ниже приводится подробное описание команд и их вариаций. Также представлена краткая аннотация команд для быстрой справки.

## Опции при разделении

Полная команда :split, открывающая новое горизонтальное окно, имеет вид:

```
:[n]split [++opt] [+cmd] [file]
```

где

*n*
: Указывает Vim, сколько строк нужно отображать в новом окне, которое будет располагаться над всеми остальными.

*opt*
: Передает информацию об опциях Vim в сеанс работы с новым окном (обратите внимание, что *opt* должен предваряться двумя плюсами).

*cmd*
: Передает команду, которую нужно выполнить в новом окне (обратите внимание, что перед *cmd* надо поставить один плюс).

*file*
: Указывает файл, который следует открыть в новом окне.

Предположим, при работе с файлом вы хотите разделить окно, чтобы отредактировать другой файл с именем otherfile. При этом необходимо, чтобы в новом сеансе использовалось значение fileformat, равное unix (формат текстовых файлов UNIX, где конец строки обозначается символом новой строки, а не комбинацией из символа возврата каретки и новой строки). Высота окна должна быть равна 15 строкам. Введите:

```
:15split ++fileformat=unix otherfile
```

Чтобы просто разделить экран, имея в обоих окнах один и тот же файл и используя все настройки по умолчанию, можно применять сочетания клавиш ^Ws, ^WS или ^W^S.

Для разделения экрана на равные части используйте опцию equalalways. Ее предпочтительно прописать в файле .vimrc, чтобы она стала постоянной для всех сеансов работы. По умолчанию при использовании equalalways экран разделится на равные части по горизонтали и вертикали. Для управления тем, какое именно направление разделения сделать одинаковым, воспользуйтесь опцией eadirection (ее значения – hor, ver и both для одинакового разделения по горизонтали, вертикали, а также того и другого соответственно).

Следующая форма команды `:split` также откроет новое горизонтальное окно, но с небольшими нюансами:

```
:[n]new [++opt] [+cmd] [file]
```

Помимо создания нового окна будут выполнены автокоманды `WinLeave`, `WinEnter`, `BufLeave` и `BufEnter` (подробнее об автокомандах можно узнать в разделе «Автокоманды» на стр. 237).

Наряду с командой горизонтального разделения Vim предоставляет аналогичную команду для разделения по вертикали. Так, для создания вертикально разделенного окна вместо `:split` или `:new` следует использовать `:vsplit` и `:vnew` соответственно. В командах разделения по вертикали используются такие же необязательные параметры, как и в командах горизонтального разделения.

Существуют две команды разделения по горизонтали, у которых нет вертикальных «собратьев»:

`:sview` *filename*

: Разделяет экран горизонтально, чтобы создать новое окно, устанавливая для этого буфера режим «только для чтения». Команда `:sview` требует наличия аргумента — имени файла.

`:sfind [++opt] [+cmd]` *filename*

: Работает аналогично `:split`, но ищет *filename* в `path`. Если Vim не обнаруживает файла, то разделения окна не происходит.

## Команды условного разделения

Vim позволяет определять команду, которая создает новое окно, только если найден новый файл. `:topleft` *cmd* предписывает редактору выполнить команду *cmd* и отобразить новое окно с курсором слева вверху, если *cmd* откроет новый файл. Команда может привести к трем различным результатам:

- *cmd* разделяет окно горизонтально, после чего новое окно занимает верхнюю часть окна Vim.
- *cmd* разделяет окно вертикально, и новое окно занимает левую сторону окна Vim.
- *cmd* не приводит к разделению окна, но вместо этого помещает курсор в левый верхний угол текущего окна.

В дополнение к команде условного разделения `:topleft` Vim предлагает ряд других аналогичных команд: `:leftabove`, `:rightbelow`, `:botright` и `:vertical`. Подробную информацию о них вы найдете во встроенной справке Vim, доступной по команде `:help`.

## Сводка команд работы с окнами

Таблица 11.1 резюмирует команды для разделения окон.

*Таблица 11.1. Сводка команд работы с окнами*

| Команда ex | Команда vi | Описание |
|---|---|---|
| :[*n*]split [++*opt*] [+*cmd*] [*file*] | ^Ws<br>^WS<br>W^S | Разделяет текущее окно на две части слева направо (по горизонтали), курсор помещается в новое окно. Необязательный аргумент *file* открывает указанный файл в новом окне. Создаваемые окна по возможности имеют одинаковый размер, что определяется свободным местом на экране. |
| :[*n*]new [++*opt*] [+*cmd*] | ^Wn<br>^W^N | То же, что :split, но в новом окне появляется пустой файл. Обратите внимание, что буфер останется безымянным, пока ему не будет присвоено имя. |
| :[*n*]sview [++*opt*] [+*cmd*] [*file*] | | Версия :split «только-для-чтения». |
| :[*n*]sfind [++*opt*] [+*cmd*] [*file*] | | Разделяет окно и открывает *file* (если таковой указан) в новом окне. Ищет *file* в path. |
| :[*n*]vsplit [++*opt*] [+*cmd*] [*file*] | ^Wv<br>^W^V | Разделяет текущее окно на два сверху вниз (по вертикали) и открывает *file* (если таковой указан) в новом окне. |
| :[*n*]vnew [++*opt*] [+*cmd*] | | «Вертикальная» версия :new. |

# Перемещение по окнам (движение курсора между окнами)

С помощью мыши несложно переходить из одного окна в другое как в gvim, так и в Vim. gvim поддерживает щелчок мышью по умолчанию, тогда как в Vim такое поведение нужно настроить опцией mouse. Хорошим примером настройки по умолчанию является :set mouse=a, при этом мышь будет использоваться везде: и в командной строке, и для ввода, и для перемещения.

Если у вас нет мыши либо вы предпочитаете управлять сеансом с клавиатуры, Vim предоставляет полный набор команд для быстрого и точного перехода между окнами. К счастью, для этого он использует все то же сочетание клавиш ^W. Следующая после этого последовательность кнопок указывает на перемещение или на другое действие. Это должно быть знакомо опытным пользователям vi и Vim, поскольку данные команды практически повторяют команды перемещения при редактировании.

Мы не будем описывать каждую команду и ее работу, а лучше рассмотрим пример, после которого таблица обзора команд станет очевидной.

Чтобы перейти из текущего окна Vim в следующее, введите CTRL-W j (или CTRL-W <стрелка вниз>, или CTRL-W CTRL-J). CTRL-W является сокращением команды «window», а `j` аналогичен команде Vim `j`, перемещающей курсор на следующую строку.

В табл. 11.2 приведен обзор команд перехода между окнами.

Как и в случае многих других команд Vim и `vi`, их можно выполнить многократно, если приписать вначале числовой индекс. Например, `3^Wj` укажет Vim перейти на третье окно ниже текущего.

*Таблица 11.2. Команды перехода между окнами*

| Команда | Описание |
|---|---|
| CTRL-W <стрелка вниз><br>CTRL-W CTRL-J<br>CTRL-W j | Переход на окно ниже.<br>Обратите внимание, что эта команда не делает циклического перехода между окнами. Она просто выполняет переход в окно ниже текущего. Если курсор находится в самом нижнем окне экрана, то команда ничего не делает. Кроме того, на своем «пути назад» она не переходит в смежные окна: например, если справа от текущего окна расположено другое, то команда не выполнит переход на него (для циклического перехода воспользуйтесь CTRL-W CTRL-W). |
| CTRL-W <стрелка вверх><br>CTRL-W CTRL-K<br>CTRL-W k | Переход на окно выше. Действие команды противоположно CTRL-W j. |
| CTRL-W <стрелка влево><br>CTRL-W CTRL-H<br>CTRL-W h<br>CTRL-W <BS> | Переход на окно, расположенное слева от текущего. |
| CTRL-W <стрелка вправо><br>CTRL-W CTRL-L<br>CTRL-W l | Переход на окно справа от текущего. |
| CTRL-W w<br>CTRL-W CTRL-W | Переход на следующее окно снизу или справа. Обратите внимание, что эта команда, в отличие от CTRL-W j, производит циклический переход через все окна в Vim. При достижении самого нижнего окна Vim запустит цикл заново и переведет пользователя в окно, занимающее самое верхнее левое положение. |
| CTRL-W W | Переход на следующее окно, расположенное сверху или слева. Действие команды обратно CTRL-W w. |

| Команда | Описание |
|---|---|
| CTRL-W t<br>CTRL-W CTRL-T | Переход в верхнее левое окно. |
| CTRL-W b<br>CTRL-W CTRL-B | Переход на окно, занимающее самое нижнее правое положение. |
| CTRL-W p<br>CTRL-W CTRL-P | Переход на предыдущее окно (в котором вы находились до последнего перехода). |

### Мнемонические подсказки

`t` и `b` – сокращения для *верхнего* (*top*) и *нижнего* (*bottom*) окон.

Опираясь на соглашение о том, что прописные и строчные буквы реализуют противоположное действие, перемещение по окнам, вызываемое командой CTRL-W w, будет противоположно действию CTRL-W W.

Клавиша Control не различает прописные и строчные буквы. Другими словами, нажатие Shift при нажатом CTRL не даст никакого эффекта. Однако при нажатии второй клавиши (без Control) прописные и строчные буквы уже *будут* различаться.

# Перемещение окон

В Vim перемещать окна можно двумя способами. Первый просто меняет местами два окна на экране, второй меняет раскладку окон. В первом случае размеры окон остаются постоянными, а сами они лишь меняют свои позиции на экране. Во втором – окна не только перемещаются, но и меняют размер, чтобы заполнить то место, куда их передвинули.

## Перемещение окон (ротация или обмен)

Три команды перемещают окна, не меняя их раскладки. Две из них позиционно сдвигают окна: первая – направо или вниз, вторая – в противоположном направлении (налево или вверх), а третья обменивает позиции двух, возможно, несмежных окон. Эти команды действуют *только* в том столбце или ряду, в котором находится текущее окно.

Команда CTRL-W r циклически сдвигает окна вправо или вниз. Дополнением к ней служит команда CTRL-W R, сдвигающая окна в противоположном направлении.

Самый простой способ понять работу этих команд – представить, что ряд или столбец окон Vim – это одномерный массив. CTRL-W r сдвигает

каждый элемент этого массива на одну позицию вправо, а на первое освободившееся место помещает окно, стоявшее последним. CTRL-W R просто смещает все в обратном направлении.

Если в столбце или строке нет окон, выровненных с текущим, то эта команда ничего не делает.

После того как Vim переместит окна, курсор останется в том окне, из которого была вызвана команда перемещения, то есть переместится вместе с окном.

Команды CTRL-W x и CTRL-W CTRL-X позволяют менять местами два окна в ряду или столбце. По умолчанию Vim меняет текущее окно со следующим, а если следующего нет, то с предыдущим. Можно меняться с *n*-м окном, задав числовой параметр перед командой. Например, чтобы поменять текущее окно с идущим третьим после него, введите `3^Wx`.

Как и в двух предыдущих командах, курсор остается в том окне, откуда вызывалась команда обмена.

## Перемещение окон и изменение их раскладки

Пять команд перемещают и меняют раскладку окон: две перемещают текущее окно на самую верхнюю (или нижнюю) позицию и растягивают его до максимальной ширины, две другие перемещают текущее окно в крайнее правое или левое положение, распахивая его по вертикали, а последняя команда перемещает текущее окно в новую вкладку (см. раздел «Редактирование со вкладками» на стр. 222). Первые четыре команды имеют мнемонические связи с другими командами Vim: например CTRL-W K опирается на традиционную функцию `k` – «вверх». Эти сочетания обобщены в табл. 11.3.

*Таблица 11.3. Команды для перемещения и переформатирования окон*

| Команда | Описание |
|---|---|
| `^WK` | Помещает текущее окно на самый верх экрана, распахивая его во всю ширину. |
| `^WJ` | Помещает текущее окно в самый низ экрана, распахивая его во всю ширину. |
| `^WH` | Помещает текущее окно в самое левое положение экрана, распахивая его во всю высоту. |
| `^WL` | Помещает текущее окно в самое правое положение экрана, распахивая его во всю высоту. |
| `^WT` | Перемещает текущее окно в новую существующую вкладку. |

В точности описать действие этих команд сложно. После перемещения окна и изменения его размера Vim перераспределяет остальные окна приемлемым способом, чтобы ширина или высота перемещенного окна совпадали с шириной или высотой экрана. На это действие могут влиять некоторые опции окон.

## Команды перемещения окон: обзор

Таблицы 11.4 и 11.5 обобщают команды, изученные в этом разделе.

*Таблица 11.4. Команды циклического сдвига положений окон*

| Команда | Описание |
|---|---|
| `^Wr`<br>`^W^R` | Сдвигает окна вниз или вправо. |
| `^WR` | Сдвигает окна вверх или влево. |
| `^Wx`<br>`^W^X` | Обмен позициями со следующим окном или, при указании числа *n*, с *n*-м окном. |

*Таблица 11.5. Команды смены размера и формата окон*

| Команда | Описание |
|---|---|
| `^WK` | Помещает текущее окно на самый верх экрана и использует полную ширину экрана. Курсор остается в перемещенном окне. |
| `^WJ` | Помещает текущее окно в самый низ экрана и использует всю ширину экрана. Курсор остается в перемещенном окне. |
| `^WH` | Помещает текущее окно в крайнее левое положение экрана и использует всю высоту экрана. Курсор остается в перемещенном окне. |
| `^WL` | Помещает текущее окно в крайнее правое положение экрана и использует полную высоту экрана. Курсор остается в перемещенном окне. |
| `^WT` | Перемещает текущее окно в новую вкладку. Курсор остается в перемещенном окне. Если текущее окно – единственное в текущей вкладке, никаких действий не производится. |

# Изменение размера окна

Теперь, когда вы лучше знакомы с функциями работы с несколькими окнами в Vim, пора познакомиться с управлением ими. В этом разделе вы узнаете, как менять размер текущего окна, что, конечно, окажет влияние и на другие окна на экране. Чтобы управлять размерами окон и их поведением при появлении новых окон, Vim предоставляет специальные опции.

Если вы предпочитаете контролировать размеры окна, *не прибегая к командам*, используйте `gvim`, и пусть всю работу за вас выполнит мышь. Просто перетаскивайте границы окон с ее помощью, и размеры окна будут меняться автоматически. Для окон, разделенных по вертикали, щелкайте мышью над вертикальными разделителями – символами `|`. Горизонтальные окна разделяются своими строками состояния.

## Команды изменения размера окна

Как и следовало ожидать, в Vim есть команды для изменения вертикального и горизонтального размеров окон. Как и все другие команды для работы с окнами, они начинаются с CTRL-W и следуют привычным мнемоническим правилам, что делает их простыми в запоминании и использовании.

Команда CTRL-W = пытается сделать размер всех окон одинаковым (на это также влияют текущие значения параметров `winheight` и `windwidth`, о которых рассказано в следующем разделе). Если текущее состояние окон на экране не позволяет выровнять их размер, Vim устанавливает его как можно ближе к нужному.

CTRL-W – уменьшает высоту текущего окна на одну строку. Также в Vim есть команда `ex`, позволяющая явно уменьшить размер окна. Например, команда `resize -4` уменьшит высоту текущего окна на четыре строки, при этом размер стоящего ниже окна соответственно увеличится.

Интересно, что Vim покорно уменьшит размер окна, даже если вы не находитесь в многооконном режиме. Хотя на первый взгляд это может показаться нелогичным, но побочный эффект уменьшения окна по требованию состоит в том, что освободившаяся область добавляется к окну командной строки. Как правило, окно командной строки занимает одну строку, но есть ситуации, когда нужно его увеличить (самая распространенная причина – дать Vim достаточно места для отображения полного состояния и отклика командной строки без промежуточных приглашений[1]). Тем не менее команду `:resize` лучше применять для смены размера текущего окна, а для установки размера окна командной строки лучше пользоваться опцией `cmdheight`.

CTRL-W + увеличивает высоту текущего окна на одну строку, а команда `:resize +`*n* увеличивает высоту на *n* строк. Если достигнут максимум высоты окна, то дальнейший вызов команды ничего не меняет.

Один из авторов предпочитает отобразить команды CTRL-W + и CTRL-W – на две смежные клавиши. Удобно взять для этого кнопку +. Хотя она уже используется в Vim как команда «вниз», это является избыточным[2] и не используется ветеранами Vim (они для этого применяют команду `j`). Следовательно, эта клавиша – хороший кандидат для отображения чего-нибудь другого, в нашем случае –

[1] Если результат выполнения какой-либо команды (например, упомянутой ниже `:buffers`) не помещается в окне командной строки, Vim выводит приглашение «Press ENTER or type command to continue» (нажмите ENTER или введите команду для продолжения). По нажатию результаты исчезают. – *Прим. науч. ред.*

[2] Строго говоря, + и `j` не эквивалентны: + автоматически помещает курсор на первый печатный символ в следующей строке. Аналогами + служат Enter и ^M. – *Прим. науч. ред.*

CTRL-W +. Слева от + (на большей части клавиатур) располагается клавиша –. Однако – (минус) вводится без Shift, а + – с Shift, значит, отобразить нужно на клавишу с Shift, то есть _. Теперь у вас есть две удобные, расположенные рядом кнопки, которые быстро и легко расширяют и сокращают текущее окно по горизонтали.

`:resize` *n* устанавливает горизонтальный размер текущего окна равным *n* строкам. В отличие от ранее рассмотренных команд, которые задавали относительное изменение, эта устанавливает абсолютное значение.

`z`*n* устанавливает высоту текущего окна равной *n*. Обратите внимание, что аргумент *n обязателен*! Если про него забыть, это приведет к выполнению команды `vi`/Vim `z`, перемещающей курсор наверх экрана.

CTRL-W < и CTRL-W > уменьшают и увеличивают ширину окна соответственно. Вспомните мнемоническое правило «сдвига влево» (`<<`) и «сдвига вправо» (`>>`), чтобы запомнить действие этих команд.

Наконец, команда CTRL-W | изменяет размер окна так, чтобы оно приобрело максимально возможную ширину (значение по умолчанию[1]). Величину изменения ширины окна можно указать с помощью `vertical resize` *n*, где *n* задает новую ширину окна.

## Опции при изменении размеров окон

Несколько опций Vim влияют на результат команд изменения размера окна, описанных в предыдущем разделе.

Когда окно становится активным, `winheight` и `winwidth` определяют его минимальную высоту и ширину соответственно. Например, если экран вмещает два одинаковых окна из 45 строк, то по умолчанию Vim попытается разделить экран поровну. Если установить `winheight` большим, чем 45, например 60, то всякий раз при переходе в новое окно редактор будет устанавливать высоту этого окна равной 60 строкам, а высоту другого – 30. Это удобно при одновременном редактировании двух файлов, так как позволяет автоматически увеличивать размер выделенного окна до максимального при перемещении от окна к окну и из одного файла в другой.

`equalalways` говорит Vim всегда делать размеры окон одинаковыми после разделения или закрытия окна. Это хороший вариант, когда нужно получить разумную раскладку окон по мере их добавления и удаления.

`eadirection` определяет направления, в которых действует опция `equalalways`. Возможные значения: `hor`, `ver` и `both`. Они приказывают Vim делать окна одного размера по *горизонтали*, *вертикали* и в *обоих* направлениях соответственно. Изменение размера происходит всякий раз при разделении и удалении окна.

---

[1] Необязательный числовой параметр, предшествующий команде, устанавливает ширину окна равной этому конкретному значению. Для задания высоты окна используется команда ^W_; принцип действия аналогичен. – *Прим. науч. ред.*

`cmdheight` устанавливает высоту командной строки. Как уже описывалось выше, уменьшение высоты окна, когда оно является единственным, увеличивает высоту командной строки. С помощью этой опции ее размер можно зафиксировать.

Наконец, `winminwidth` и `winminheight` определяют *минимальную* ширину и высоту окон в Vim. Он рассматривает их как жесткие условия, т. е. окнам никогда не будет позволено иметь размер меньше, чем эти значения.

## Обзор команд изменения размера

В табл. 11.6 обобщены способы изменения размеров окон. Эти опции устанавливаются с помощью команды `:set`.

*Таблица 11.6. Команды изменения размера окна*

| Команда или опция | Описание |
|---|---|
| `^W=` | Делает размеры всех окон одинаковыми. Текущее окно соблюдает установки `winheight` и `winwidth`. |
| `:resize -n`<br>`^W-` | Уменьшает размер окна. Изменение по умолчанию равно одной строке. |
| `:resize +n`<br>`^W+` | Увеличивает размер окна. Изменение по умолчанию равно одной строке. |
| `:resize n`<br>`^W^_ ^W_` | Устанавливает высоту текущего окна. По умолчанию размер делается максимально большим (если не указан *n*). |
| `zn` <ENTER> | Устанавливает высоту текущего окна равной *n*. |
| `^W<` | Уменьшает ширину текущего окна. Изменение по умолчанию – один столбец. |
| `^W>` | Увеличивает ширину текущего окна. Изменение по умолчанию – один столбец. |
| `:vertical resize n`<br>`^W\|` | Устанавливает ширину текущего окна равной *n*. По умолчанию окно становится максимально широким. |
| `winheight`, опция | При переходе в окно или при его создании устанавливает его высоту по меньшей мере равной заданному значению. |
| `winwidth`, опция | При переходе в окно или при его создании устанавливает его ширину по меньшей мере равной заданному значению. |
| `equalalways`, опция | Если изменяется количество окон из-за разделения или закрытия окна, старается сделать их размер одинаковым. |
| `eadirection`, опция | Определяет способ выравнивания размеров окон в Vim: по горизонтали, по вертикали или по обоим параметрам. |
| `cmdheight`, опция | Устанавливает высоту командной строки. |
| `winminheight`, опция | Определяет минимальную высоту окна, которая затем применяется ко всем создаваемым окнам. |
| `winminwidth`, опция | Определяет минимальную ширину окна, которая затем применяется ко всем создаваемым окнам. |

# Буферы и их взаимодействие с окнами

Vim использует *буферы* как контейнеры во время работы. Для полного понимания их сущности необходима большая практика. Для управления буферами и перемещения между ними существует множество команд, однако стоит познакомиться с некоторыми основными понятиями, касающимися буферов, и понять, зачем и как они присутствуют в сеансе работы Vim.

Начнем с открытия нескольких окон с разными файлами. Например, запустите Vim, открыв file1, затем внутри этого сеанса введите :split file2, а потом :split file3. Появятся три файла, открытые в трех разных окнах Vim.

Теперь введите команды :ls, :files или :buffers, чтобы увидеть список буферов. Редактор выдаст три пронумерованных строки, содержащих имя файла и дополнительную информацию. Это буферы Vim для текущего сеанса. Каждому файлу соответствует свой буфер, каждый буфер имеет уникальный, неизменяемый, связанный с ним номер. В нашем примере file1 находится в буфере 1, file2 – в буфере 2 и т. д.

Если после любой из трех перечисленных выше команд поставить восклицательный знак (!), то будет выведена дополнительная информация по всем буферам.

Справа от номера каждого буфера стоит флаг состояния. Эти флаги описывают буфер, как показано в табл. 11.7.

*Таблица 11.7. Флаги состояния, описывающие буферы*

| **Код** | **Описание** |
|---|---|
| u | Неотображаемый буфер. Такой буфер не появится в списке, если только вы не используете параметр !. Чтобы увидеть пример неотображаемого буфера, введите :help. Vim разделит текущее окно, после чего в новом окне появится встроенная справка. Простой ввод команды :ls не покажет буфер справки, но :ls! его выведет. |
| % или (взаимоисключающее) # | % – это буфер текущего окна, а # указывает буфер, в который можно переключиться с помощью команды :edit #. |
| a или (взаимоисключающее) h | a указывает на активный буфер. Это значит, что буфер загружен и видим. h указывает на скрытый буфер. Скрытые буферы существуют, но не видны ни в одном окне. |
| - или (взаимоисключающее) = | Символ - означает, что для данного буфера выключена опция редактирования modifiable. Файл открыт только для чтения. Символ = означает, что буфер нельзя сделать редактируемым (например, из-за отсутствия у пользователя прав на запись этого файла). |
| + или (взаимоисключающее) x | + указывает, что буфер изменен, x – буфер содержит ошибки чтения. |

Флаг `u` предоставляет интересный способ узнать, какой файл справки вы читаете в Vim в данный момент. Например, если вызвать `:help`, а затем `:ls!`, то станет видно, что неотображаемый буфер ссылается на `windows.txt` – встроенный файл справки Vim.

Теперь, когда вы умеете выводить список буферов Vim, можно поговорить о различных способах использования этих буферов.

## Специальные буферы Vim

Vim применяет некоторые буферы для своих собственных целей. Эти буферы называются *специальными*. Например, буферы справки, описанные в предыдущем разделе, являются специальными. Как правило, подобные буферы нельзя менять и редактировать.

Вот примеры четырех специальных буферов Vim:

*quickfix*

Содержит список ошибок, созданный вашими командами[1] (их можно увидеть командой `:cwindow`), или список положений (его можно увидеть командой `:lwindow`). Не редактируйте содержимое этого буфера! Он помогает программистам заново проходить по циклу «редактирование-компиляция-отладка». Более подробно об этом написано в главе 14.

*help*

Содержит файлы справки Vim, про которые рассказывалось в разделе «Встроенная справка» на стр. 187. `:help` загружает их в специальные буферы.

*directory*

Представляет собой содержимое каталога, то есть список файлов в каталоге (и немного дополнительных подсказок по командам). Vim позволяет перемещаться по этому буферу как по обычному текстовому файлу и выбирать файлы под курсором для редактирования нажатием ENTER, что очень удобно.

*scratch*

Эти буферы содержат текст для общих целей. Этот текст не сохраняется, и его можно удалить в любой момент.

## Скрытые буферы

Скрытые буферы – это такие буферы Vim, которые в данный момент не отображены ни в одном окне. Это позволяет редактировать несколько файлов, принимая в учет ограниченность реального экрана, без постоянного перезаписывания и считывания файлов. Например, представь-

---

[1] Например, `:make`, упомянутая в главе 8. – *Прим. науч. ред.*

те, что вы редактируете файл `myfile`, но вскоре решаете поработать над другим файлом – `myOtherfile`. Если установлена опция `hidden`, то можно начать редактирование `myOtherfile`, вызвав `:edit myOtherfile`. Эта команда заставит Vim скрыть буфер `myfile` и показать на его месте `myOtherfile`. Это можно проверить, выполнив `:ls` и увидев оба буфера в списке, причем `myfile` будет помечен как *скрытый*.

## Команды работы с буферами

Существует почти 50 команд, нацеленных исключительно на буферы. Многие из них полезны, но большая их часть выходит за рамки нашего обсуждения. Когда вы открываете и закрываете несколько файлов и окон, Vim управляет буферами автоматически. Комплект команд работы с буферами позволяет делать с ними все, что угодно. Часто их используют в скриптах при выполнении таких задач, как выгрузка, удаление и изменение буфера.

Две команды работы с буферами полезно знать для общих целей из-за их способности выполнять много работы над многими файлами:

`windo` *`cmd`*

Сокращение от «window do» (мы, по крайней мере, считаем это неплохой мнемоникой). Эта «псевдобуферная» команда (на самом деле она относится к командам работы с окнами) выполняет команду *cmd* в каждом окне. Она работает аналогично переходу наверх экрана (`^Wt`) и проходит через все окна, выполняя в каждом указанную команду в формате `:cmd`. Команда работает только в текущей вкладке и прекращает свое выполнение, как только `:cmd` приводит к ошибке. Окно, где произошла эта ошибка, становится текущим.

Команде `cmd` не разрешается менять состояние окон, то есть она не может удалять, создавать и менять их порядок.

`cmd` может быть составлена из нескольких команд, разделенных вертикальной чертой (`|`). *Не путайте это обозначение с соглашением, принятым в UNIX о конвейерных командах!* Здесь команды выполняются последовательно, причем сначала во всех окнах выполняется первая команда, затем – вторая и т. д.

В качестве примера работы `:windo` предположим, что вы редактируете пакет файлов Java, и по какой-то причине у вас произошла неправильная капитализация имени класса. Это необходимо исправить, поменяв каждое вхождение `myPoorlyCapitalizedClass` на `MyPoorlyCapitalizedClass`. Такую операцию можно проделать при помощи `:windo`:

```
:windo %s/myPoorlyCapitalizedClass/MyPoorlyCapitalizedClass/g
```

Прекрасно!

bufdo [!] *cmd*

Команда аналогична windo, но работает во всех буферах сеанса редактирования, а не только в видимых буферах текущей вкладки. Как и windo, команда bufdo прекращает работу на первой ошибке и оставляет курсор в том буфере, где произошла эта ошибка.

В следующем примере все буферы преобразуются в файловый формат UNIX:

```
:bufdo set fileformat=unix
```

## Обзор команд работы с буферами

Мы не ставили целью описать в табл. 11.8 все команды, связанные с буферами. В ней приведены команды, о которых рассказано в этом разделе, а также некоторые другие из популярных.

*Таблица 11.8. Обзор команд работы с буфером*

| Команда | Описание |
|---|---|
| :ls[!]<br>:files[!]<br>:buffers[!] | Выводит список всех буферов и имен файлов. Неотображаемые буферы выводятся, если стоит модификатор !. |
| :ball<br>:sball | Редактировать все файлы в списке аргументов Vim (args) или буферы (sball откроет их в новых окнах). |
| :unhide<br>:sunhide | Редактировать все *загруженные* буферы (sunhide откроет их в новых окнах). |
| :badd file | Добавляет file в список буферов. |
| :bunload[!] | Выгружает буфер из памяти. Модификатор ! заставляет выгрузить даже измененный буфер без сохранения. |
| :bdelete[!] | Выгружает буфер и удаляет его из списка буферов. Модификатор ! выгружает измененный буфер без сохранения. |
| :buffer [*n*]<br>:sbuffer [*n*] | Переход в буфер *n* (sbuffer откроет новое окно). |
| :bnext [*n*]<br>:sbnext [*n*] | Переход в следующий по порядку *n*-й буфер (sbnext откроет новое окно). |
| :bNext [*n*]<br>:sbNext [*n*]<br>:bprevious [*n*]<br>:sbprevious [*n*] | Переход в *n*-й предыдущий буфер. (sbNext и sbprevious откроют новое окно.) |
| :bfirst<br>:sbfirst | Переход к первому буферу (sbfirst откроет новое окно). |
| :blast<br>:sblast | Переход к последнему буферу (sblast откроет новое окно). |
| :bmod [*n*]<br>:sbmod [*n*] | Переход к *n*-му измененному буферу (sbmod откроет новое окно). |

# Теги и окна

Vim расширяет функциональность тегов vi на окна, предоставляя тот же механизм перехода по тегам, но уже для нескольких окон. Переход по тегу откроет файл в новом окне в соответствующем месте.

Оконные команды для работы с тегами разделяют текущее окно и переходят либо на файл, соответствующий тегу, либо на файл, соответствующий имени файла под курсором.

:stag[!] *tag* разделяет окно, чтобы отобразить местоположение найденного тега. Окно с новым файлом, содержащим соответствующий тег, становится текущим, а курсор помещается на соответствующий тег. Если тег не найден, команда завершается, и новое окно не создается.

Когда вы освоитесь со справочной системой Vim, то сможете использовать команду :stag, чтобы открывать каждый новый термин справки в отдельном окне вместо перехода от одного файла к другому в одном и том же окне.

^W] или ^W^] разделяет окно и открывает новое окно над текущим. При этом новое окно становится текущим, а курсор помещается на соответствующий тег. Если соответствие тегу не найдено, команда завершает работу.

^Wg] разделяет окно и создает новое окно над текущим. В новом окне Vim выполняет команду :tselect *tag*, где *tag* – это идентификатор тега под курсором. Если соответствующего тега не обнаружено, команда прекращает работу. Курсор помещается в новое окно, которое становится текущим.

^Wg^] работает аналогично ^Wg], но выполняет :tselect вместо tjump.

^Wf (или ^W^F) разделяет окно и редактирует имя файла под курсором. Vim последовательно просмотрит файлы, прописанные в опции path при поиске этого файла. Если файл не существует ни в одном из каталогов path, команда прекращает работу, не создавая нового окна.

^WF разделяет окно и редактирует имя файла под курсором. Он помещается в новое окно с этим файлом на ту строку, которая соответствует номеру, стоящему после имени файла в первом окне.

^Wgf открывает файл под курсором в новой вкладке. Если файл не существует, вкладка не создается.

^WgF открывает файл под курсором в новой вкладке и помещает курсор в строке, номер которой стоял после имени файла в первом окне. При отсутствии файла вкладка не создается.

## Редактирование с вкладками

Вы знали, что кроме редактирования нескольких окон можно создавать несколько *вкладок*? Vim позволяет создавать новые вкладки, каждая из которых ведет себя независимо от других. В каждой вкладке можно разделять экран, редактировать несколько файлов, в общем, теоретически делать все, что можно проделывать в одном окне, однако теперь управлять работой еще проще, пользуясь одним окном с вкладками.

Многие пользователи Firefox хорошо знакомы с вкладками[1] и уже не представляют себе веб-серфинга без них. Поэтому они согласятся, что эта возможность добавит функциональности редактированию. Тем, кто с этим не знаком, стоит попробовать.

Вкладки можно использовать как в обычном Vim, так и в `gvim`, однако в `gvim` они выглядят приятнее и удобнее. Некоторые из наиболее важных способов создания и управления вкладками включают:

`:tabnew` *`filename`*

Открывает новую вкладку и редактирует в ней файл (необязательно). Если файл не указан, Vim создаст новую вкладку с пустым буфером.

`:tabclose`

Закрывает текущую вкладку.

`:tabonly`

Закрывает все остальные вкладки. Если в них находятся измененные файлы, они не будут закрыты, кроме случая, когда установлена опция `autowrite` (тогда все измененные файлы будут сохранены перед закрытием вкладок).

В `gvim` вкладку можно активировать простым щелчком по «корешку» вверху экрана. В символьных терминалах вкладки тоже можно активировать с помощью мыши, если она настроена (обратитесь к опции `mouse`). Также можно очень быстро перемещаться от одной вкладки к другой с помощью CTRL PAGE DOWN (переход на вкладку справа) и CTRL PAGE UP (переход на вкладку слева). Если вы попали на самую левую или самую правую вкладки и пытаетесь переместиться дальше влево или вправо, то Vim переместит вас на самую правую или самую левую вкладки соответственно.

`gvim` предоставляет всплывающее по правой кнопке мыши меню, из которого можно открыть новую вкладку (с новым файлом или без такового) и закрыть вкладку.

На рис. 11.4 приведен пример набора вкладок (обратите внимание на всплывающее меню вкладки).

---

[1] Впрочем, сейчас эту функцию предлагают едва ли не все браузеры. – *Прим. науч. ред.*

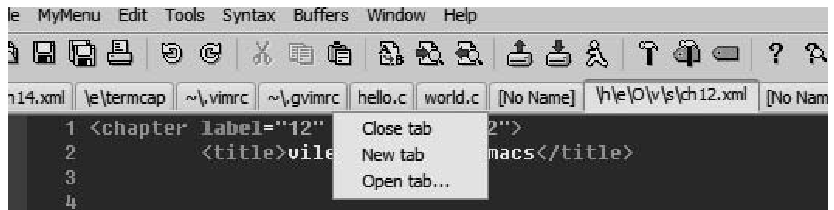



*Рис. 11.4. Пример вкладок и редактирования с вкладками в gvim*

# Закрытие и выход из окон

Существует четыре различных способа закрыть окно, специфичных для многооконного редактирования: *выйти*, *закрыть*, *скрыть* и *закрыть остальные*.

^Wq (или ^W^Q, или :quit) – это оконная версия команды :quit. В самом простом случае (один сеанс редактирования с единственным окном) она ведет себя так же, как и команда vi :quit. Если установлена опция hidden и текущее окно является последним из окон на экране, ссылающимся на данный файл, то окно закрывается, однако буфер этого файла сохраняется (к нему можно обратиться) и скрывается. Другими словами, Vim все еще хранит файл, поэтому позже можно вернуться к редактированию. Если hidden *не установлена*, а окно является последним, содержащим этот файл, и в буфере текущего окна есть несохраненные изменения, то команда прекращает работу, дабы не потерять изменения. Однако если файл отображается в каком-нибудь другом окне, текущее окно закрывается.

^Wc (или :close[!]) закрывает текущее окно. Если установлена опция hidden и это окно – последнее, ссылающееся на данный файл, то Vim закрывает окно и скрывает буфер. Если окно расположено на странице вкладки и является последним окном на этой вкладке, то закроется как окно, так и вкладка. Если вы не используете модификатор !, то эта команда не станет закрывать никакой файл с несохраненными изменениями. Модификатор ! говорит Vim безоговорочно закрыть текущее окно.

Обратите внимание, что данная команда не имеет вариации ^W^C, так как Vim использует ^C в качестве *отмены* команды. Следовательно, если вы попробуете использовать ^W^C, то ^C попросту отменит команду.

Аналогично при использовании команды ^W в сочетании с ^S и ^Q можно столкнуться с тем, что их терминалы заморозились, поскольку *некоторые* терминалы интерпретируют ^S и ^Q как управляющие символы для остановки и возобновления отображения информации на экране. Если вы столкнулись с загадочной заморозкой при использовании этих команд, попробуйте взять другие приведенные здесь комбинации.

^Wo, ^W^O и :only[!] закрывают все окна кроме текущего. Если установлена опция hidden, то все закрываемые окна скрывают свои буферы. Если она не установлена, то любое окно, содержащее файл с несохраненными изменениями, остается на экране, если только не используется модификатор !. Тогда все окна закрываются с отказом от изменений файлов. Поведение этой команды также определяется опцией autowrite: если она установлена, все окна закрываются. При этом окна, содержащие несохраненные изменения, перед закрытием сохраняют содержимое своих файлов на диск.

:hide [*cmd*] выходит из текущего окна и скрывает буфер, если только на него не ссылаются другие окна. Если стоит необязательная команда *cmd*, то буфер скрывается и выполняется эта команда.

В табл. 11.9 дан обзор этих команд.

*Таблица 11.9. Команды для закрытия и выхода из окон*

| Команда | Описание |
|---|---|
| :quit[!]<br>^Wq<br>^W^Q | Выход из текущего окна. |
| :close[!]<br>^Wc | Закрытие текущего окна. |
| :only[!]<br>^Wo<br>^W^O | Текущее окно становится единственным. |

# Итог

Как вы теперь понимаете, с помощью многооконных функций Vim добавляет себе «лошадиных сил» по части редактирования. Он позволяет создавать и удалять окна быстро, на лету. Кроме того, у Vim есть в рукаве команды для работы напрямую с буферами, которые являются главным механизмом управления файлами, с помощью которого Vim производит редактирование с несколькими окнами. Это еще один пример того, как Vim предоставляет многооконное редактирование новичкам, одновременно предлагая опытным пользователям инструменты, позволяющие подстроить работу с окнами под свои потребности.

# 12

# Скрипты Vim

Иногда для создания требуемого окружения редактора недостаточно одной настройки. Vim позволяет определить все установки в файле `.vimrc`, но, возможно, вам захочется более динамичной или «актуальной» настройки. Скрипты Vim позволяют это сделать.

Они дают возможность выполнять сложные задачи и принимать решения согласно *вашим* потребностям, от проверки содержимого буфера до обработки непредвиденных внешних факторов.

Если у вас есть конфигурационный файл Vim (`.vimrc`, `.gvimrc` или оба), значит вы уже используете скрипты Vim, но просто этого не знаете. Все команды и опции Vim работают и в скриптах. Кроме того, как можно было предположить, Vim предоставляет стандартное управление выполнением программы (`if...then...else`, `while` и т. д.), переменные и функции, типичные для других языков.

В этой главе мы рассмотрим на примере пошаговое создание скрипта и простые конструкции, воспользуемся некоторыми встроенными в программу функциями и изучим правила, которых следует придерживаться при написании правильных и предсказуемых скриптов Vim.

## Какой ваш любимый цвет?

Начнем с самых простых настроек и подстроим окружение под *свою* любимую цветовую схему. Это просто, к тому же здесь используется только базовый элемент скрипта Vim, то есть просто команда редактора.

Vim поставляется с 17 настраиваемыми цветовыми схемами. Цветовую схему можно выбирать и активировать, прописав в файле `.vimrc` или `.gvimrc` команду `colorscheme`. Самой любимой «недооцененной» схемой одного автора является «пустынная» (desert) схема:

```
colorscheme desert
```

Вставьте `colorscheme` подобного вида в свой конфигурационный файл, после чего при каждом старте Vim вы увидите свои любимые цвета.

Конечно, наш первый скрипт тривиален. А что если у вас более сложные запросы в цветовой схеме? Что если вам нравится несколько схем? Если ваши вкусы зависят от времени суток? Скрипты Vim позволяют с легкостью настроить цвета под нужды пользователя.

Выбор цветовой схемы в зависимости от времени суток может показаться банальным, но не настолько, как выглядит на первый взгляд. Даже Google меняет цвета и тоны вашей домашней страницы *iGoogle* в течение дня.

## Условное выполнение

Один из авторов предпочитает делить сутки на четыре части, для каждой назначая свою цветовую схему.

`darkblue`
: От полуночи до 6 утра.

`morning`
: От 6 утра до полудня.

`shine`
: От полудня до 6 вечера.

`evening`
: От 6 вечера до полуночи.

Для этой цели мы создадим вложенный блок `if...then...else...`, в котором можно использовать два различных синтаксиса. Один из них является более общепринятым, с явной раскладкой:

```
if cond expr
  line of vim code
  another line of vim code
  ...
elseif some secondary cond expr
  code for this case
else
  code that runs if none of the cases apply
endif
```

Блоки `elseif` и `else` необязательны, и можно включать несколько блоков `elseif`. Vim также допускает более сжатую конструкцию в стиле C:

```
cond ? expr 1 : expr 2
```

Редактор проверяет условие *cond*. Если оно истинно, то выполняется *expr 1*, иначе – *expr 2*.

## Использование функции strftime( )

Теперь, когда мы можем осуществлять условное выполнение кода, пора вычислять время суток. В Vim есть встроенные *функции*, возвращающие информацию подобного рода. В данном случае мы воспользуемся функцией `strftime()`. В ней передается два параметра, первый из которых задает формат времени (он зависит от системы[1] и не является портируемым, так что при его выборе нужно соблюдать осторожность. К счастью, большинство основных форматов одинаковы во всех системах). Второй необязательный параметр – это время в секундах, прошедшее с первого января 1970 года (стандартное представление времени в C). Этот необязательный параметр по умолчанию соответствует текущему времени. В нашем примере можно использовать формат `%H`, что приведет к `strftime("%H")`, поскольку для определения цветовой схемы нам нужно лишь знать, который час.

Теперь мы знаем, как использовать условный код и встроенные функции Vim, дающие информацию о времени суток, с помощью которой мы будем выбирать соответствующую цветовую схему. Вставьте в файл `.vimrc` следующий код:

```
" progressively check higher values... falls out on first "true"
" (note addition of zero ... this guarantees return from function is numeric
if strftime("%H") < 6 + 0
  colorscheme darkblue
  echo "setting colorscheme to darkblue"
elseif strftime("%H") < 12 + 0
  colorscheme morning
  echo "setting colorscheme to morning"
elseif strftime("%H") < 18 + 0
  colorscheme shine
  echo "setting colorscheme to shine"
else
  colorscheme evening
  echo "setting colorscheme to evening"
endif
```

Обратите внимание, что мы использовали новую команду для скриптов Vim – `echo`. Для удобства мы сообщаем самим себе о выбранной схеме. Это также помогает убедиться, что код на самом деле выполнен и дал нужный результат. Сообщение появится в окне состояния Vim или в виде всплывающего окна в зависимости от того, на каком этапе стартовой последовательности встретилась команда `echo`.

При использовании команды `colorscheme` имя схемы (то есть `desert`) *не заключается* в кавычки, однако в команде `echo` оно должно стоять в кавычках (`"desert"`). Это важное отличие!

[1] На самом деле, Vim просто вызывает для вас функцию `strftime()` стандартной системной библиотеки C, а различия между этими библиотеками оказывают большое влияние на результат. – *Прим. науч. ред.*

> В случае команды `colorscheme` рассматриваемого скрипта мы вызываем непосредственно команду Vim. Ее аргумент должен быть буквальным. Если поставить кавычки, то они будут восприняты командой `colorscheme` как часть имени цветовой схемы. Это приведет к ошибке, ведь ни одна цветовая схема не содержит в своем имени кавычек.
>
> С другой стороны, команда `echo` воспринимает слова без кавычек как выражения (вычисления, возвращающие значения) или функции. Следовательно, имя выбранной нами цветовой схемы должно быть заключено в кавычки.

## Переменные

Если вы программист, то, возможно, заметили, что предложенный скрипт несет в себе проблему. Конечно, в данном случае это едва ли будет иметь какие-то последствия, но мы выполняем проверку условия часа, вызывая функцию `strftime()` каждый раз. Технически мы проверяем одно условие, но вычисляем выражение много раз, поэтому потенциально решение зависит от значения, которое может измениться в процессе выполнения.

Вместо того чтобы каждый раз вычислять значение функции, вычислим ее один раз и сохраним значение в *переменной* скрипта Vim. Затем это значение может использоваться в условии сколько угодно раз без многократных вызовов функции.

Переменные Vim довольно просты, однако есть несколько вещей, которые нужно знать и уметь. Необходимо управлять *областью действия* переменных. Редактор определяет область видимости переменных по соглашению, согласно которому она зависит от префикса имени. Существуют следующие префиксы:

`b:`

Переменная распознается в одном буфере Vim.

`w:`

Переменная распознается в одном окне Vim.

`t:`

Переменная распознается в одной вкладке Vim.

`g:`

Переменная распознается глобально, то есть к ней можно обращаться *отовсюду*.

`l:`

Переменная распознается внутри функции (локальная переменная).

`s:`

Переменная распознается внутри исходного кода скрипта Vim.

`a:`

Аргумент функции.

`v:`

Переменная Vim – управляется редактором (это также глобальные переменные).

Если вы не задаете область видимости посредством префикса, то переменная считается глобальной (`g:`), если она определена за пределами функции, и локальной (`l:`), если – внутри функции.

Значение переменной присваивается командой `let`:

```
:let var = "value"
```

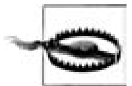
В наших целях можно определить переменную любого типа (допустимого контекстом), поскольку мы используем ее только один раз (хотя в дальнейшем будет по-другому). На данный момент мы не будем ставить никакого префикса, поэтому Vim по умолчанию посчитает эту переменную глобальной. Назовем ее `currentHour`. Присвоив ей результат вызова функции `strftime()` только один раз, мы получим более эффективный скрипт:

```
" progressively check higher values... falls out on first "true"
" (note addition of zero ... this guarantees return from function is numeric)
let currentHour = strftime ("%H")
echo "currentHour is " currentHour
if currentHour < 6 + 0
  colorscheme darkblue
  echo "setting colorscheme to darkblue"
elseif currentHour < 12 + 0
  colorscheme morning
  echo "setting colorscheme to morning"
elseif currentHour < 18 + 0
  colorscheme shine
  echo "setting colorscheme to shine"
else
  colorscheme evening
  echo "setting colorscheme to evening"
endif
```

Можно еще немного подчистить код, если избавиться от нескольких строк, введя переменную `colorScheme`. В ней содержится значение цветовой схемы, определяемой временем суток. Прописная S поставлена, чтобы отличить эту переменную от имени команды `colorscheme`, хотя если бы мы использовали точно такое же имя, ничего бы не изменилось, так как Vim может по контексту определить, что является командой, а что – переменной.

Обратите внимание на использование точки (`.`) в команде `echo`. Этот оператор объединяет выражения в одну строку, которую затем выводит `echo`. В нашем случае мы создаем строку из символов `"setting color scheme to "` и значения, которое присвоено переменной `colorScheme`.

```
" progressively check higher values... falls out on first "true"
" (note addition of zero ... this guarantees return from function is numeric
let currentHour = strftime("%H")
echo "currentHour is " . currentHour
if currentHour < 6 + 0
  let colorScheme ="darkblue"
elseif currentHour < 12 + 0
  let colorScheme = "morning"
elseif currentHour < 18 + 0
  let colorScheme = "shine"
else
  let colorScheme = "evening"
endif
echo "setting color scheme to" . colorScheme
colorscheme colorScheme
```



В этом скрипте команды выполняются не так, как мы предполагали. Если вы пробовали запустить этот пример, то уже это увидели. В следующем разделе мы исправим свою ошибку.

## Команда execute

Пока что мы только улучшили выбор цветовой схемы, но последнее изменение породило неожиданный поворот. Изначально мы хотели задавать цветовую схему в зависимости от времени суток. Последнее усовершенствование выглядит правильным, но после определения переменной (`colorScheme`) для хранения значения цветовой схемы мы увидим, что команда

```
colorscheme colorScheme
```

приводит к ошибке, показанной на рис. 12.1.

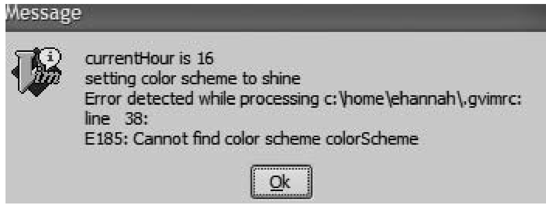



***Рис. 12.1.*** *Ошибка colorscheme colorScheme*

Необходим способ выполнения команд Vim, в котором можно сослаться на переменную, а не на буквальное значение, такое как `darkblue`. Для этой цели редактор предоставляет команду `execute`. Когда ей передают команду, она вычисляет переменные и выражения и подставляет их значения в эту команду. Мы используем это, а также сложение строк,

с которым мы встретились в предыдущем разделе, чтобы передать значение переменной команде colorscheme:

```
execute "colorscheme " . colorScheme
```

Используемый здесь синтаксис (особенно кавычки) может привести к замешательству. Команда execute принимает переменные и выражения, но colorscheme не является ни переменной, ни выражением. Это просто строка. Нам нужно, чтобы execute не вычисляла colorscheme, а приняла это имя как есть. Для этого мы превращаем имя команды в символьную строку, заключив его в кавычки. Перед закрывающей кавычкой поставим дополнительный пробел. Это необходимо, так как между командой и аргументом должен стоять пробел.

Наша переменная colorScheme должна стоять *за пределами* кавычек, то есть она должна вычисляться. Действие execute можно рассматривать как:

- Слова без кавычек вычисляются как переменные или выражения, и execute подставляет полученные значения.
- Строки, заключенные в кавычки, воспринимаются буквально. execute не пытается вычислить их, чтобы получить значение.

Использование execute исправит нашу ошибку, после чего Vim загрузит цветовую схему как следует.

После загрузки Vim можно проверить, загружена ли правильная цветовая схема. Команда colorscheme устанавливает собственную переменную под названием colors_name. Вдобавок к отображению значений переменных командой echo в скрипте, можно вызвать echo вручную и узнать значение переменной colors_name. Тогда можно увидеть, вызвал ли наш скрипт нужную команду colorscheme, основываясь на времени суток:

```
echo colors_name
```

## Определение функций

Мы создали прекрасно работающий скрипт. Теперь превратим его в код, который можно выполнять в любое время сеанса редактирования, а не только при старте Vim. Пример этого действия приведем чуть позже, а сейчас необходимо написать *функцию,* содержащую код рассматриваемого скрипта.

Используя конструкцию function...endfunction, в Vim можно определять собственные функции. Вот примерный каркас пользовательской функции:

```
function myFunction (arg1, arg2...)
  line of code
  another line of code
endfunction
```

Наш скрипт легко можно сделать функцией. Заметим, что в нее не нужно передавать никаких аргументов, то есть скобки в определении функции будут пустыми:

```
function SetTimeOfDayColors()
  " progressively check higher values... falls out on first "true"
  " (note addition of zero ... this guarantees return from function is numeric)
  let currentHour = strftime("%H")
  echo "currentHour is " . currentHour
  if currentHour < 6 + 0
    let colorScheme = "darkblue"
  elseif currentHour < 12 + 0
    let colorScheme = "morning"
  elseif currentHour < 18 + 0
    let colorScheme = "shine"
  else
    let colorScheme = "evening"
  endif
  echo "setting color scheme to" . colorScheme
  execute "colorscheme " . colorScheme
endfunction
```

Пользовательские функции Vim должны начинаться с прописной буквы.

Сейчас у нас есть функция, определенная в файле .gvimrc. Однако если ее не вызвать, то ее код выполняться не будет. Вызвать функцию можно с помощью оператора call. В нашем примере это будет выглядеть так:

```
call SetTimeOfDayColors()
```

Теперь цветовую схему можно устанавливать в любое время, из любого места сеанса работы в Vim. Одна из возможностей – вставить последнюю строку с call в .gvimrc. Результат будет таким же, как и в примере, где запускался код без определения функции. Однако в следующем разделе мы познакомимся с изящным трюком в Vim, при котором функция будет вызываться многократно, так что цветовая схема будет правильно устанавливаться на протяжении всего сеанса работы, динамически меняясь в течение всего дня! Конечно, при этом мы столкнемся с новыми проблемами.

## Интересный трюк

В предыдущем разделе мы определили функцию Vim SetTimeOfDayColors(), которая вызывается один раз и устанавливает цветовую схему. А что если проверять время суток неоднократно и менять цветовую схему соответственно? Очевидно, однократный вызов из .gvimrc не позволяет достигнуть этой цели. Чтобы исправить это, введем интересный трюк, использующий опцию statusline.

Многие пользователи Vim воспринимают строку состояния редактора как данность. По умолчанию statusline не имеет значения, однако можно определить ее так, чтобы в ней отображалась почти любая информация, доступная Vim. А поскольку строка состояния может отображать динамически меняющуюся информацию, такую как текущие строка и столбец, Vim пересчитывает и заново отображает statusline всякий раз, когда меняется состояние редактора. Почти каждое действие в Vim приводит к перерисовке statusline. Это можно использовать как трюк для вызова функции цветовой схемы и динамического изменения последней. Однако вскоре мы увидим, что такой подход несовершенен.

statusline принимает выражение, вычисляет его и отображает в строке состояния. Оно может содержать и функции. Используем эту особенность для вызова SetTimeOfDayColors() при каждом обновлении строки состояния, что происходит очень часто. Поскольку эта функция отменяет строку состояния, имеющуюся по умолчанию, а нам не хочется терять содержащуюся в ней ценную информацию, объединим все вместе в следующем изначальном определении строки состояния:

```
set statusline=%<%t%h%m%r\ \ %a\ %{strftime(\"%c\")}%=0x%B\
    \\ line:%l,\ \ col:%c%V\ %P
```

Определение statusline разбито на две строки. Vim рассматривает каждую строку с обратной косой чертой (\) в качестве первого непустого символа как продолжение предыдущей строки и игнорирует все пробельные символы, стоящие до обратной косой черты. Таким образом, при использовании нашего определения убедитесь, что оно скопировано и введено правильно. Если не можете заставить его работать, вернитесь к запуску скрипта с неопределенной statusline.

В документации Vim можно отыскать значение различных символов, предваряемых знаком процента. Наше определение приводит к строке состояния следующего вида:

```
ch12.xml   Wed 13 Feb 2008 06:24:25 PM EST      0x3C line:1,  col:1 Top
```

В этой главе мы уделим внимание не столько содержимому строки состояния, сколько использованию опции statusline для вычисления функции.

Теперь добавим функцию SetTimeOfDayColors() к statusline. Используя += вместо обычного знака равенства, добавим что-нибудь в конец вместо замены предыдущего определения:

```
set statusline += \ %{SetTimeOfDayColors()}
```

Сейчас наша функция является частью строки состояния. Хотя она и не выдает полезной информации в строке состояния, но проверяет время суток и потенциально может сменить цветовую схему, когда пробьет нужный час. Не видите в этом проблемы?

У нас есть функция скрипта Vim, проверяющая время суток всякий раз при обновлении строки состояния редактора. Однако в предыдущем разделе мы приложили некоторые усилия, чтобы ради повышения эффективности уменьшить количество вызовов `strftime()`. Теперь же мы увеличили число вызовов настолько, что их количество ошеломляет.

Когда во время сеанса `statusline` вычисляется в правильное время, функция делает то, что нам надо, меняя цветовую схему. Однако, согласно нашему определению, она проверяет время и устанавливает цветовую схему независимо от того, нужно ее менять или нет. В следующем разделе мы рассмотрим более эффективные способы проделать это с привлечением глобальных переменных вне функции.

## Наладка скрипта Vim с помощью глобальных переменных

Как установлено во время последней модификации нашего скрипта Vim, сейчас он ведет себя *почти* как надо. Функция вызывается всякий раз при обновлении строки состояния Vim, но из-за частых вызовов это порождает проблемы на нескольких уровнях.

Во-первых, из-за частого обращения к функции может вызвать беспокойство создаваемая ею дополнительная нагрузка на процессор компьютера. К счастью, на современных машинах это можно не принимать во внимание, но все равно переопределять цветовую схему так часто – плохой стиль. Если бы это было единственной проблемой, мы бы считали скрипт завершенным и больше не беспокоились на его счет. Однако это не так.

Если вы проверяете наши примеры на практике, то уже видите источник проблемы. Постоянная переустановка цветовой схемы при перемещении в сеансе редактирования создает заметное и раздражающее мерцание, поскольку каждое определение цветовой схемы, даже при ее совпадении с текущей, требует, чтобы Vim заново считал скрипт определения цветовой схемы, интерпретировал текст и применил правила цветовой подсветки. Даже компьютеры с высокой производительностью, скорее всего, не смогут обеспечить достаточной мощности графики, чтобы постоянное обновление не вызывало мерцания. Это нужно исправить.

Необходимо определить цветовую схему один раз, а затем в условном блоке каждый раз определять, меняется ли цветовая схема. Сделаем это с использованием глобальной переменной, устанавливаемой командой `colorscheme`: `colors_name`. Переработаем нашу функцию с учетом этого:

```
function SetTimeOfDayColors()
  " progressively check higher values... falls out on first "true"
  " (note addition of zero ... this guarantees return from function is numeric)
  let currentHour = strftime("%H")
    if currentHour < 6 + 0
    let colorScheme = "darkblue"
  elseif currentHour < 12 + 0
    let colorScheme = "morning"
```

```
    elseif currentHour < 18 + 0
      let colorScheme = "shine"
    else
      let colorScheme = "evening"
    endif

    " if our calculated value is different, call the colorscheme command.
    if g:colors_name !~ colorScheme
      echo "setting color scheme to " . colorScheme
      execute "colorscheme " . colorScheme
    endif
  endfunction
```

Похоже, это исправит проблему, но мы получим новую: появляются ошибки, показанные на рис. 12.2.

```
currentHour is 19
Error detected while processing function MyColorScheme:
line   12:
E121: Undefined variable: g:colors_name
Press ENTER or type command to continue
```

***Рис. 12.2.*** *Неопределенная переменная*

Оказывается, Vim очень строго относится к попыткам обратиться к еще не определенным переменным. Но что не так с переменной `colors_name`? Мы знаем, что ее задает команда `colorscheme`, и даже приняли меры предосторожности, использовав префикс `g:`, указывающий на глобальность этой переменной. Однако при первом выполнении функции переменная `g:colors_name` не имеет значения и еще не определена, так как не выполнялась команда `colorscheme`. Только после выполнения этой команды можно безопасно обращаться к `g:colors_name`.

Это довольно легко поправить двумя способами: в файл `.gvimrc` нужно вписать либо

```
let g:colors_name = "xyzzy"
```

либо

```
colorscheme default
```

Каждый из этих операторов определяет глобальную переменную при запуске сеанса, поэтому в рассматриваемой функции сравнение всегда будет работать. Сейчас функция динамическая и эффективная, а в следующем разделе будет сделано последнее усовершенствование.

## Массивы

Было бы здорово извлекать значение цветовой схемы, не обращаясь к огромному блоку `if...then...else`. Использование массивов улучшит наш скрипт и сделает его более наглядным.

Массив Vim можно задать, определив значение переменной как заключенный в квадратные скобки список, разделенный запятыми. Для нашей функции введем массив под названием `Favcolorschemes`. Его можно определить в самой функции, но, чтобы оставить возможность обращаться к нему из других мест сеанса, определим его за пределами функции как глобальную переменную.

```
let g:Favcolorschemes = ["darkblue", "morning", "shine", "evening"]
```

Эту строку нужно поместить в файл `.gvimrc`. После этого к любому элементу массива `g:Favcolorschemes` можно будет обратиться с помощью индекса. Индексы нумеруются с нуля, то есть `g:Favcolorschemes[2]` соответствует строке `"shine"`.

Воспользовавшись трактовкой принятых в Vim математических функций, согласно которой результат деления целых чисел — тоже целое число (остаток от деления отбрасывается), мы теперь можем быстро и легко вычислить предпочтительную цветовую схему исходя из времени суток. Взглянем на окончательную версию нашей функции:

```
function SetTimeOfDayColors()
  " currentHour will be 0, 1, 2, or 3
  let g:CurrentHour = (strftime("%H") + 0) / 6
  if g:colors_name !~ g:Favcolorschemes[g:CurrentHour]
    execute "colorscheme " . g:Favcolorschemes[g:CurrentHour]
    echo "execute " "colorscheme " . g:Favcolorschemes[g:CurrentHour]
    redraw
  endif
endfunction
```

Команда `echo` выводит информацию и «анонсирует» изменения в окружении, только что проделанные скриптом. Команда `redraw` предписывает Vim немедленно обновить экран.

Поздравляем! Вы создали законченный скрипт, приняв во внимание множество факторов, которые необходимо учитывать при составлении любого полезного скрипта.

## Динамическая конфигурация типов файлов при помощи скриптов

Рассмотрим еще один пример «стильного» скрипта. Как правило, при редактировании нового файла единственный источник, откуда Vim может определить и установить `filetype`, — это расширение файла. Например, `.c` означает, что файл является программой на C. Vim легко определяет это и загружает подходящие действия для удобного редактирования программы на C.

Однако не все файлы требуют расширения. Например, хотя скриптам оболочки уже принято назначать расширение `.sh`, автору этих строк не нравится такое соглашение, поскольку он написал тысячи скриптов за-

долго до того, как оно появилось. В действительности, Vim хорошо адаптирован к распознаванию скриптов оболочки, не нуждаясь в расширении файла, а просто основываясь на его содержимом. Однако он может их распознать только при повторном редактировании, когда файл предоставит контекст для определения типа. Скрипты Vim могут исправить эту ситуацию!

## Автокоманды

В первом примере скрипта мы полагались на привычку Vim постоянно обновлять строку состояния и «запрятали» функцию в строку состояния, чтобы устанавливать цветовую схему исходя из времени суток. Наш скрипт, динамически определяющий тип файла, будет опираться на более формальное соглашение Vim – *автокоманды* (*autocommands*).

Автокомандами могут быть любые команды, принимаемые Vim. Для их выполнения редактор использует *события*. Рассмотрим некоторые из них:

`BufNewFile`

: Запускает связанную с событием команду, когда Vim начинает редактировать новый файл.

`BufReadPre`

: Запускает связанную с событием команду *до того*, как Vim переходит на новый буфер.

`BufRead`, `BufReadPost`

: Запускает связанную с событием команду, когда Vim начинает редактировать новый буфер, но *после* считывания файла.

`BufWrite`, `BufWritePre`

: Запускает связанную с событием команду перед записью буфера в файл.

`FileType`

: Запускает связанную с событием команду после установки опции `filetype`.

`VimResized`

: Запускает связанную с событием команду при смене размера окна Vim.

`WinEnter`, `WinLeave`

: Запускает связанную с событием команду при входе в окно Vim и выходе из него соответственно.

`CursorMoved`, `CursorMovedI`

: Запускает связанную с событием команду каждый раз при перемещении курсора в *обычном* режиме и в режиме *вставки* соответственно.

Всего существует почти 80 событий Vim. Для каждого из них можно определить автоматическую команду autocmd, которая выполняется, когда событие происходит. Формат autocmd следующий:

```
autocmd [group] event pattern [nested] command
```

Элементы этого формата:

*group*

Необязательная группа команды (описана ниже).

*event*

Событие, запускающее команду *command*.

*pattern*

Шаблон, соответствующий имени файла, для которого будет выполняться *command*.

nested

Если присутствует, то позволяет этой автокоманде быть вложенной в другие.

*command*

Команда Vim, функция или пользовательский скрипт, который нужно запустить при наступлении данного события.

В нашем примере целью является распознавание типа файла для любого открываемого нами нового файла, так что в роли *pattern* выступит *.

Следующим шагом будет определение события, запускающего нужный скрипт. Поскольку необходимо, чтобы тип файла распознавался как можно раньше, наиболее подходящими выглядят два кандидата: CursorMovedI и CursorMoved.

CursorMoved срабатывает при перемещении курсора, что вряд ли можно использовать, поскольку простое перемещение курсора не добавит информации о типе файла. CursorMovedI, напротив, запускается при вводе текста, поэтому, скорее всего, подойдет.

Нужно написать функцию, которая каждый раз будет выполнять всю работу. Назовем ее CheckFileType. Теперь у нас есть все, чтобы определить команду autocmd. Она выглядит следующим образом:

```
autocmd CursorMovedI * call CheckFileType()
```

## Проверка опций

В функции CheckFileType необходимо проверить значение опции filetype. Скрипты Vim используют специальные переменные для извлечения значений опций. Имена этих переменных образуются добавлением перед именем опции (в нашем случае это filetype) символа амперсанда (&). Следовательно, в данной функции мы будем использовать переменную &filetype.

Начнем с простой версии функции CheckFileType:

```
function CheckFileType()
 if &filetype == ""
   filetype detect
 endif
 endfunction
```

Команда Vim filetype detect — это скрипт Vim, установленный в каталоге $VIMRUNTIME. Он проверяет множество критериев и пытается приписать вашему файлу определенный тип. Как правило, это происходит один раз, так что если файл новый и filetype не может определить его тип, то в сеансе редактирования не будет возможности установить никакое форматирование синтаксиса.

Проблема в следующем: функция вызывается всякий раз, когда курсор перемещается в режиме ввода, каждый раз пытаясь определить тип файла. Вначале мы проверяем, имеется ли уже у файла тип, что означало бы, что в предыдущем выполнении функция была успешной и больше делать ничего не нужно. Мы не будем беспокоиться об аномалиях, таких как ошибочное распознавание или случаи, когда в файле сначала использовался один язык программирования, а затем его поменяли на другой.

Отредактируем новый файл скрипта оболочки и посмотрим на результат:

```
$ vim ScriptWithoutSuffix
```

Введите следующее:

```
#!/bin/sh

inputFile="DailyReceipts"
```

Теперь Vim включает раскраску синтаксиса, как показано на рис. 12.3.

Из картинки можно заметить, что для строк Vim использует серый цвет, но на черно-белой картинке не видно, что #!/bin/sh синего цвета, inputFile= — черного, а "DailyReceipts" — фиолетового. К сожалению, эти цвета не соответствуют раскраске скриптов оболочки. Быстрая проверка опции filetype, сделанная командой set filetype, выдаст сообщение, показанное на рис. 12.4.

```
1 # /bin/sh
2
3 inputFile="DailyReceipts"
```

***Рис. 12.3.*** *Тип нового файла определен*

***Рис. 12.4.*** *Тип файла определен как conf*

Vim приписал файлу тип conf, а это не то, что нам нужно. Что не так?

Если вы испытывали рассматриваемый пример на практике, то могли заметить, что Vim приписывает файлу тип сразу после ввода первого символа – # – при первом событии CursorMovedI. Файлы конфигурации для утилит и демонов UNIX, как правило, используют # для комментирования строк, поэтому Vim эвристически полагает, что # в начале строки является началом комментария в конфигурационном файле. Следует добиться, чтобы Vim был более терпеливым.

Изменим функцию так, чтобы она требовала больше контекста. Вместо попыток определить тип файла при первой возможности Vim должен подождать, пока пользователь введет около 20 символов.

## Переменные буфера

В функцию необходимо ввести переменную, которая сможет дать Vim команду подождать и *не пытаться* определить тип файла, пока автокоманда CursorMovedI не вызовет функцию около 20 раз. Наше представление о новом файле и количестве символов, которые осталось в нем ввести, привязаны к конкретному буферу. То есть перемещения курсора в других буферах выполняемого сеанса работы не должны учитываться при проверке. Поэтому введем переменную буфера и назовем ее b:countCheck.

После этого мы потребуем, чтобы функция «выждала» не менее 20 перемещений курсора в режиме ввода (то есть чтобы было введено примерно 20 символов), и сделаем проверку наличия приписанного к файлу типа:

```
function CheckFileType()

let b:countCheck += 1

" Don't start detecting until approx. 20 chars.
 if &filetype == "" && b:countCheck > 20
   filetype detect
 endif
 endfunction
```

Но теперь мы получим сообщение об ошибке, показанное на рис. 12.5.

Знакомая ошибка. Как и раньше, мы имели наглость проверить значение переменой до ее определения. На этот раз это наша вина, поскольку именно данный скрипт отвечает за определение переменной b:countCheck. В следующем разделе мы покажем, как можно искусно устранить эту проблему.

```
ScriptWithoutSuffix[+]   09/15/2007 01:21:40 PM
Error detected while processing function CheckFileType:
line    1:
E121: Undefined variable: b:countCheck
Press ENTER or type command to continue
```

***Рис. 12.5.*** *b:countCheck приводит к ошибке «undefined»*

## Функция exists( )

Знание техники управления переменными и функциями очень важно: Vim требует определения каждой из них, чтобы переменная *существовала* перед обращением к ней какого-либо выражения.

Ошибку рассматриваемого скрипта легко исправить, если сделать проверку на существование переменной b:countCheck и присвоить ей значение уже известной командой :let:

```
function CheckFileType()

if exists("b:countCheck") == 0
  let b:countCheck = 0
endif

let b:countCheck += 1

 " Don't start detecting until approx. 20 chars.
 if &filetype == "" && b:countCheck > 20
   filetype detect
 endif
 endfunction
```

Теперь протестируем наш код. Рисунок 12.6 показывает момент перед достижением ограничения в 20 символов, а рис. 12.7 – что получилось после ввода 21-го символа.

Текст /bin/sh внезапно приобрел цветовую раскраску синтаксиса. Быстрая проверка с помощью set filetype подтверждает, что Vim правильно определил тип файла, как показано на рис. 12.8.

```
  1 #! /bin/sh
  2
  3 inputFil
~
```

***Рис. 12.6.*** *Тип файла еще не определен*

```
  1 #! /bin/sh
  2
  3 inputFile
~
```

***Рис. 12.7.*** *Тип файла определен*

```
ScriptWithoutSuffix[+]
  filetype=sh
```

***Рис. 12.8.*** *Определение верное*

Теперь у нас есть полное и удовлетворительное решение для всех практических целей, однако хорошим тоном будет добавить еще одну про-

верку, чтобы Vim перестал определять тип файла после ввода примерно 200 символов:

```
function CheckFileType()

if exists("b:countCheck") == 0
  let b:countCheck = 0
endif

let b:countCheck += 1

 " Don't start detecting until approx. 20 chars.
 if &filetype == "" && b:countCheck > 20 && b:countCheck < 200
   filetype detect
 endif
 endfunction
```

Сейчас, хотя функция CheckFileType вызывается при каждом перемещении курсора в Vim, «накладные расходы» незначительны, поскольку изначальная проверка приводит к выходу из функции, как только определен тип файла или превышен предел в 200 символов. Хотя это, возможно, все, что требуется для разумной функциональности при минимальных затратах на обслуживание, мы рассмотрим другие механизмы, которые обеспечат более полное и удовлетворительное решение, не только по минимуму загружающее процессор, но и действительно прекращающее работу, когда в нем больше нет надобности.

Возможно, вы заметили, что мы несколько расплывчато определяли точность порогового значения в 20 символов. Эта неопределенность была преднамеренной. Поскольку мы подсчитываем перемещения курсора, в режиме ввода разумно предположить, что каждое такое перемещение соответствует новому символу, дополняющему до «достаточного» контекста текст, по которому CheckFileType() определяет тип файла. Однако в счетчике подсчитываются *все* перемещения курсора, в том числе и удаления при исправлении ошибок. Чтобы убедиться в этом, попробуйте в рассматриваемом примере ввести символ #, удалить его клавишей BACKSPACE и проделать это 10 раз подряд. На одиннадцатый раз вы увидите, что # раскрасится, а тип файла будет (неправильно) определен как conf.

## Автокоманды и группы

Сейчас скрипт игнорирует все сторонние эффекты, возникающие при вызове нашей функции любым перемещением курсора. Мы минимизировали нагрузку на процессор путем «проверок на разумность», которые позволяют избежать необоснованного вызова тяжеловесной команды filetype detect. А что если даже минимальный код функции является затратным? Необходим способ отказа от вызова кода, когда мы в нем более не нуждаемся. Для этой цели воспользуемся понятием *групп* автокоманд и их возможностью удалять команды, основываясь на ассоциациях с группой.

Изменим наш пример, для начала связав вызываемую по событию функцию CursorMovedI с группой. Для этого Vim предоставляет команду augroup. Ее синтаксис следующий:

```
augroup groupname
```

Все последующие определения autocmd становятся связанными с группой *groupname*, пока не встретится оператор:

```
augroup END
```

(Также существует группа по умолчанию для команд, не вошедших в блок augroup.)

Свяжем предыдущую команду autocmd с собственной группой:

```
augroup newFileDetection
autocmd CursorMovedI * call CheckFileType()
augroup END
```

Теперь наша функция, включаемая по CursorMovedI, является частью группы автокоманд newFileDetection. Смысл этого будет раскрыт в следующем разделе.

## Удаление автокоманд

Для наиболее аккуратной реализации нашей функции обеспечим, чтобы она работала, только пока это необходимо. Нужно убрать ее вызов, как только функция сделала свое дело (то есть когда либо определился тип файла, либо становится понятно, что этого сделать нельзя). Vim позволяет удалять автокоманду, просто указав событие, шаблон имени файла или группу.

```
autocmd! [group] [event] [pattern]
```

Обычный символ «принуждения» в Vim – восклицательный знак (!), стоящий после ключевого слова autocmd, – указывает, что команды, связанные с этой группой, событием или шаблоном, должны быть удалены.

Поскольку ранее мы связали рассматриваемую функцию с пользовательской группой newFileDetection, у нас есть контроль над ней и возможность ее удалить, сославшись на эту группу в синтаксисе команды удаления автокоманды. Сделаем это:

```
autocmd! newFileDetection
```

При этом будут удалены все автокоманды, связанные с группой newFileDetection, которая в нашем случае содержит всего одну функцию.

Проверка того, определена ли автокоманда, а также удалена ли она, выполняется с помощью запроса при старте Vim (при создании нового файла) командой:

```
autocmd newFileDetection
```

Реакция Vim показана на рис. 12.9.

```
autocmd newFileDetection
--- Auto-Commands ---
ewFileDetection  CursorMovedI
   *         call CheckFileType()
ress ENTER or type command to continue
```

***Рис. 12.9.*** *Отклик на команду autocmd newFileDetection*

После обнаружения и присвоения типа файла *либо* после превышения порога в 200 символов нам уже не нужно определение автокоманды. Следовательно, коду необходим последний штрих. После объединения определения `augroup`, команды `autocmd` и разработанной функции строки файла `.vimrc` будут выглядеть так:

```
augroup newFileDetection
autocmd CursorMovedI * call CheckFileType()
augroup END

function CheckFileType()

  if exists("b:countCheck") == 0
    let b:countCheck = 0
  endif

  let b:countCheck += 1

  " Don't start detecting until approx. 20 chars.
  if &filetype == "" && b:countCheck > 20 && b:countCheck < 200
    filetype detect
  " If we've exceeded the count threshold (200), OR a filetype has been detected
  " delete the autocmd!
  elseif b:countCheck >= 200 || &filetype != ""
    autocmd! newFileDetection
  endif
endfunction
```

После того как появится цветовая раскраска синтаксиса, можно убедиться, что функция удалила сама себя, если ввести команду, которую мы вводили при запуске буфера:

```
autocmd newFileDetection
```

Реакция Vim показана на рис. 12.10.

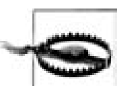
```
ScriptWithoutSuffix[+]   09/15/2007 04:1
:autocmd newFileDetection
--- Auto-Commands ---
Press ENTER or type command to continue
```

***Рис. 12.10.*** *После того как были соблюдены критерии удаления для нашей группы автокоманд*

Обратите внимание, что теперь для группы newFileDetection не определено ни одной автокоманды. Эту автогруппу можно удалить следующим образом:

```
augroup! groupname
```

Однако выполнение этой команды *не удалит* связанных с ней автокоманд, и Vim будет выдавать сообщение об ошибке всякий раз при обращении к этим автокомандам. Следовательно, перед удалением группы нужно убедиться, что связанные с ней автокоманды тоже удалены.



Не путайте удаление автокоманд с удалением автогруппы.

Поздравляем! Мы закончили второй скрипт Vim. Он расширяет наши знания о Vim и позволяет увидеть различные возможности, доступные с помощью скриптов.

# Дополнительные соображения, касающиеся скриптов Vim

Мы исследовали лишь небольшую часть огромной вселенной скриптов Vim, но надеемся, что вы почувствовали их силу. Теоретически все интерактивные действия в редакторе можно запрограммировать в скрипте.

В этом разделе мы рассмотрим прекрасный пример, который включен во встроенную документацию Vim, подробнее обсудим затронутые нами ранее концепции и познакомимся с некоторыми новыми функциями.

## Пример полезного скрипта Vim

Встроенная документация Vim содержит удобный скрипт, который, по нашему мнению, может вам пригодиться. Изначально он нацелен на отображение текущей даты в строке meta файла HTML, однако его можно использовать и для файлов многих других типов, внутри которых требуется держать время и дату его последнего изменения.

Вот почти нетронутый пример (мы внесли очень мало изменений):

```
autocmd BufWritePre,FileWritePre *.html mark s|call LastMod()|'s
fun LastMod()
  " if there are more than 20 lines, set our max to 20, otherwise, scan
  " entire file.
  if line("$") > 20
    let lastModifiedline = 20
  else
    let lastModifiedline = line("$")
  endif
  exe "1," . lastModifiedline . "g/Last modified: /s/Last modified:
```

```
        .*/Last modified: " .
    \ strftime("%Y %b %d")
endfun
```

Ниже приведен краткий анализ команды `autocmd`:

`BufWritePre`, `FileWritePre`

Cобытия, включающие выполнение команды. В нашем случае Vim выполняет автокоманду *перед* записью файла или буфера на диск.

`*.html`

Выполняет автокоманду для всех файлов, имя которых заканчивается на `.html`.

`mark s`

Здесь для удобочитаемости мы сделали изменение в оригинале. Вместо `ks` использована эквивалентная, но более понятная команда `mark s`. Она просто создает в файле метку под названием `s`, чтобы позже можно было вернуться на это место.

`|`

Вертикальная черта отделяет друг от друга команды Vim, выполняющиеся в определении автокоманды. Это обычные разделители, никак не связанные с каналами в оболочке UNIX.

`call LastMod()`

Здесь идет вызов пользовательской функции `LastMod`.

`|`

Аналогично предыдущей вертикальной черте.

`'s`

Возврат на строку с отметкой `s`.

Полезно проверить работоспособность этого скрипта, для чего нужно отредактировать файл `.html`, добавить строку «Last modified: » и вызвать команду `w`.

Этот пример хорош, но он не является канонически правильным в отношении заявленной цели делать подстановку в тег HTML `meta`. Если бы он действительно был предназначен для этого тега, то подстановка должна была бы искать в теге фрагмент `content=...`. Тем не менее этот пример является хорошей отправной точкой в решении данной проблемы и служит удачным примером для файлов других типов.

## Больше о переменных

Сейчас мы подробнее остановимся на строении переменных Vim и их использовании. В Vim есть переменные пяти типов:

*Number*

Знаковое 32-разрядное число. Число можно представлять как в десятичном, так и шестнадцатеричном (например, `0xffff`) или восьмеричном (например, `0177`) формате.

*String*

Символьная строка.

*Funcref*

Ссылка на функцию.

*List*

Это Vim-версия массивов. Представляет собой упорядоченный «список» значений и может содержать в качестве элементов любые данные Vim.

*Dictionary*

Это Vim-версия *хэша*, который часто называют также *ассоциативным массивом*. Это неупорядоченные пары значений, причем первое выступает как *ключ*, по которому можно найти связанное с ним *значение*.

Vim производит удобные преобразования переменных, когда это позволяет контекст, в том числе, что наиболее важно, преобразование числа в строку и обратно. В целях безопасности (как в нашем первом примере) при использовании строки в качестве числа надежность преобразования можно обеспечить, добавив к числу 0:

```
if strftime("%H") < 6 + 0
```

## Выражения

Редактор вычисляет выражения довольно очевидным путем. Выражение может быть простым, например число или символьная строка, а также сложным, например составное выражение, составленное из других выражений.

Важно отметить, что математические функции Vim работают только с целыми числами. Если вам нужны точность и операции с плавающей запятой, воспользуйтесь расширениями, такими как системные вызовы внешних математических процедур.

## Расширения

Vim предоставляет множество расширений и интерфейсов для других скриптовых языков, в частности для трех наиболее популярных из них – Perl, Python и Ruby. За деталями использования обратитесь к встроенной документации Vim.

## Некоторые замечания об autocmd

В более раннем примере, рассмотренном в разделе «Динамическая конфигурация типов файлов при помощи скриптов» на стр. 236, мы использовали команду Vim autocmd для обработки событий, из которых вызываются наши пользовательские функции. Это очень хорошо, но не отменяет более простого использования команды autocmd. Например, ее можно применять для настройки специальных опций Vim для различных типов файлов.

Неплохим примером может служить смена опции shiftwidth для файлов различных типов. Файлы, тип которых предполагает многочисленные отступы и вложенные уровни, могут выиграть от уменьшения отступа. Возможно, для HTML вам захочется установить shiftwidth равным 2, чтобы предотвратить «уход» кода направо за пределы экрана, а для программ на C – shiftwidth, равный 4. Чтобы выполнить это разделение, внесите в файлы .vimrc или .gvimrc следующие строки:

```
autocmd BufRead,BufNewFile *.html set shiftwidth=2
autocmd BufRead,BufNewFile *.c,*.h set shiftwidth=4
```

## Внутренние функции

Кроме команд Vim у вас есть доступ примерно к 200 встроенным функциям. Их перечисление и документирование выходит за рамки нашей темы, однако полезно знать, функции каких типов и категорий доступны пользователю. Следующие категории взяты из файла usr_41.txt встроенной справки Vim:

*Управление строками*

Сюда включены все стандартные функции работы со строками, знакомые каждому программисту, от преобразования строк до процедур работы с подстроками и многие другие.

*Функции работы со списками*

Это целый «массив функций работы с массивами». Здесь хорошо отражены функции работы с массивами, присутствующие в Perl.

*Функции работы со словарями (ассоциативными массивами)*

Здесь содержатся функции, включающие извлечение, обработку и проверку, а также функции других типов. Опять-таки они сильно напоминают аналоги из Perl.

*Функции работы с переменными*

Являются «получателями» (getter) и «установщиками» (setter), передающими переменные между окнами и буферами Vim. Сюда также входит функция type, определяющая тип переменной.

*Функции работы с курсором и позицией*

Позволяют перемещаться по файлам и буферам, а также создавать отметки для запоминания позиций и возвращения к ним. Кроме то-

го, здесь содержатся функции, выдающие позиционную информацию (то есть текущие строку и столбец).

*Функции для работы с текстом в текущем буфере*

Управляют текстом в буферах, например меняют строку, запрашивают строку и т. д. Также сюда входят функции поиска.

*Функции управления файлами и системой*

Эта категория содержит функции для навигации по операционной системе, в которой выполняется Vim. Например, здесь есть функции поиска файлов по путям, определения текущего рабочего каталога, создания, удаления файлов и прочие. В этой группе также есть функция `system()`, которая передает команды операционной системе для внешнего выполнения.

*Функции даты и времени*

Выполняют широкий спектр манипуляций с форматами даты и времени.

*Функции для буферов, окон и списка аргументов*

Предоставляют механизм для сбора информации о буферах и аргументах каждого из них.

*Функции командной строки*

Возвращают положение командной строки, саму командную строку и тип командной строки, а также устанавливают курсор в командной строке.

*Функции Quick Fix и списка местоположений*

Функции возвращают и меняют списки, обеспечивающие работу функции `quickfix` в Vim.

*Функции завершения в режиме вставки*

Используются при автоматическом завершении команд и в режиме вставки.

*Функции сворачивания*

Дают информацию о свертках и разворачивают текст из закрытых сверток.

*Функции синтаксиса и подсветки*

Возвращают информацию о группах расцветки синтаксиса и об идентификаторах синтаксиса.

*Функции проверки орфографии*

Находят слова с возможными ошибками и предлагают правильное написание.

*Функции истории*

Получают, добавляют и удаляют записи в истории.

*Интерактивные функции*

Предоставляют пользователю интерфейс для таких действий, как выбор файла.

*Функции GUI*

Здесь содержатся три простые функции, возвращающие текущий шрифт, координаты *x* и *y* графического окна.

*Функции сервера Vim*

Служат для связи с (возможно) удаленным сервером Vim.

*Функции размера и положения окна*

Эти функции возвращают информацию об окне и позволяют сохранять и восстанавливать «виды» окон.

*Прочие функции*

Сюда попадают различные «прочие» функции, не вошедшие в вышеприведенные категории, например `exists`, проверяющая существование элемента в Vim, и `has`, проверяющая, поддерживает ли редактор определенную функцию.

# Ресурсы

Надеемся, мы пробудили достаточный интерес и предоставили достаточно информации, чтобы вы сами смогли написать скрипты Vim. Написанию скриптов можно было бы посвятить всю книгу. К счастью, существуют другие ресурсы, к которым можно обратиться за помощью.

Хорошей отправной точкой является сама домашняя страница программы Vim. Зайдите на нее и перейдите на страницы, посвященные скриптам: *http://www.vim.org/scripts/index.php*. Здесь вы найдете более 2000 доступных для скачивания скриптов. По всему содержимому можно проводить поиск. Также вас пригласят поучаствовать в ранжировании скриптов и даже опубликовать свой собственный.

Напоминаем, что сложно переоценить встроенную справку Vim. Рекомендуем следующие, самые продуктивные ее разделы:

```
help autocmd
help scripts
help variables
help functions
help usr_41.txt
```

Не забывайте о той тьме скриптов Vim, которые содержатся в каталогах среды времени выполнения редактора. Все файлы, заканчивающиеся на `.vim`, являются скриптами – примерами, на основе которых можно учиться программировать.

Экспериментируйте. Это лучший вид обучения.

# 13

# Графический Vim (gvim)

Довольно долго к vi и его модификациям предъявлялась претензия на отсутствие графического пользовательского интерфейса (GUI). Если вы застали религиозные войны между сторонниками Emacs и vi, то помните, что отсутствие GUI было главным козырным аргументом в пользу того, что vi – неудачник.

В конце концов разновидности и подобия vi обзавелись собственными графическими версиями. Графический Vim называется gvim. Как и другие модификации vi, gvim предоставляет надежные и расширяемые графические функции и возможности. В этой главе мы рассмотрим лишь самые полезные из них.

Одни графические функции gvim служат обертками для базовых возможностей Vim, а другие реализуют парадигму «наведи и щелкни», к которой уже привыкла большая часть пользователей компьютеров. Хотя сердца некоторых пользователей со стажем (в том числе авторов этих строк!) холодеют от мысли о присутствии GUI в их рабочем редакторе, gvim задуман и реализован как следует. Он предоставляет функциональность и возможности, обычно предлагаемые другими программами, таким образом делая кривую обучения более пологой для новичков и ненавязчиво предоставляя опытным пользователям дополнительные мощные средства редактирования. Это выглядит как хороший компромисс.

gvim для MS Windows поставляется с программой, в меню называемой «easy gvim» (упрощенный gvim). Возможно, это ценно для тех, кто раньше не сталкивался с Vim, но для опытных пользователей это что угодно, *только не* «easy».

В этой главе мы сначала представим общие концепции и графические функции, а также вводный раздел, касающийся взаимодействия с мы-

шью. Вдобавок мы детально обсудим отличия gvim в различных окружениях и представим необходимые сведения об этом. Основное внимание будет уделяться двум главным графическим платформам: MS Windows и X Window System. Другие платформы мы рассмотрим очень бегло, но дадим ссылки на подходящие ресурсы, где можно почерпнуть больше информации. Также вы получите краткий список опций GUI с описанием.

# Общее введение в gvim

gvim дает те же функциональность, силу и возможности, что и Vim, дополняя их удобством и интуитивностью графического окружения. gvim предоставляет графические возможности, привычные любому современному пользователю, от традиционных меню до редактирования с визуальной подсветкой. Тем не менее ветеранам, привыкшим к консольному, текстовому vi, gvim предоставляет знакомую базовую систему и не ломает парадигму, обеспечившую vi репутацию мощного редактора.

## Запуск gvim

Если Vim компилировался с поддержкой GUI, то графический интерфейс можно вызвать командой gvim или Vim с дополнительной опцией -g. В Windows самоустанавливающийся исполняемый файл добавляет интересную функцию, которую многие обнаруживают уже после установки: в Windows Explorer появляется новый пункт под названием «Edit with Vim». Это дает быстрый и удобный доступ к gvim, интегрированный в окружение Windows. Стоит испробовать его на тех файлах, о которых вы раньше не думали, например на файлах .exe. Однако редактирование двоичных файлов *небезопасно*, поэтому предупреждаем: делайте это с особой осторожностью.

Требуемые для gvim конфигурационные файлы и опции немного отличаются от таковых для Vim. gvim при запуске считывает и выполняет два файла: сначала .vimrc, а затем .gvimrc. Хотя уникальные для gvim опции и определения можно прописывать в .vimrc, лучше делать это в файле .gvimrc. Это разделит настройки для gvim и стандартного Vim. Например, :set columns=100 не будет работать в Vim, а выдаст сообщение об ошибке при запуске редактора.

Если существует системный gvimrc (обычно это $VIM/gvimrc), то он тоже выполняется. Этот файл настройки, распространяющийся на всю систему, может использоваться администратором для задания опций, общих для всех пользователей. Он создаст базовую конфигурацию, после чего пользователи получат общие настройки редактирования.

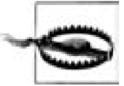
Более опытные пользователи Vim могут добавить собственные любимые пользовательские настройки и функции. После считывания необязательной системной конфигурации gvim ищет дополнительную информацию о настройке в следующих местах (в приведенном ниже порядке) и, отыскав один из этих пунктов, останавливает поиск:

- Команда exrc, которая хранится в переменной окружения $GVIMINIT.
- Пользовательский файл gvimrc, обычно содержащийся в $HOME/.gvimrc. Если он найден, то он *подключается* (*sourced*).
- Если $HOME не установлен, то в окружении Windows gvim ищет в $VIM/_gvimrc. (Это обычная ситуация для пользователя Windows, однако она сильно отличается там, где установлены аналоги UNIX и переменная $HOME. Популярным примером служит комплект программ UNIX Cygwin.)
- Наконец, если _gvimrc не обнаружен, то gvim заново ищет .gvimrc. Если программа выясняет, что есть непустой файл для выполнения, то имя этого файла запоминается в переменной $MYGVIMRC и дальнейшая инициализация останавливается.

Есть еще один вариант настройки. Если в течение вышеописанных этапов инициализации установлена опция exrc:

```
set exrc
```

gvim производит дополнительный поиск файлов .gvimrc, .exrc или .vimrc в текущем каталоге и *подключает* найденный файл, если тот не был одним из вышеперечисленных (то есть если он еще не был найден при инициализации и не был выполнен).



В окружении UNIX локальные каталоги, содержащие конфигурационные файлы (такие как .gvimrc и .vimrc), могут приводить к проблемам безопасности. По умолчанию в gvim установлена опция secure, которая ставит ограничение на выполнение определенных действий этими файлами, если пользователь не является их владельцем. Если вы хотите быть уверенным, установите эту опцию в файле .vimrc или .gvimrc.

## Использование мыши

Мышь в gvim выполняет полезные функции в любом режиме редактора. Рассмотрим стандартные режимы Vim и узнаем, как gvim обращается с мышью в каждом из них:

*Командный режим*

Вы входите в этот режим, когда нажатием двоеточия (:) открываете расположенный внизу окна командный буфер. Если это окно находится в командном режиме, то можно использовать мышь для смены позиции курсора в любом месте командной строки. Этот режим включен по умолчанию или при использовании флага c в опции mouse.

*Режим вставки*

Служит для ввода текста. Если вы щелкаете по буферу, который находится в режиме вставки, то мышь меняет позицию курсора и позволяет сразу начать ввод текста с этого места. Этот режим включен по умолчанию либо при использовании флага i в опции mouse.

Поведение мыши в режиме вставки дает простое и интуитивное позиционирование в стиле «наведи и щелкни». В частности, это устраняет необходимость выходить из режима вставки, перемещаться мышью, командами перемещения или другими методами, после чего снова входить в режим редактирования.

На первый взгляд это кажется хорошей идеей, но на практике применяется только ограниченным числом пользователей Vim. Опытных пользователей это скорее будет раздражать, нежели помогать им.

Только представьте, что произойдет, если при нахождении в режиме вставки переключиться из `gvim` в другое приложение. При щелчке для возврата в окно `gvim` точка, на которую пришелся ваш щелчок, становится точкой вставки текста. Скорее всего, это место не совпадает с нужным. В однооконном сеансе `gvim` можно перейти на первоначальное место; в многооконном режиме указатель мыши может попасть совсем в другое окно. Текст может быть введен совсем в другом файле!

*Нормальный режим*

Сюда относится ситуация, когда вы не находитесь ни в командном режиме, ни в режиме вставки. Щелчок по экрану просто оставляет курсор на том символе, по которому вы щелкнули. Этот режим включен по умолчанию или при использовании флага `n` в опции `mouse`.

Нормальный режим обеспечивает легкий и простой способ смены позиции курсора, однако в нем реализована несколько странная поддержка перемещения за пределы верхнего или нижнего краев видимого экрана. Щелкните и удерживайте кнопку мыши, после чего переместитесь к верху или к низу экрана. `gvim` сделает прокрутку вверх и вниз соответственно. Если прокрутка остановилась, подвигайте мышью влево-вправо (неясно, почему редактор ведет себя именно так).

Другой недостаток нормального режима заключается в том, что пользователи, особенно новички, могут начать пользоваться исключительно способом «указал и щелкнул». Это может удержать их от изучения команд перемещения и, следовательно, методов редактирования в Vim. Наконец, это может вызвать те же трудности, что и в режиме вставки.

Кроме того, в `gvim` есть *визуальный* режим, также известный как режим *выделения*. Он включен по умолчанию либо при использовании флага `v` в опции `mouse`. Визуальный режим является самым гибким, так как он позволяет выделять текст перетаскиванием мыши, при этом подсвечивая выделение. Им можно пользоваться в сочетании с командным режимом, а также режимами вставки и нормальным.

В опции `mouse` можно указать любое сочетание флагов. Синтаксис использования проиллюстрирован следующими командами:

`:set mouse=""`

Выключает все действия мыши.

`:set mouse=a`

Включает все действия мыши (по умолчанию).

`:set mouse+=v`

Включает визуальный режим (`v`). В нашем примере используется синтаксис `+=`, чтобы добавить флаг к текущему значению опции `mouse`.

`:set mouse-=c`

Выключает действия мыши в командном режиме (`c`). В этом примере используется синтаксис `-=`, чтобы удалить флаг из текущего значения опции `mouse`.

Возможно, новички предпочтут включить все по максимуму, тогда как эксперты (такие как автор этих строк) могут вообще выключить мышь.

Если вы ее используете, мы рекомендуем выбрать ее привычное поведение с применением команды `gvim :behave`, которая в качестве аргументов использует `mswin` или `xterm`. Как можно понять из названия аргументов, `mswin` установит опции для достижения сходства с поведением в Windows, а `xterm` – в стиле X Window System.

В Vim есть несколько других опций, таких как `mousefocus`, `mousehide`, `mousemodel` и `selectmode`. Чтобы узнать о них больше, обратитесь к встроенной документации Vim.

Если в вашей мыши есть колесо прокрутки, `gvim` будет поддерживать его по умолчанию, предсказуемо прокручивая окно вверх и вниз независимо от значения опции `mouse`.

## Полезные меню

`gvim` вносит в графическое окружение один хороший штрих, упрощающий отдельные, особо таинственные команды Vim – действия, доступные из меню. Стоит упомянуть о двух из них.

### Меню Windows в gvim

Меню *Windows* в `gvim` содержит множество самых полезных и популярных команд Vim по управлению окнами, к которым относятся команды, разделяющие отдельное графическое окно на несколько областей. Это меню можно «оторвать», как показано на рис. 13.1, чтобы было удобно открывать окна и перемещаться между ними. Результат показан на рис. 13.2.

### Меню в gvim, выпадающее по правой кнопке мыши

Меню, выпадающее при щелчке правой кнопки мыши в буфере редактирования `gvim`, показано на рис. 13.3.

Если есть выделенный (подсвеченный) текст, то выпадает другое меню, показанное на рис. 13.4.

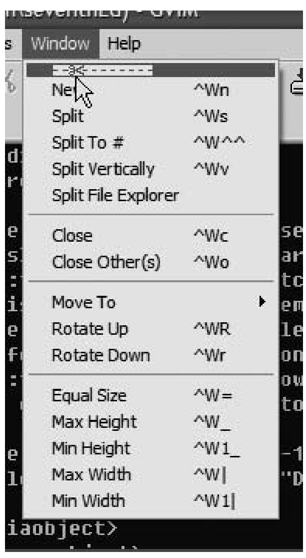



***Рис. 13.1.*** *Меню Windows в gvim*

***Рис. 13.2.*** *Меню Windows в gvim, оторванное и плавающее*

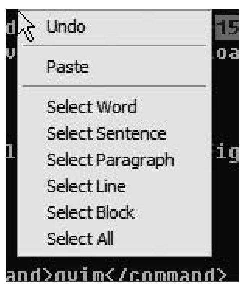



***Рис. 13.3.*** *Общее меню редактирования gvim*

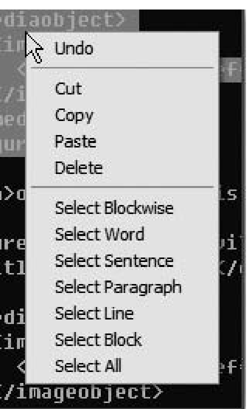



***Рис. 13.4.*** *Меню редактирования gvim при выделенном тексте*

Обратите внимание, что на рис. 13.3 меню переместилось и находится над совершенно другим приложением. Это прекрасный способ держать часто используемое меню под рукой, но в стороне от редактирования. Оба они удобны для операций выделения, вырезания, копирования, удаления и вставки. Пользователи других графических редакторов

часто используют такие окна, но они полезны даже для опытных пользователей Vim. Особенно хороша их способность взаимодействовать с буфером обмена Windows предсказуемым образом.

# Настройка полос прокрутки, меню и панелей инструментов

В gvim присутствуют обычные виджеты GUI, такие как полосы прокрутки, меню и панели инструментов. Как и в большинстве современных графических приложений, эти виджеты можно настраивать.

По умолчанию окно gvim содержит несколько меню и панель инструментов наверху (рис. 13.5).

***Рис. 13.5.*** *Верхняя часть окна gvim*

## Полосы прокрутки

Полосы прокрутки, позволяющие быстро перемещаться по файлу вверх и вниз либо влево и вправо, не являются обязательными. Их можно отображать или скрывать с помощью опции guioptions, как описано в конце этой главы в разделе «Опции GUI и обзор команд» на стр. 269.

Поскольку стандартным действием в Vim является показ всего текста в файле (с переносом строк в окне при необходимости), следует заметить, что в настроенном обычным образом сеансе gvim горизонтальная полоса прокрутки совершенно бесполезна.

Левая и правая полосы прокрутки включаются/выключаются путем включения/выключения флагов r или l в опции guioptions. l гарантирует, что левая полоса всегда видима, а r – что всегда видна правая полоса прокрутки. Прописные варианты L и R указывают редактору показывать левую или правую полосы, только когда окно разделено вертикально.

Горизонтальная полоса прокрутки управляется включением/исключением b из опции guioptions.

Ну и, конечно, *можно* прокручивать правую и левую полосы одновременно! Точнее, прокрутка любой из них приводит к тому, что и другая перемещается в том же направлении. Держать полосы прокрутки на обеих сторонах окна очень удобно. В зависимости от того, где находится указатель мыши, вы просто щелкаете и перетаскиваете ближайшую полосу прокрутки.

Многие опции, включая `guioptions`, задают много параметров, поэтому по умолчанию они могут включать множество флагов. Будущие версии `gvim` могут добавить свои, новые флаги. Следовательно, в команде `:set guioptions` важно использовать синтаксис `+=` и `-=`, дабы избежать удаления нужных установок. Например, `:set guioptions+=l` добавит в `gvim` опцию «полоса прокрутки всегда слева» и оставит остальные компоненты строки `guioptions` нетронутыми.

## Меню

Меню `gvim` полностью настраиваемое. В этом разделе мы рассмотрим характеристики меню по умолчанию, показанного на рис. 13.5, и покажем способ управления его форматом.

На рис. 13.6 показан пример использования меню. Здесь мы выбрали пункт Global Settings из меню Edit.

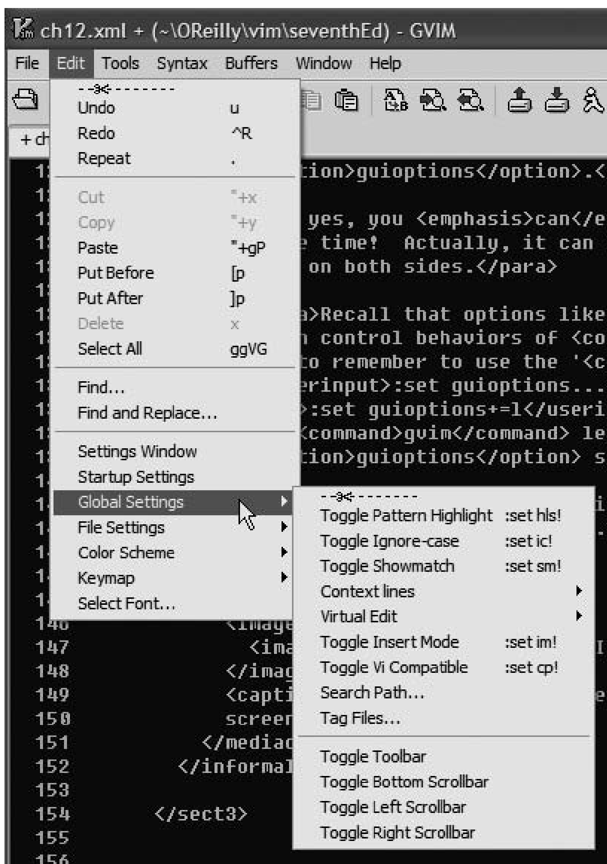



*Рис. 13.6. Расположение меню Edit каскадом*

Интересно заметить, что эти опции меню служат просто обертками для команд Vim. Фактически вы сами можете создавать и настраивать пункты меню. Вскоре мы расскажем об этом.

Заметим, что если обращать внимание на меню, содержащее справа сочетания клавиш и команды, то можно со временем выучить команды Vim. Например, из рис. 13.6 видно, что хоть для новичка удобно найти знакомую команду отмены Undo в меню Edit – там же, где она находится и в других популярных программах, – но гораздо *быстрее* будет использовать клавишу u, указанную в меню.

Как видно из рис. 13.6, вверху каждого меню находится пунктирная линии, содержащая пиктограмму ножниц. Щелчок по этой линии «отрезает» меню, после чего появляется свободно плавающее окно, в котором доступны все пункты подменю без обращения к строке меню. Если щелкнуть по пунктирной линии над пунктом меню Toggle Pattern Highlight, показанном на рис. 13.6, то вы увидите что-то наподобие рис. 13.7. Плавающее окно можно поместить в любое место рабочего стола.

Все команды этого подменю становятся мгновенно доступными по одному щелчку мышью по окну этого подменю. Каждому пункту меню назначена клавиша. Если пункт меню является подменю, то он представлен как кнопка со знаком «больше» (который выглядят как стрелка вправо) в правой части кнопки. Нажатие этой кнопки раскрывает подменю.

***Рис. 13.7.*** *Отрывание меню*

## Основная настройка меню

gvim сохраняет определения меню в файле с именем $VIMRUNTIME/menu.vim.

Определение пунктов меню очень похоже на отображение клавиш. Как было видно в разделе «Использование команды отображения map» на стр. 126, клавиша может отображаться как:

```
:map <F12> :set syntax=html<CR>
```

Меню организуется сходным образом.

Предположим, что вместо назначения установки синтаксиса html клавише F12 нам для этой задачи нужен специальный пункт «HTML» в меню *File*. Воспользуемся командой :amenu:

```
:amenu File.HTML :set syntax=html<CR>
```

Четыре символа <CR> следует вводить как показано, так как это часть команды.

Теперь посмотрим на меню *File*. Мы увидим новый пункт *HTML*, как показано на рис. 13.8. Использование amenu вместо menu гарантирует, что этот пункт будет доступен во всех режимах (командном, режиме вставки и нормальном режиме).

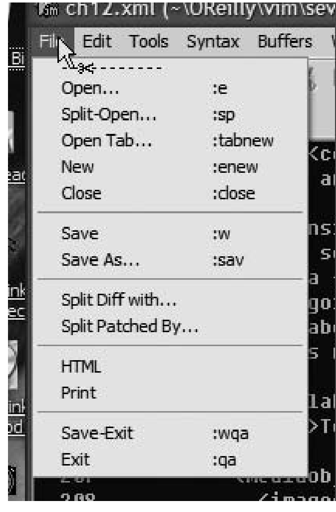
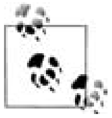



***Рис. 13.8.*** *Пункт меню HTML в меню File*



Команда menu добавляет пункт в меню только для командного режима. В нормальном режиме и режиме вставки он доступен не будет.

Расположение пункта меню задается рядом его многоуровневых элементов, разделенных точками (.). В нашем примере `File.HTML` добавляет пункт «HTML» в меню *File*. Последний элемент в этом ряду – тот, который вы хотите добавить. Мы добавляли элементы к существующему меню, но вскоре будет видно, что создание целого каскадного ряда новых меню ничуть не сложнее.

Проверим работоспособность нового пункта меню. Допустим, мы начали редактировать файл, распознанный Vim как файл XML, как можно узнать из строки состояния на рис. 13.9. Мы настроили ее так, чтобы Vim и `gvim` отображали текущий синтаксис в ее самой правой части (см. подраздел «Интересный трюк» на стр. 232).

После вызова нового пункта меню *HTML* строка состояния Vim подтверждает, что этот пункт сработал, так что теперь установлен синтаксис HTML (рис. 13.10).

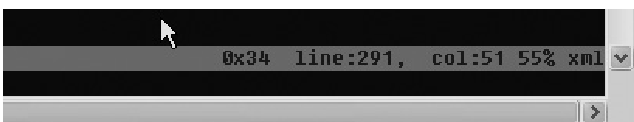



***Рис. 13.9.*** *Строка состояния, показывающая установленный синтаксис XML перед действием нового пункта меню*

***Рис. 13.10.*** *Строка состояния, показывающая синтаксис HTML после действия нового пункта меню*

Обратите внимание, что добавленный пункт меню *HTML* не имеет сокращения или команды справа от него. Сейчас мы переделаем это так, чтобы этот пункт содержал подобное дополнение.

Во-первых, удалите существующий пункт:

```
:aunmenu File.HTML
```

Если вы добавляли пункт для командного режима, то есть пользовались командой `menu`, то его можно удалить с помощью `unmenu`.

После этого добавим новый пункт меню *HTML*, отображающий команду, связанную с этим пунктом:

```
:amenu File.HTML<TAB>syntax=html<CR> :set syntax=html<CR>
```

Теперь, после спецификации пункта меню, следует <TAB> (набирать надо буквально это) и syntax=html<CR>. Вообще говоря, чтобы отобразить текст справа от пункта меню, поместите его после строки <TAB> и закончите ввод сочетанием <CR>. Полученное в результате меню *File* показано на рис. 13.11.

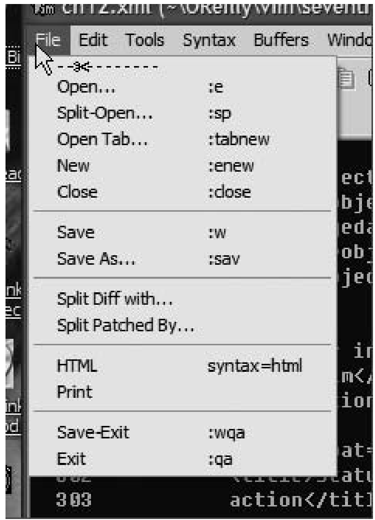



***Рис. 13.11.*** *Пункт меню HTML, в котором отображена соответствующая команда*

Если вам нужны пробелы в тексте описания пункта меню (или в названии меню), их нужно предварять обратной косой чертой (\), иначе Vim воспримет все, что стоит после первого пробела, как определение действия меню. В предыдущем примере, если бы мы захотели в качестве текста описания иметь :set syntax=html вместо простого syntax=html, то команда :amenu имела бы следующий вид:

```
:amenu File.HTML<TAB>set\ syntax=html<CR> :set syntax=html<CR>
```

В большинстве случаев лучше не менять определения имеющихся пунктов меню, а вместо этого создавать отдельные независимые элементы. Это требует определения нового меню в корневом уровне, но подобное действие не сложнее, чем добавление пункта к существующему меню. Продолжая наш пример, создадим новое меню под названием MyMenu и добавим в него пункт HTML. Для начала удалим пункт HTML из меню *File*:

```
:aunmenu File.HTML
```

Затем введем команду:

```
:amenu MyMenu.HTML<TAB>syntax=html :set syntax=html<CR>
```

Рисунок 13.12 показывает, как будет выглядеть строка меню после выполнения этой команды.

*Рис. 13.12. Строка меню с добавленным пунктом MyMenu*

Команды меню предоставляют искусный контроль над расположением меню и их поведением, например они следят за тем, указывают ли команды на действие. Можно также управлять видимостью меню. В следующем разделе мы рассмотрим эти возможности.

## Дальнейшая настройка меню

Теперь, когда мы знаем способы простого изменения и расширения меню `gvim`, рассмотрим другие примеры настройки и управления.

В предыдущем примере мы не указали, где именно нужно разместить новый пункт `MyMenu`, поэтому `gvim` сам поместил его между пунктами Window и Help. `gvim` позволяет задавать позицию с помощью такого понятия, как *приоритет*, который является просто числом, присвоенным каждому пункту меню и определяющим занимаемое этим пунктом место в строке меню. К сожалению, многие пользователи рассматривают приоритет совершенно не так, как определено в `gvim`. Чтобы понять его суть, взгляните снова на порядок следования пунктов меню на рис. 13.5 и сравните с приоритетом пунктов меню `gvim`, установленным по умолчанию и приведенным в табл. 13.1.

*Таблица 13.1. Приоритеты меню, стоящие в gvim по умолчанию*

| Меню | Приоритет |
|---|---|
| File | 10 |
| Edit | 20 |
| Tools | 40 |
| Syntax | 50 |
| Buffers | 60 |
| Window | 70 |
| Help | 9999 |

Многие пользователи считают, что File имеет более высокий приоритет, чем Help (из-за чего File расположен левее Help), но на самом деле приоритет Help выше. Так что рассматривайте значение приоритета как указание, насколько правее будет расположен этот пункт меню.

Приоритет меню можно определить, если приписать его числовое значение к команде `menu`. Если значение не установлено, приоритету присваивается значение, равное 500. Именно поэтому *MyMenu* в прошлом примере оказалось между *Window* (приоритет 70) и *Help* (приоритет 9999).

Предположим, что нужно поместить новый пункт меню между File и Edit. `MyMenu` следует присвоить приоритет больше 10, но меньше 20. Следующая команда установит значение приоритета равным 15, после чего будет достигнут нужный эффект:

```
:15amenu MyMenu.HTML<TAB>syntax=html :set syntax=html<CR>
```

Коль скоро меню появилось, его положение зафиксировано для всего сеанса редактирования и не меняется в результате действия других команд, затрагивающих это меню. Например, если вы добавите новый пункт, приписав к команде префикс, устанавливающий новый приоритет, то положение меню от этого не изменится.

Чтобы внести еще больше неразберихи в приоритеты и позиции меню, добавим, что положение пунктов меню *внутри* меню тоже можно задавать с помощью приоритета. Пункты меню с более высоким приоритетом располагаются ниже, чем пункты с меньшим приоритетом, но синтаксис для пунктов меню отличается от используемого при определении места меню.

Расширим один из наших старых примеров, присвоив очень большой приоритет (9999) для пункта меню *HTML*, чтобы он появился в самом низу меню File:

```
:amenu File.HTML .9999 <TAB>syntax=html<CR> :set syntax=html<CR>
```

Зачем нужна точка перед 9999? Здесь нужно указать два приоритета, отделенных друг от друга точками: один – для *File*, а другой – для *HTML*. Приоритет для *File* мы оставим пустым, поскольку это меню уже существует и его приоритет нельзя изменить.

Вообще говоря, приоритеты для пунктов меню расположены между названием меню для этого пункта и определением пункта. Для каждого уровня в иерархии меню следует указать приоритет или поставить точку, указывая этим, что он остается без изменения. Таким образом, если вы добавляете пункт, расположенный глубоко в иерархии меню, например Edit→Global Settings→Context lines→Display, и хотите присвоить последнему пункту (Display) приоритет 30, то его нужно указать как `...30`. Размещение с приоритетом выглядит так:

```
Edit.Global\ Settings.Context\ lines.Display ...30
```

Как и в случае с приоритетами меню, приоритеты пунктов меню становятся фиксированными после их задания.

Наконец, вы можете управлять «промежутками» в меню с помощью разделителей меню в `gvim`. Используйте те же определения, что и для

добавления обычного пункта меню, но вместо названия команды «...» поставьте минус (-) до и после него[1].

## Соберем все вместе

Теперь мы знаем, как создавать, помещать и настраивать меню. Сделаем наш пример постоянной частью окружения gvim, поместив рассмотренные команды в файл .gvimrc. Последовательность строк должна выглядеть примерно так:

```
" add HTML menu between File and Edit menus
❶15amenu MyMenu.XML<TAB>syntax=xml :set syntax=xml<CR>
❷amenu ❸.600 MyMenu.-Sep- :
❹amenu ❺.650 MyMenu.HTML<TAB>syntax=html :set syntax=html<CR>
❻amenu ❼.700 MyMenu.XHTML<TAB>syntax=xhtml :set syntax=xhtml<CR>
```

Теперь у нас есть высококлассное собственное меню, дающее быстрый доступ к трем часто используемым командам. В этом примере нужно отметить несколько важных моментов:

- Первая команда (❶) использует префикс 15, предлагая gvim применять приоритет, равный 15. Для ненастроенного окружения этот префикс поместит меню между меню *File* и *Edit*.
- Последующие команды (❷, ❹ и ❻) *не определяют* приоритет, поскольку будучи единожды заданным, он не может принимать другие значения.
- После первой команды мы использовали синтаксис приоритета подменю (❸, ❺ и ❼), чтобы удостовериться в правильном порядке каждого нового пункта. Обратите внимание, что первое определение начинается с .600. Это гарантирует, что пункт меню помещается после первого определенного нами пункта, так как мы не задавали *его* приоритет, а присвоили значение по умолчанию, равное 500.

Для более удобного доступа щелкните по линии «с ножницами», чтобы появилось собственное плавающее меню, как показано на рис. 13.13.

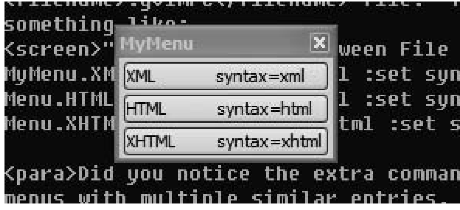



***Рис. 13.13.*** *Пользовательское плавающее «оторванное» меню*

[1] Пример из встроенной справки Vim: :menu Example.-Sep- :. – *Прим. науч. ред.*

## Панели инструментов

Панели инструментов – это длинные полосы значков для быстрого доступа к функциям программы. Например, в Windows панель инструментов `gvim` расположена вверху окна.

В табл. 13.2 показаны значки панели инструментов и их значения.

*Таблица 13.2. Значки панели инструментов gvim и их значения*

| Значок | Описание | Значок | Описание |
|---|---|---|---|
| | Диалог открытия файла | | Найти следующее вхождение шаблона поиска |
| | Сохранить текущий файл | | Найти предыдущее вхождение шаблона поиска |
| | Сохранить все файлы | | Загрузить ранее сохраненный сеанс |
| | Распечатать буфер | | Сохранить текущий сеанс редактирования |
| | Отменить последнюю команду | | Выбрать скрипт Vim для запуска |
| | Заново выполнить последнее действие | | Собрать текущий проект командой `make` |
| | Вырезать выделенную часть текста в буфер обмена | | Сгенерировать теги для текущего дерева каталогов |
| | Скопировать выделение в буфер обмена | | Перейти на тег под курсором |
| | Вставить буфер обмена в буфер | ? | Открыть справку |
| | Найти и заменить | | Поиск в справке |

Если эти значки вам не знакомы или не понятны, можно отобразить в панели инструментов как значки, так и текст. Вызовите команду:

```
:set toolbar="text,icons"
```

Как принято во многих продвинутых функциях, Vim требует наличия включенной панели инструментов на этапе компиляции, чтобы в случае отсутствия необходимости ее можно было выключить, сэкономив память. Панель инструментов не появится, пока в `gvim` не будет включена одна из следующих опций: `+GUI_GTK`, `+GUI_Athena`, `+GUI_Motif` или `+GUI_Photon`. В главе 9 рассказано, как перекомпилировать Vim и в результате создать ссылку на исполняемый файл `gvim`.

Изменение панели инструментов очень похоже на изменение меню. В сущности, для этого используется та же самая команда `:menu`, но с расширенным синтаксисом для определения графики. Хотя есть алго-

ритм, помогающий gvim находить связанный с каждой командой значок, мы рекомендуем указать его явно.

gvim рассматривает панель инструментов как одномерное меню. Кроме того, аналогично управлению положением нового меню относительно имеющихся пунктов, можно управлять положением кнопок на панели инструментов, приписывая к команде menu число, определяющее приоритет позиции. В отличие от меню, новую панель инструментов создать нельзя. Все определения новых пунктов панели инструментов появляются на одной панели. Синтаксис добавления новой кнопки на панель инструментов следующий:

```
:amenu icon=/some/icon/image.bmp ToolBar.NewToolBarSelection Action
```

где */some/icon/image.bmp* – путь к файлу, содержащему кнопку панели или рисунок (обычно это значок), который будет отображаться на панели, *NewToolBarSelection* – новый пункт для кнопки панели инструментов, а *Action* задает действие, выполняемое при нажатии кнопки.

Например, определим новый элемент панели инструментов, который при нажатии или выделении выводит окно DOS в Windows. Полагая, что переменная PATH в Windows задана правильно (так и должно быть), определим элемент для запуска окна DOS из gvim, выполняя следующую команду (т. е. то, что обозначено как *Action*):

```
:!cmd
```

Для кнопки нового элемента или изображения мы используем значок, содержащийся в нашей системе в $HOME/dos.bmp и показывающий приглашение командной строки DOS, как видно на рис. 13.14.

***Рис. 13.14.*** *Значок DOS*

Выполните команду:

```
:amenu icon="c:$HOME/dos.bmp" ToolBar.DOSWindow :!cmd<CR>
```

Она создаст пункт панели инструментов и добавит наш значок в ее конец. Панель инструментов должна приобрести вид, как на рис. 13.15. Новый значок появится крайним справа.

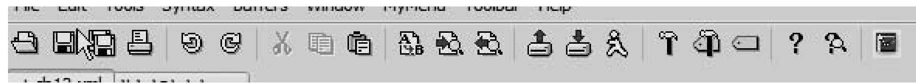


***Рис. 13.15.*** *Панель с добавленной командой DOS*

## Всплывающие подсказки

gvim позволяет определить всплывающие подсказки как для пунктов меню, так и для значков панели инструментов. Подсказки для пунктов меню появляются в области командной строки редактора при выделении пункта меню мышью. Всплывающие подсказки для панели инструментов отображаются при нахождении указателя мыши над значком панели. Например, на рис. 13.16 показано, как подсказка всплывает при наведении мыши на кнопку Find Previous в панели инструментов.

***Рис. 13.16.*** *Всплывающая подсказка для значка Find Previous*

Команда :tmenu определяет всплывающие подсказки как для меню, так и для панели инструментов. Она имеет следующий синтаксис:

```
:tmenu TopMenu.NextLevelMenu.MenuItem tool tip text
```

где *TopMenu.NextLevelMenu.MenuItem* определяет меню каскадом от верхнего уровня до пункта меню, для которого задается всплывающая подсказка. Например, для пункта Open из меню File она будет определяться командой:

```
:tmenu File.Open Open a file
```

При определении пункта панели инструментов используйте ToolBar в качестве «меню» самого высокого уровня (для панели инструментов реального меню верхнего уровня не существует).

Определим всплывающую подсказку для значка панели инструментов DOS из предыдущего раздела. Введите команду:

```
:tmenu ToolBar.DOSWindow Open up a DOS window
```

Теперь всякий раз при удержании мыши над недавно созданной кнопочкой вы увидите всплывающую подсказку, как показано на рис. 13.17.

***Рис. 13.17.*** *Панель с новой командой DOS и новая всплывающая подсказка*

## gvim в Microsoft Windows

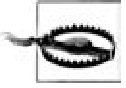
Редактор `gvim` становится все более популярным среди пользователей Windows. Старожилы `vi` и Vim увидят, что версия для Windows превосходна. Кроме того, она, скорее всего, является самой свежей для всех операционных систем.

Программа-установщик автоматически и аккуратно интегрирует Vim в окружение Windows. Если нет, обратитесь за справкой к файлу `gui-w32.txt` в каталоге с исполняемым файлом Vim, где прописаны инструкции для программы `regedit`. Поскольку это требует редактирования реестра Windows, *не стоит этого делать*, если при этом вы чувствуете малейший дискомфорт. Лучше обратитесь за помощью к более компетентному в данном вопросе пользователю. Это обычная, но нетривиальная процедура.

Пользователи Windows со стажем знакомы с *буфером обмена* – областью памяти, где содержится текст и другая информация, которую можно использовать при операциях копирования, вырезания и вставки. Vim поддерживает взаимодействие с буфером Windows. Просто выделите текст в визуальном режиме и выберите пункт меню Copy или Cut, после чего текст из Vim будет перенесен в буфер обмена Windows. После этого вы сможете вставить его в другие приложения Windows.

## gvim в X Window System

Пользователи, знакомые с окружением X, могут определять и использовать многие легко настраиваемые функции X. Например, можно задавать ресурсы при помощи стандартных определений класса, обычно содержащихся в файле `.Xdefaults`.



Обратите внимание, что эти стандартные ресурсы X будут полезными только для GUI-версий Motif или Athena. Очевидно, версия Windows не поймет ресурсы X. Менее очевидно, что они не воспринимаются ни в KDE, ни в GNOME.

Полное описание X, а также процесса его конфигурации и настройки скрупулезно задокументировано и не рассматривается в этой книге. Для краткого (или не очень краткого) введения в X мы рекомендуем соответствующее руководство (man).

## Опции GUI и обзор команд

В табл. 13.3 собраны команды и опции, специально предназначенные для `gvim`. Они добавляются в Vim, скомпилированный с поддержкой GUI, и становятся действенными, если вызвать редактор командой `gvim` или `vim -g`.

*Таблица 13.3. Опции, специфические для gvim*

| Команда или опция | Тип | Описание |
|---|---|---|
| guicursor | опция | Настройки формы и мерцания курсора |
| guifont | опция | Имена используемых однобайтовых шрифтов |
| guifontset | опция | Имена используемых многобайтовых шрифтов |
| guifontwide | опция | Список имен шрифтов для символов двойной ширины |
| guiheadroom | опция | Количество пикселов, оставленных для оконных рамок |
| guioptions | опция | Узнать, какие опции и компоненты используются |
| guipty | опция | Использовать псевдотерминал для команды «:!» |
| guitablabel | опция | Пользовательская метка для корешков вкладок |
| guitabtooltip | опция | Пользовательская всплывающая подсказка для корешков вкладок |
| toolbar | опция | Пункты, отображаемые в панели инструментов |
| -g | ключ командной строки | Запуск GUI (также допускает и другие опции) |
| -U gvimrc | ключ командной строки | Использовать при запуске GUI файл инициализации gvimrc либо файл с похожим именем |
| :gui | команда | Запуск GUI (только в UNIX-подобных системах) |
| :gui *filename*... | команда | Запуск GUI вместе с редактированием указанных файлов |
| :menu | команда | Перечислить все меню |
| :menu *menupath* | команда | Перечислить меню, которые начинаются с *menupath* |
| :menu *menupath action* | команда | Добавить меню *menupath*, чтобы произвести действие *action* |
| :menu *n menupath action* | команда | Добавить меню *menupath* с приоритетом позиции *n* |

| Команда или опция | Тип | Описание |
|---|---|---|
| :menu ToolBar.*toolbarname action* | команда | Добавить пункт панели инструментов *toolbarname*, чтобы произвести действие *action* |
| :tmenu *menupath text* | команда | Создать всплывающую подсказку для пункта меню *menupath* с текстом *text* |
| :unmenu *menupath* | команда | Удалить меню *menupath* |

# 14

# Улучшения Vim для программистов

Редактирование текстов – не единственная область применения Vim. Хорошим программистам требуются мощные средства, гарантирующие эффективную профессиональную работу. Хорошая программа-редактор – только начало, и одного его недостаточно. Многие современные среды программирования пытаются обеспечить всесторонние решения, однако в действительности необходим лишь мощный редактор с грамотными дополнениями.

Инструменты программирования предоставляют дополнительные функции, начиная от подсветки синтаксиса, автоматических отступов, форматирования, автозавершения ключевых слов и т. д. до полнофункциональной *интегрированной среды разработки* (IDE) с изощренной интеграцией, составляющей полную экосистему разработки. Такие IDE могут быть дорогостоящими (например, Visual Studio) или свободно распространяемыми (Eclipse), но их требования к дисковому пространству и памяти очень высоки, они имеют крутую кривую обучения и безмерные запросы ресурсов.

У программистов разные цели и требования к технологиям. Небольшие задачи легко решаются простыми редакторами, возможности которых чуть больше, чем простая правка текста. Тяжелые, многокомпонентные, многоплатформенные и требующие целого штата программистов задачи *почти наверняка* потребуют «тяжелой артиллерии», обеспечиваемой IDE. Однако, основываясь на своем обширном опыте, многие умудренные пользователи знают, что IDE дает едва ли больше, чем дополнительную сложность, при этом не повышая вероятность решения задачи.

Vim предлагает прекрасный компромисс между простыми редакторами и монолитными IDE. Некоторые его возможности до недавнего времени существовали только в дорогих средах разработки. Он позволяет быстро и просто решать задачи программирования без перегрузки и сложности обучения, присущих IDE.

Многие опции, функции и команды специально разработаны для облегчения работы программиста – от сворачивания строк программы в одну строку до расцветки синтаксиса и автоматического форматирования. Vim обеспечивает специалистов множеством утилит, ценность которых постигается только при их использовании. С одной стороны, он предоставляет в некотором роде «мини-IDE» под названием Quickfix, а с другой – содержит удобные функции для разнообразных задач программирования. В этой главе мы представим следующие темы:

- Свертка
- Автоматические и умные отступы
- Автозавершение ключевых слов и слов из словаря
- Теги и расширенные теги
- Подсветка синтаксиса
- Авторская (ваша собственная) подсветка синтаксиса
- Quickfix, «мини-IDE» в Vim

# Свертка и контуры (режим контуров)

Свертка (fold) позволяет определять, какие части файла оставлять видимыми. Например, в блоке кода можно спрятать все, что находится внутри фигурных скобок, или скрыть все комментарии. Свертка – двухэтапный процесс. Сначала нужно определить содержимое сворачиваемого блока с использованием одного из *методов свертки* (про них мы расскажем позже). Затем при вызове команды свертки Vim прячет обозначенный текст и ставит на его место однострочный заполнитель. Рисунок 14.1 показывает, как выглядит свертка в Vim. Заполнитель свертки позволяет управлять строками, которые он скрывает.

```
      2
      3
      4
      5 int fcn (int v1, int v2)
      6   {
      7
      8   printf ("02 some line\n");
-     9   printf ("03 some line\n");
|+   10 +---  2 lines: printf ("04 some line\n");--------
     12   printf ("06 some line\n");
     13
     14   if (thiscode == anysense)
+    15 +--  8 lines: {----------------------------------
     23
     24   printf ("07 some line\n");
     25   printf ("08 some line\n");
+    26 +--  4 lines: printf ("09 some line\n");---------
     30
     31   }
   ~
```

***Рис. 14.1.*** *Пример сверток в Vim*

В этом примере строка 11 скрыта в двухстрочной свертке, начиная со строки 10. Восьмистрочная свертка, начинающаяся со строки 15, скрывает строки с 16 по 22. Четырехстрочная свертка, начинающаяся на строке 26, скрывает строки с 27 по 29.

Теоретически не существует предела для количества создаваемых сверток. Вы также можете создавать вложенные свертки (свертки внутри сверток).

Создание и отображение сверток в Vim управляется несколькими опциями. Также, если вы насоздавали множество сверток, Vim предоставит вам команды, облегчающие работу с ними: `:mkview` и `:loadview`. Они служат для сохранения сверток между сеансами работы[1], чтобы не пришлось создавать их снова.

Изучение сверток требует некоторого времени, однако, освоив их, вы получаете мощный инструмент управления отображением кода. Не стоит недооценивать его преимущества. Правильная и читаемая программа требует надежного дизайна на нескольких уровнях, а хорошее программирование часто требует умения «увидеть лес за деревьями», другими словами, игнорировать некоторые детали реализации, чтобы видеть полную структуру файла.

Для опытных пользователей Vim предоставляет шесть различных способов определения, создания и управления свертками. Эта гибкость позволяет создавать свертки и управлять ими в различных контекстах. Важно то, что после создания свертки открываются, закрываются и одинаково реагируют на все команды, входящие в комплект работы со свертками.

Ниже приведены шесть методов создания сверток:

`manual`

: Определяет охват свертки с помощью стандартных конструкций Vim, например команд перемещения.

`indent`

: Свертки и уровни сверток соответствуют уровням отступа текста и значению параметра `shiftwidth`.

`expr`

: Свертку определяют регулярные выражения.

`syntax`

: Свертки соответствуют семантике языка файла (например, в программе на C сворачиваются блоки отдельных функций).

`diff`

: Свертки определяются различиями между двумя файлами.

---

[1] Свертки также могут сохраняться наряду с другими настраиваемыми объектами Vim при вызове команды `:mksession`; см. главу 15. – *Прим. науч. ред.*

`marker`

Границы сверток определяются предопределенными (но также и определяемыми пользователем) маркерами в файле.

Работа со свертками (открытие, закрытие, удаление и прочее) одинакова для всех методов. Мы рассмотрим ручные свертки и детально обсудим команды Vim для работы со свертками, а также затронем отдельные подробности, относящиеся к другим методам, которые являются сложными, узкоспециальными и лежат за пределами нашего введения. Надеемся, что наш обзор подтолкнет вас к исследованиям богатства остальных методов.

Рассмотрим кратко основные команды свертки и приведем небольшой пример использования сверток на практике.

## Команды свертки

Все команды свертки начинаются с `z`. В качестве мнемоники для запоминания представьте сложенный (соответствующим образом) лист бумаги и его сходство с буквой «z».

Существует примерно 20 команд свертки, начинающихся с `z`. С их помощью можно создавать и удалять, открывать и закрывать (прятать и показывать принадлежащий свертке текст) свертки, а также переключать их состояние (скрытая/раскрытая). Ниже даны краткие описания этих команд:

`zA`

Переключает состояние сверток, рекурсивно.

`zC`

Закрывает свертки, рекурсивно.

`zD`

Удаляет свертки, рекурсивно.

`zE`

Удаляет *все* свертки.[1]

`zf`

Создает свертку с текущей строки до той, куда переместит курсор последующая команда перемещения.

*count*`zF`

Создает свертку, охватывающую *count* строк, начиная с текущей.

`zM`

Устанавливает опцию `foldlevel` равной нулю.

[1] В отличие от команды `zD`, действующей на текущую (и вложенные в нее) свертку, `zE` удаляет все свертки в буфере независимо от положения курсора. – *Прим. науч. ред.*

`zN`, `zn`

Устанавливает (`zN`) или сбрасывает (`zn`) опцию `foldenable`.

`zO`

Раскрывает свертки, рекурсивно.

`za`

Переключает состояние одной свертки.

`zc`

Закрывает одну свертку.

`zd`

Удаляет одну свертку.

`zi`

Переключает значение опции `foldenable`.

`zj`, `zk`

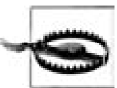
Перемещает курсор на начало следующей (`zj`) или на конец предыдущей (`zk`) свертки. (Обратите внимание на мнемонику команд `j` («jump») и `k`, а также на их сходство с аналогичными командами перемещения в контексте сверток.)

`zm`, `zr`

Уменьшает (`zm`) или увеличивает (`zr`) на единицу значение опции `foldlevel`.

`zo`

Открывает одну свертку.



Не следует путать *удаление свертки* с командой *удаления*. Используйте команду *удаления свертки* для удаления определения свертки. Удаление свертки не оказывает никакого влияния на текст, содержащийся в этой свертке.

Команды `zA`, `zC`, `zD` и `zO` называются *рекурсивными*, поскольку они действуют на все свертки, вложенные в ту, к которой вы применяли эти команды.

## Ручное сворачивание

Если вам знакомы команды перемещения в Vim, то вы уже владеете половиной информации, нужной для эффективной работы с созданными вручную свертками.

Например, для сворачивания трех строк введите следующее:

```
3zF
2zfj
```

`3zF` выполняет команду сворачивания `zF` для трех строк, начиная с текущей; `2zfj` – с текущей строки до той, куда `j` переместит курсор (в нашем случае – на две строки вниз).

Попробуем более сложную команду для программистов на С. Чтобы свернуть блок кода на С, поместите курсор на начальную или конечную скобки ({ или }) программного блока и введите `zf%` (помните, что `%` перемещает на вторую соответствующую скобку).

Создайте свертку от курсора до начала файла, введя `zfgg` (`gg` перемещает на начало файла).

Свертки проще понять на примерах. Возьмем простой файл, создадим свертки, поупражняемся с ними и посмотрим на результаты. Мы также увидим некоторые визуальные подсказки (cues) сверток, предоставляемые Vim.

Сначала рассмотрим пример файла с рис. 14.2, содержащего некоторые (бессмысленные) строки кода на С. Изначально никаких сверток не было.

```
 4
 5 int fcn (int v1, int v2)
 6   {
 7
 8   printf ("02 some line\n");
 9   printf ("03 some line\n");
10   printf ("04 some line\n");
11   printf ("05 some line\n");
12   printf ("06 some line\n");
13
14   if (thiscode == anysense)
15     {
16
17     printf ("07 some other line\n");
18     printf ("08 some other line\n");
19     printf ("09 some other line\n");
20     printf ("10 some other line\n");
21
22     }
23
24   printf ("07 some line\n");
25   printf ("08 some line\n");
26   printf ("09 some line\n");
27   printf ("10 some line\n");
28   printf ("11 some line\n");
29   printf ("12 some line\n");
30
31   }
```

***Рис. 14.2.*** *Файл-образец без сверток*

Несколько замечаний по этому рисунку. Во-первых, на левой стороне экрана Vim отображает номера строк. Мы рекомендуем всегда оставлять их видимыми (с помощью опции `number`), чтобы иметь больше зрительной информации о положении в файле, а в данной ситуации, когда часть строк пропадает из вида, эта информация имеет еще большую ценность. Vim показывает количество неотображаемых строк, а номера подтверждают и подкрепляют эти данные.

Также обратите внимание на серые столбцы слева от номеров строк. Они зарезервированы для дополнительных визуальных подсказок сверткам. По мере изучения сверток мы увидим визуальные подсказки, которые Vim будет вставлять в эти столбцы.

Обратите внимание, что на рис. 14.2 курсор находится на строке 18. Поместим ее и две последующие строки в одну свертку, после чего введем `zf2j`. На рис. 14.3 показан результат.

```
   13
   14   if (thiscode == anysense)
   15     {
   16
   17     printf ("07 some other line\n");
+  18 +--  3 lines: printf ("08 some other line\n");-----
   21
   22     }
   23
   24   printf ("07 some line\n");
```

***Рис. 14.3.*** *Три строки, свернутые на строке 18*

Обратите внимание, что Vim создает легко распознаваемый маркер с сочетанием `+--` в начале, а текст первой строки *свертки* размещен в заполнителе. Кроме того, слева программа вставила знак `+`. Это еще одна визуальная подсказка.

Теперь в том же файле мы свернем следующий программный блок, находящийся между скобок после оператора `if`. Поместите курсор на любую из этих скобок и введите `zf%`. После этого файл примет вид, показанный на рис. 14.4.

```
    8   printf ("02 some line\n");
    9   printf ("03 some line\n");
   10   printf ("04 some line\n");
   11   printf ("05 some line\n");
   12   printf ("06 some line\n");
   13
   14   if (thiscode == anysense)
+  15 +--  8 lines: {---------------------------
   23
   24   printf ("07 some line\n");
   25   printf ("08 some line\n");
   26   printf ("09 some line\n");
```

***Рис. 14.4.*** *Программный код, свернутый за оператором if*

Сейчас свернуто восемь строк кода, три из которых содержатся в ранее созданной свертке. Это называется *вложенной* сверткой. Заметим, что никаких указаний на наличие вложенной свертки нет.

Наш следующий эксперимент – помещение курсора на строку 25 и сворачивание всех предыдущих строк вплоть до объявления функции `fcn`

включительно. На этот раз применим команду перемещения с *поиском*. Начните команду свертки с zf, произведите обратный поиск к началу fcn, используя ?int fcn (команда обратного поиска в Vim), и нажмите ENTER. Экран приобретет вид, как на рис. 14.5.

```
    3
    4
+   5 +-- 21 lines: int fcn (int v1, int v2)--------
   26   printf ("09 some line\n");
   27   printf ("10 some line\n");
   28   printf ("11 some line\n");
   29   printf ("12 some line\n");
   30
   31   }
~
```

***Рис. 14.5.*** *Свертка до начала функции*

Если вы подсчитываете строки и создаете свертку, охватывающую другую свертку (например, 3zf), то все строки в этой свертке рассматриваются как одна строка. Например, если курсор находится на строке 30, а строки 31–35 скрыты в свертке на следующей строке экрана (то есть следующая строка экрана имеет номер 36), то 3zf создаст новую свертку, содержащую три строки, как они показаны на экране: текстовая строка 30, пять строк, содержащихся в свертке со строками 31–35, и текстовая строка 36, отображаемая ниже на экране. Неожиданно? Немного. Можно считать, что команда zf считает строки по правилу «Что видим, то и сворачиваем».

Попробуем другие функции. Во-первых, откроем все свертки с помощью команды zO (после z стоит буква O, а не ноль). Мы увидим визуальные подсказки в левом поле, как показано на рис. 14.6. Каждый из столбцов в этом поле называется *foldcolumn*.

На рисунке первая строка каждой свертки отмечена знаком минус (-), а все остальные – вертикальной чертой (|). Самая большая (самая внешняя) свертка находится в самом левом столбце, а самая внутренняя – в самом правом. Как видно из рисунка, строки 5–25 представляют уровень самой низкой вложенности (в нашем случае – первый), строки 15–22 – следующий уровень (2), а строки 18–20 – самый высокий уровень.

По умолчанию эта полезная визуальная метафора отключена (не знаем, почему; может быть, потому что отнимает много места на экране). Включить и определить ее ширину можно командой:

```
:set foldcolumns=n
```

где *n* – количество используемых столбцов (максимум 12, минимум 0). На рисунке мы использовали foldcolumn=5 (внимательный читатель заметит, что на более ранних рисунках foldcolumn был равен 3. Мы поменяли значение для лучшего зрительного восприятия).

```
 4
 5 int fcn (int v1, int v2)
 6   {
 7
 8   printf ("02 some line\n");
 9   printf ("03 some line\n");
10   printf ("04 some line\n");
11   printf ("05 some line\n");
12   printf ("06 some line\n");
13
14   if (thiscode == anysense)
15     {
16
17     printf ("07 some other line\n");
18     printf ("08 some other line\n");
19     printf ("09 some other line\n");
20     printf ("10 some other line\n");
21
22     }
23
24   printf ("07 some line\n");
25   printf ("08 some line\n");
```

***Рис. 14.6.*** *Все свертки открыты*

```
 5 int fcn (int v1, int v2)
 6   {
 7
 8   printf ("02 some line\n");
 9   printf ("03 some line\n");
10   printf ("04 some line\n");
11   printf ("05 some line\n");
12   printf ("06 some line\n");
13
14   if (thiscode == anysense)
15     {
16
17     printf ("07 some other line\n");
18 +---  3 lines: printf ("08 some other line\n");---
21
22     }
```

***Рис. 14.7.*** *После повторной свертки строк 18–20*

Теперь создайте другие свертки и понаблюдайте за их действиями.

Для начала снова сверните самую глубокую свертку, содержащую строки 18–20, поместив курсор на одну из строк из диапазона и введя `zc` (закрыть свертку). Рисунок 14.7 приводит результат.

Увидели изменение в сером поле? Vim поддерживает визуальные подсказки, упрощая визуализацию и управление свертками.

Далее посмотрим, как работает в свертке типичная «однострочная» команда. Поместите курсор на свернутую строку (18) и введите `~~` (смена регистра у всех символов текущей строки). Не забывайте, что в Vim ко-

манда ~ является объектным оператором (если только не включена опция compatible), и, следовательно, он должен сменить регистр у всех символов строки в нашем примере. После этого откройте свертку, введя zo (команда открытия свертки). Теперь программа выглядит, как на рис. 14.8.

```
|      14   if (thiscode == anysense)
|-     15     {
||     16
||     17       printf ("07 some other line\n");
||-    18       PRINTF ("08 SOME OTHER LINE\N");
|||    19       PRINTF ("09 SOME OTHER LINE\N");
|||    20       PRINTF ("10 SOME OTHER LINE\N");
||     21
||     22     }
|      23
|      24   printf ("07 some line\n");
```

***Рис. 14.8.*** *Смена регистра, примененная к свертке*

Любое действие в свертке влияет на всю свертку. Например, на рис. 14.7 при помещении курсора на строку 18 – свертку, скрывающую строки с 18 по 20, – и ввода dd (удаление строки) будут удалены все три строки вместе со сверткой.

Важно отметить, что Vim управляет всеми командами редактирования так, будто бы никаких сверток не было, то есть любая отмена аннулирует всю операцию редактирования. Например, если вы ввели u (отмена) после предыдущей правки, то все три удаленные строки будут восстановлены. Функция отмены отличается от «однострочных» команд, рассмотренных в этом разделе, хотя иногда их действия кажутся аналогичными.

Теперь самое время познакомиться с визуальными подсказками поля foldcolumn. Они облегчают обзор свертки, на которую вы собираетесь подействовать. Например, команда zc (закрытие свертки) закрывает самую внутреннюю свертку, содержащую строку с курсором. Величину этой свертки можно узнать по вертикальным черточкам в foldcolumn. Как только вы освоитесь с этим, такие действия, как открытие, закрытие и удаление сверток, станут для вас естественными.

## Схема

Рассмотрим следующий простой (и не слишком естественный) файл, где в качестве отступов используется табуляция:

```
1. This is Headline ONE with NO indentation and NO fold level.
    1.1 This is sub-headline ONE under headline ONE
        This is a paragraph under the headline. Its fold
        level is 2.
    1.2 This is sub-headline TWO under headline ONE.
```

```
2. This is Headline TWO. No indentation, so no folds!
    2.1 This is sub-headline ONE under headline TWO.
        Like the indented paragraph above, this has fold level 2.
            - Here is a bullet at fold level 3.
                A paragraph at fold level 4.
            - Here is the next bullet, again back at fold level 3.
      And, another set of bullets:
            - Bullet one.
            - Bullet two.
    2.2 This is heading TWO under Headline TWO.
3. This is Headline THREE.
```

Свертки Vim можно использовать для просмотра файла в виде схемы (outline). Определите метод сворачивания как `indent`:

```
:set foldmethod=indent
```

В нашем файле зададим `shiftwidth` (уровень отступов для табуляций) равным 4. После этого мы сможем открывать и закрывать свертки на основании отступов строк. Для каждого `shiftwidth` (в нашем случае – кратного четырем столбцам) в строке с отступом уровень свертки увеличивается на 1. Например, в нашем файле подзаголовки имеют отступ, равный одному `shiftwidth` или четырем столбцам. Следовательно, уровень сворачивания равен 1. Строки с отступом восемь столбцов (два `shiftwidth`) имеют уровень 2 и т. д.

Уровнем видимых сверток можно управлять с помощью команды `foldlevel`. В качестве аргумента ей передается целое число, после чего она отображает только те строки, где уровень свертки *меньше или равен* этому аргументу. В рассматриваемом файле следующей командой можно указать редактору показывать только заголовки самого высокого уровня:

```
:set foldlevel=0
```

Экран приобретет вид, как на рис. 14.9.

Отобразить все вплоть до пунктов ненумерованных списков, включая их самих, можно путем установки `foldlevel`, равным 2. Все свертки с уровнем *не меньше* 2 будут видимы, как на рис. 14.10.

Используя этот метод для проверки файла, вы сможете быстро разворачивать и сворачивать видимые уровни детализации с помощью команд инкремента (`zr`) и декремента (`zm`).

```
1. This is Headline ONE with NO indentation and NO fold level.
+--  4 lines: 1.1 This is sub-headline ONE under headline ONE---------
2. This is Headline TWO.  No indentation, so no folds!
+--  5 lines: 2.1 This is sub-headline ONE under headline TWO.--------
+--  4 lines: And, another set of bullets:----------------------------
3. This is Headline THREE.
~
```

***Рис. 14.9.*** *fold level = 0*

```
1. This is Headline ONE with NO indentation and NO fold level.
    1.1 This is sub-headline ONE under headline ONE
        This is a paragraph under the headline.  Its fold
        level is 2.
    1.2 This is sub-headline TWO under headline ONE.
2. This is Headline TWO.  No indentation, so no folds!
    2.1 This is sub-headline ONE under headline TWO.
        Like the indented paragraph above, this has fold level 2.
+----  3 lines: - Here is a bullet at fold level 3.---------------
        And, another set of bullets:
+----  2 lines: - Bullet one.--------------------------------------
    2.2 This is heading TWO under Headline TWO.
3. This is Headline THREE.
```

***Рис. 14.10.*** *fold level = 2*

## Несколько слов о других методах сворачивания

У нас нет времени на рассмотрение всех остальных методов сворачивания, но для возбуждения вашего аппетита сделаем беглый обзор *синтаксического* (*syntax*) метода.

Возьмем уже знакомый файл C, но на этот раз позволим Vim решать, что нужно свернуть на основании синтаксиса C. Этот язык имеет сложные правила сворачивания, однако простого куска кода в примере достаточно для демонстрации возможностей редактора по автоматизации.

Во-первых, с помощью команды zE (удаление всех сверток) убедимся, что мы избавились от всех сверток. Теперь экран показывает весь код без визуальных маркеров в столбце сверток (слева).

Командой

```
:set foldenable
```

убедитесь, что сворачивание включено (в случае *ручного* сворачивания можно было этого не делать, поскольку foldenable автоматически устанавливается при задании foldmethod, равным manual). Введите команду:

```
:set foldmethod=syntax
```

Свертки приобретут вид, как на рис. 14.11.

Vim свернул все программные блоки в скобках, поскольку они являются логически смысловыми блоками в C. Если выполнить zo в строке 6 этого примера, то Vim раскроет свертку и отобразит внутреннюю.

```
      2
      3
      4
      5 int fcn (int v1, int v2)
+     6 +-- 26 lines: {---------------------------
   ~
   ~
   ~
```

***Рис. 14.11.*** *После применения команды set foldmethod=syntax*

Каждый метод сворачивания использует различные правила для определения сверток. Мы призываем засучить (развернуть?) рукава и узнать в документации Vim больше об этих мощнейших средствах.

Режим diff в Vim (который можно включить командой `vimdiff`) – это мощное сочетание сворачивания, окон и подсветки синтаксиса, который мы обсудим позже. Как показано на рис. 14.12, этот режим отображает различия в файлах (обычно между двумя версиями одного файла).

***Рис. 14.12.*** *Функция diff Vim и ее использование в свертках*

# Автоматические и умные отступы

Vim предлагает чрезвычайно сложные и мощные методы автоматической расстановки отступов в тексте. В общем случае он ведет себя почти аналогично `vi` с включенной опцией `autoindent`, и для описания такого поведения используется то же имя.

Метод расстановки отступов можно выбирать, просто указав его название в команде `:set`:

```
:set cindent
```

В Vim есть следующие методы, перечисленные в порядке возрастания сложности:

`autoindent`

Автоматические отступы, похожие на `autoindent` в `vi`. Небольшие отличия заключаются в месте помещения курсора при удалении отступа.

`smartindent`

Чуть мощнее, чем `autoindent`, но распознает некоторые примитивы синтаксиса C для определения уровней отступов.

`cindent`

Как следует из названия, `cindent` включает гораздо больше знаний о синтаксисе C и вводит усложненную настройку, выходящую за рамки простых уровней отступа. Например, `cindent` можно настроить так, чтобы он соответствовал вашим (или вашего босса) любимым

стилям программирования, включая (но не ограничиваясь) отступы фигурных скобок ({}), размещение скобок, наличие отступов у обеих скобок и даже способ соответствия отступа включенному тексту.

indentexpr

Позволяет определять пользовательские выражения, которые Vim вычисляет в контексте каждой начатой новой строки. Имея такую функцию, вы сами пишете собственные правила. За подробностями обращайтесь к описанию скриптов и функций в этой книге, а также к документации Vim. Если первые три опции не давали достаточной гибкости для автоматических отступов, то indentexpr, несомненно, обеспечит ее.

## Расширения autoindent Vim по сравнению с vi

В Vim autoindent ведет себя почти как в vi, к тому же его можно сделать совершенно аналогичным, включив опцию compatible. Одно из полезных расширений Vim заключается в его способности распознавать «тип» файла и вставлять подходящие символы комментария, когда закомментированный текст переходит на новую строку. Эта функция прекрасно работает как с опцией wrapmargin (текст переносится, оставляя по правому краю количество столбцов, указанное в wrapmargin), так и с опцией textwidth (текст переносится, когда число символов в строке превысит показатель textwidth). На рис. 14.3 показан результат в Vim и vi на одном и том же файле.

```
  1 #! /bin/sh
  2
  3 if [ $xyz -eq 0 ]
  4 then
  5     # this block of comments I typed
  6     # with option textwidth set to 40,
  7     # autoindent on, and
  8     # (automatically), syntax=sh.
  9     # Notice how each line has the '#'
 10     # with a separating space, all
 11     # courtesy of vim's autoindent...
 12     # now I will type the same text but
 13     # instead with the option
 14     # "compatible" (with vi) set...
 15
 16     # this block of comments I typed with option textwidth s
 17     and (automatically), syntax=sh.  Notice how each line h
 18     separating space, all courtesy of vim's autoindent... n
 19     text but instead with the option "compatible" (with vi)
```

***Рис. 14.13.*** *Различия между autoindent в Vim и в vi*

Обратите внимание, что во втором текстовом блоке (строка 16 и ниже) отсутствует первый символ комментария. Кроме того, если установлена опция compatible (подражание действиям vi), то опция textwidth не бу-

дет распознаваться, поэтому текст переносится на новые строки только по значению опции `wrapmargin`.

## smartindent

`smartindent` — это слегка расширенный `autoindent`. Метод может оказаться полезным, но если вы программируете на C и используете сложные конструкции, то лучше пользоваться `cindent`.

`smartindent` автоматически расставляет отступ, если:

- Новая строка следует за строкой, содержащей левую фигурную скобку.
- Новая строка начинается с ключевого слова, которое перечислено в опции `cinwords`.
- Созданная новая строка предшествует строке, начинающейся с правой фигурной скобки (}), *если* курсор был помещен на строку, содержащую скобку, и пользователь создал новую строку с помощью команды `O` (создание новой строки выше текущей).
- Новая строка — это закрывающая (или правая) фигурная скобка (}).

## cindent

Скорее всего, пользователи Vim, программирующие на подобных C языках, захотят использовать `cindent` или `indentexpr`. Хотя `indentexpr` более мощный и гибкий, а также лучше настраивается, для многих задач программирования более практичен `cindent`. В нем есть множество опций для самых разных нужд программистов (и под различные корпоративные стандарты). Сначала попробуйте `cindent` с настройками по умолчанию, а затем поменяйте их, если они не соответствуют вашим стандартам.

Если опция `indentexpr` задана, она отменяет действие `cindent`.

Поведение `cindent` определяется тремя опциями:

`cinkeys`

Задает клавиши клавиатуры, заставляющие Vim заново рассчитать отступы.

`cinoptions`

Определяет стиль отступов.

`cinwords`

Определяет ключевые слова, предписывающие Vim добавить дополнительный отступ к последующим строкам.

`cindent` использует строку, заданную `cinkeys`, как набор правил, определяющих отступы. Мы рассмотрим значение `cinkeys` по умолчанию, а затем изучим другие доступные значения и принципы их работы.

## Опция cinkeys

`cinkeys` – это список из следующих элементов, разделенных запятыми:

```
0{,0},0),:,0#,!^X^F,o,O,e
```

Ниже перечислены их значения, разнесенные по контекстам, с кратким описанием действия каждого из них:

`0{`

`0` (ноль) устанавливает контекст *начала строки* для следующего символа, `{`. То есть если вы ввели символ `{` первым в строке, Vim пересчитает отступ для данной строки.

Не путайте ноль, стоящий в этой опции, с действием «использовать здесь нулевой отступ», представляющим собой обычное дело при отступах в C. В нашем случае ноль означает, что «если символ введен в начале строки, не вынуждать его отображаться в начале строки».

Значение отступа для `{` по умолчанию равно нулю: поверх текущего отступа новый добавляться не будет. Следующий пример показывает типичный результат:

```
main ()
{
    if ( argv[0] == (char *)NULL )
    { ...
```

`0}, 0)`

Как и в предыдущем примере, эти две установки определяют контекст *начала строки*. То есть если в начале строки ввести `}` или `)`, то Vim пересчитает отступ.

Заданный по умолчанию отступ для `}` соответствует отступу, установленному для соответствующей открывающей скобки `{`.

Отступ для `)` по умолчанию равняется одному `shiftwidth`.

`:`

Это контекст для метки C или для `case statement`. Если `:` (двоеточие) поставлено в конце метки или оператора `case statement`, то Vim пересчитает отступ.

По умолчанию для `:` отступом является столбец 1 – первый столбец в файле. Не путайте его с нулевым отступом, когда следующая строка имеет тот же уровень отступа, что и предыдущая. Когда отступ равен 1, первый символ новой строки сдвигается влево *к* самому первому столбцу.

`0#`

Это также контекст *начала строки*. Когда первым символом строки является `#`, Vim пересчитывает отступ.

Как и в предыдущем определении, по умолчанию отступ сдвигает всю строку к первому столбцу. Это соответствует практике начинать макросы (`#define...`) с первого столбца.

`!^F`

Специальный символ определяет любой последующий символ как *переключатель* для пересчета отступа текущей строки. В приведенном примере символ-переключатель – `^F` (или CTRL-F), то есть при каждом нажатии CTRL-F по умолчанию Vim будет заново вычислять отступ для данной строки.

`o`

Этот контекст определяет любую создаваемую вами новую строку либо по нажатию клавиши ENTER в *режиме вставки*, либо с использованием команды `o` (новая строка).

`O`

Отвечает за создание новой строки *выше* текущей при помощи команды `O` (новая строка выше текущей).

`e`

Контекст для *else*. Если вы начинаете строку со слова `else`, Vim пересчитывает отступ. Редактор не будет распознавать этот контекст, пока не будет набрана последняя «e» в слове *else*.

**Правила синтаксиса cinkeys**. Каждое определение `cinkeys` состоит из необязательного префикса (`!`, `*` или `0`) и клавиши, для которой пересчитывается отступ. Префиксы имеют следующие значения:

`!`

Указывает на клавишу (по умолчанию CTRL-F), заставляющую Vim пересчитывать отступ текущей строки. Можно добавлять дополнительное определение клавиши как команды (используя `+=`), не отменяя предопределенных команд. Другими словами, можно задавать несколько клавиш, пересчитывающих отступ строки. Любая клавиша, добавленная в определение `!`, будет выполнять и старую функцию.

`*`

Предписывает Vim пересчитать отступ текущей строки перед тем, как встаить символ.

`0`

Устанавливает контекст *начала строки*. Клавиша, которую вы пропишите после `0`, будет вызывать пересчет отступа, только если она будет первым символом в строке.

Нужно понимать отличие в Vim и `vi` между «первым символом строки» и «первым столбцом строки». Мы уже знаем, что ввод `^` перемещает курсор на первый символ в строке, но не обязательно на первый столбец (по левому краю); то же относится к вставке с `I`. Точно так же префикс `0` применяется к вводу символа как первого в строке независимо от того, выровнен он по левому краю или нет.

cinkeys использует специальные названия клавиш и дает возможность задействовать любой из зарезервированных символов, например, используемых в качестве префиксов. У специальных клавиш есть следующие опции:

<>

Эта форма используется для определения символов буквально. Для некоторых непечатаемых символов используйте их «побуквенные» представления. Например, с помощью <:> можно определить символ «:». Другой пример – непечатаемый символ «стрелка вверх», определяемый как <Up>.

^

Каретка (^) используется для обозначения управляющего символа. Например, ^F определяет клавишу CTRL-F.

o, O, e, :

Мы уже говорили об этих клавишах в качестве значений cinkeys по умолчанию.

=*word*, =~*word*

Используется для определения слова, вызывающего специальное поведение. Если символы, являющиеся начальным текстом в строке, соответствуют *word*, Vim пересчитывает отступ.

Форма =~word аналогична =word, только она игнорирует регистр.

К сожалению, термин «слово» (*word*) используется здесь неверно. Правильнее говорить, что *word* представляет *начало слова*, так как переключение происходит при совпадении строки со «словом», причем конец образца для сравнения может не совпадать с концом слова. Во встроенной документации Vim приводится пример слова end, соответствующего как end, так и endif.

## Опция cinwords

cinwords определяет ключевые слова, которые, будучи набранными, включают дополнительный отступ у следующей строки. По умолчанию эта опция содержит:

```
if,else,while,do,for,switch
```

Эти слова охватывают стандартные ключевые слова в языке программирования C.

Эти ключевые слова зависят от регистра. При такой проверке Vim даже игнорирует опцию ignorecase. Если вам нужны варианты с различными регистрами в ключевых словах, придется прописать в строке cinwords все возможные комбинации.

## Опция cinoptions

cinoptions управляет расстановкой Vim отступов строк с учетом контекста C. Данная опция включает установки для управления несколькими стандартами форматирования кода, например:

- Величиной отступа программного блока, заключенного в фигурные скобки.
- Необходимостью вставки новой строки перед фигурной скобкой, стоящей после условного оператора.
- Способом выравнивания программных блоков относительно обрамляющих их скобок.

В варианте по умолчанию cinoptions определяет 28 установок:

```
s,e0,n0,f0,{0,}0,^0,:s,=s,l0,b0,gs,hs,ps,ts,is,+s,c3,C0,/0,(2s,us,U0,w0,W0,
     m0,j0,)20,*30
```

Уже сам перечень этих вариантов дает представление о количестве способов задания правил отступов в Vim. Большая часть настроек cinoptions определяет различия в контекстных блоках. В некоторых настройках задается, насколько далеко необходимо сканировать (сколько строк файла вверх и вниз нужно пройти) для установки правильного контекста и вычисления отступов без ошибок.

Установки, меняющие количество отступов для различных контекстов, могут увеличивать или уменьшать уровни отступов. Кроме того, можно переопределять количество столбцов, используемых в отступах. Например, установка cinoptions=f5 приводит к тому, что открывающая фигурная скобка ({) будет отступать на пять столбцов, если только она не располагается внутри *других* скобок.

Другой способ определить увеличение отступов заключается в использовании некоторого множителя (он необязательно должен быть целым числом) для shiftwidth. Если в предыдущем примере приписать в определение w (то есть cinoptions=f5w), то открывающая фигурная скобка сместится на пять shiftwidth.

Для изменения уровня отступов в меньшую сторону (отрицательный отступ) вставьте знак минус (-) перед числовым значением.

Опцию cinoptions и ее строковое значение нужно менять с особой осторожностью. Не забывайте, что при использовании = вы переопределите всю опцию. Так как cinoptions содержит множество всевозможных установок, для ее изменения используйте команды: += для добавления параметра, -= для удаления существующего параметра, а для изменения существующего — -=, за которым следует +=.

Далее следует краткий список опций, которые вам, возможно, захочется поменять. Это малое подмножество установок в cinoptions. Многие читатели могут захотеть использовать другие (или даже *все*) опции для своих настроек.

>*n* (*по умолчанию* s)

Любая строка с отступом должна быть сдвинута на *n* столбцов. По умолчанию стоит s, то есть по умолчанию отступ строки равен одному shiftwidth.

f*n*, {*n*

f определяет, насколько далеко нужно ставить отступ у открывающей невложенной фигурной скобки ({). По умолчанию стоит ноль, то есть скобка выравнивается по своему логическому «собрату». Например, фигурная скобка, стоящая после строки с оператором while, помещается под буквой «w» в слове while.

{ ведет себя аналогично f, но применяется и к *вложенным* фигурным скобкам. Здесь значение по умолчанию соответствует нулевому отступу.

На рис. 14.14 и 14.15 показан один и тот же текст в Vim, но в первом случае cinoptions=s,f0,{0, а во втором cinoptions=s,fs,{s. В обоих примерах опция shiftwidth имеет значение 4 (четыре столбца).

```
18
19 while (condition)
20 {
21     if (someothercondition)
22     {
23         printf("looks like I've got both conditions!\n");
24     }
25 }
26
```

***Рис. 14.14****. cinoptions=s,f0,{0*

```
27 while (condition)
28     {
29     if (someothercondition)
30         {
31         printf("looks like I've got both conditions!\n");
32         }
33     }
34 |
```

***Рис. 14.15****. cinoptions=s,fs,{s*

}*n*

Эта установка используется, чтобы определить смещение закрывающей фигурной скобки (}) по отношению к соответствующей открывающей. По умолчанию стоит ноль (то есть скобки выровнены относительно друг друга).

^*n*

Добавляет *n* к текущему отступу внутри набора скобок ({...}), если открывающая скобка находится в первом столбце.

`:n`, `=n`, `bn`

Эти три установки контролируют отступы в операторах `case`. С помощью `:` Vim ставит отступ у меток равным *n* символов по отношению к соответствующему оператору `switch`. Значение по умолчанию – один `shiftwidth`.

Установка `=` определяет смещение для строк кода от соответствующей им метки. По умолчанию операторы отступают на один `shiftwidth`.

`b` задает место установки операторов `break`. Значение по умолчанию (ноль) выравнивает `break` с остальными операторами в соответствующем блоке `case`. Любое ненулевое значение выравнивает `break` с соответствующей меткой.

`)n`, `*n`

Эти две установки передают Vim количество строк, которые нужно сканировать при поиске незакрытой круглой скобки (20 строк по умолчанию) и незакрытых комментариев (по умолчанию 30 строк), соответственно.

Эти два параметра, предполагается, должны ограничить объем работы Vim при поиске совпадений. При работе на современных мощных компьютерах можно подумать об увеличении этих значений, обеспечивая больший объем поиска соответствий комментариям и скобкам. Для начала попробуйте удвоить каждое значение до 40 и 60 соответственно.

## indentexpr

Будучи определенной, опция `indentexpr` отменяет действие `cindent`, так что можете определять правила проставления отступов и подстраивать их под ваш язык программирования.

`indentexpr` определяет выражение, вычисляемое каждый раз при создании новой строки в файле. Это выражение определяет целое число, которое Vim использует для задания отступа новой строки.

Кроме того, опция `indentkeys` позволяет определять полезные ключевые слова аналогично ключевым словам в `cinkeys`, задающим строки, после которых отступ пересчитывается.

Плохая новость заключается в том, что записать пользовательские правила проставления отступов с нуля – нетривиальная задача для любого языка. Хорошая же состоит в том, что, скорее всего, эта работа уже проделана. Посмотрите в каталог `$VIMRUNTIME/indent` и проверьте, есть ли там ваш любимый язык. На сегодняшний день там содержится около 70 файлов с правилами отступов.

В них представлены самые популярные языки программирования, такие как *ada*, *awk*, *docbook* (файл называется `docbk`), *eiffel*, *fortran*, *html*, *java*, *lisp*, *pascal*, *perl*, *php*, *python*, *ruby*, *scheme*, *sh*, *sql* и *zsh*. Есть даже файл, определенный для `xinetd`!

Можно указать Vim автоматически распознавать тип файла и подгружать файл отступов, если в файле `.vimrc` прописать строку `filetype indent on`. После этого редактор будет пытаться определить тип редактируемого файла и подгружать *соответствующий* файл с определениями отступов. Если правила не будут отвечать вашим требованиям, например, отступы ставятся неправильно, отключите определение с помощью команды `:filetype indent off`.

Мы хотим побудить опытных пользователей исследовать и изучать файлы определений отступов, поставляемые с Vim. А если вы разработаете новый файл определений или усовершенствуете существующий, не стесняйтесь и отправьте его на *vim.org*. Возможно, он войдет в будущий пакет Vim.

## Заключительное слово об отступах

Перед тем, как закончить обсуждение, стоит отметить несколько моментов при работе с автоматическими отступами:

*Когда автоматический отступ не работает*

Всякий раз, когда вы воздействуете на строку во время сеанса редактирования с включенными автоматическими отступами и меняете отступ вручную, Vim помечает эту строку, после чего он больше не будет определять ее отступ самостоятельно.

*Копирование и вставка*

При вставке текста в файл с включенными автоматическими отступами Vim рассматривает этот текст как ввод и применяет все правила автоматического проставления отступов. В большинстве случаев нам этого не нужно. Любые отступы во вставляемом тексте будут складываться с отступами, расставляемыми по правилам. Обычно это приводит к сильному перекосу текста вправо с большими отступами и без соответствующего возврата влево.

Чтобы избежать такого неудобства и вставлять текст без побочных эффектов, перед добавлением внешнего текста установите в Vim опцию `paste`. Она полностью перенастраивает все автоматические функции редактора, чтобы правильно внести вставляемый текст. Для возврата к автоматическому режиму сбросьте опцию `paste` с помощью команды `:set nopaste`.

# Ключевые слова и завершение слов по словарю

Vim предоставляет всесторонний набор инструментов *автозавершения ввода*. Он знает, что предлагать в качестве завершения для частично введенных слов, от ключевых слов языков программирования до имен файлов, слов из словаря и даже целых строк. Более того, Vim поддерживает механизм автозавершения, основанного на словаре, чтобы включить автозавершения, основанные на синонимах завершенного слова из тезауруса!

В этом разделе мы рассмотрим различные методы завершения, их синтаксис и описание их работы с примерами. Методы завершения включают:

- Строку целиком
- Ключевые слова из текущего файла
- Ключевые слова из опции `dictionary`
- Ключевые слова из опции `thesaurus`
- Ключевые слова из текущего и *внешнего* файлов
- Теги (как в `ctags`)
- Имена файлов
- Макросы
- Командную строка Vim
- Пользовательские методы
- Omni
- Предложения написания
- Ключевые слова из опции `complete`

Кроме последней, все команды завершения начинаются с CTRL-X. Вторая клавиша определяет тип завершения, который нужно использовать. Например, CTRL-X CTRL-F – это команда автозавершения имени файла (к сожалению, не все команды такие же мнемонические). Vim использует сочетания клавиш, по умолчанию не имеющих отображения, так что правильным отображением можно сократить большую часть этих команд до второго сочетания клавиш (например, отобразить CTRL-X CTRL-N на простое CTRL-N).

В принципе, все методы завершения ведут себя одинаково: они прокручивают список кандидатов на завершение при повторном нажатии на второе сочетание клавиш. То есть, если вы выбрали автозавершение имени файла, нажав CTRL-X CTRL-F, и при первой попытке подходящего слова не нашлось, то можно продолжать нажимать CTRL-F, получая все новые и новые варианты. Кроме того, нажав CTRL-N (от «next»), вы перейдете вперед по этим вариантам, а CTRL-P (от «previous») переместит вас назад.

Рассмотрим несколько методов автозавершения с примерами, чтобы увидеть, какая от них польза.

## Команды завершения вставки

Эти методы отличаются по своим функциям, от простого просмотра слов в текущем файле до охвата диапазона функций, переменных, макросов и других имен по всем файлам проекта. Последний приведенный метод сочетает свойства других и достигает компромисса между мощью и сложностью.

Возможно, вам захочется найти любимый метод завершения и отобразить его на одну простую клавишу. Я отобразил свой на Tab:

```
:imap Tab <C-P>
```

Этим я принес в жертву возможность быстрого создания табуляций, но получил возможность использовать ту же клавишу, которую я использую (по умолчанию) в режимах командной строки (в DOS или оболочках `xterm`, `konsole` и т. д.) для завершения частично введенного текста. (Не забывайте, что табуляцию можно вставить, если экранировать ее с помощью CTRL-V.) Отображение на клавишу Tab также соответствует обычной клавише завершения в режиме командной строки Vim.

## Завершение целой строки

Метод вызывается с помощью CTRL-X CTRL-L. Он ищет в текущем файле строку, соответствующую введенным символам. Приведем пример, чтобы вы поняли, как работает завершение.

Рассмотрим файл, содержащий консольные (терминальные) определения, характеризующие функции терминала и способ управления им. Пусть ваш экран похож на рис. 14.16.

```
  1 #       Reconstructed via infocmp from file: /etc/terminfo/r/rx
  2 # (untranslatable capabilities removed to fit entry within 1023
  3 # This terminal widely used in our company...
  4 rxvt-unicode|rxvt-unicode terminal (X Window System):\
  5         :am:bw:eo:hs:km:mi:ms:xn:xo:\
  6         :co#80:it#8:li#24:lm#0:\
  7         :AL=\E[%dL:DC=\E[%dP:DL=\E[%dM:DO=\E[%dB:IC=\E[%d@:\
  8         :K1=\EOw:K2=\EOu:K3=\EOy:K4=\EOq:K5=\EOs:LE=\E[%dD:\
  9         :RI=\E[%dC:SF=\E[%dS:SR=\E[%dT:UP=\E[%dA:ae=\E(B:al=\E[L
```

***Рис. 14.16.*** *Пример завершения строки*

Обратите внимание, что подсвеченная строка содержит «This terminal widely used in our company...». Эта строка требуется во многих местах, где вы отмечаете терминалы как «широкоиспользуемые» (widely used) в своей компании. Просто введите достаточно символов из этой строки, чтобы она стала уникальной или близкой к уникальной, а затем – CTRL-X CTRL-L. Теперь на рис. 14.17 отображена часть строки ввода:

```
# Thi
```

CTRL-X CTRL-L указывает Vim отобразить набор возможных завершений для строки, основываясь на ранее введенном в файле тексте. Список завершений показан на рис. 14.18.

В оттенках серого видно не так отчетливо, но на экране вы увидите цветное всплывающее окно, содержащее несколько строк, соответствующих

началу введенного фрагмента. Также отображается информация о том, где было найдено соответствие, хотя она и не видна на снимке экрана.

Этот метод использует опцию `complete` для определения области поиска соответствий. Эти области подробно обсуждаются в последнем методе в этом разделе.

При перемещении вперед (CTRL-N) или назад (CTRL-P) по списку во всплывающем[1] окне подсвечивается выбранный вариант. Нажмите ENTER для подтверждения своего выбора[2]. Если вам ничего не подходит из этого списка, нажмите CTRL-E, чтобы прекратить поиск соответствия, не производя замену текста. При этом курсор останется в первоначальном положении на изначально введенном фрагменте.

На рис. 14.19 показан результат после выбора одного из вариантов списка.

```
4 rxvt-unicode|rxvt-unicode terminal (X Wi
5         :am:bw:eo:hs:km:mi:ms:xn:xo:\
6         :co#80:it#8:li#24:lm#0:
7
8 # Thi
9 rxvt-unicode2|rxvt-unicode2 terminal (X
0         :AL=\E[%dL:DC=\E[%dP:DL=\E[%dM:
```

***Рис. 14.17.*** *Частично введенная строка, ожидающая завершения*

```
 8 # This terminal widely used in our company...
 9 # This section lists entries in a least-capable to most-cap
10 # This should only be used when the terminal emulator canno
11 # This entry is good for the 1.1.47 version of the Linux co
12 # This trick could work with other Intel consoles like the
13 # This terminal widely used in our company...
```

***Рис. 14.18.*** *После ввода CTRL-X CTRL-L*

```
3 # This terminal widely used in our company...
4 rxvt-unicode|rxvt-unicode terminal (X Window System):\
5         :am:bw:eo:hs:km:mi:ms:xn:xo:\
6         :co#80:it#8:li#24:lm#0:
7
8 # This terminal widely used in our company...
9 rxvt-unicode2|rxvt-unicode2 terminal (X Window System)2:\
10        :AL=\E[%dL:DC=\E[%dP:DL=\E[%dM:DO=\E[%dB:IC=\E[%d@
```

***Рис. 14.19.*** *Результат ввода CTRL-X CTRL-L и выбора соответствующей строки*

[1] Появляется в `gvim`; Vim ведет себя немного по-другому.

[2] В актуальной версии Vim (7.3) достаточно нажать любую клавишу, меняющую содержимое буфера (букву, цифру, знак препинания и т. п.). – *Прим. науч. ред.*

## Завершение по ключевому слову из файла

CTRL-X CTRL-N выполняет в текущем файле поиск ключевых слов, соответствующих ключевому слову, стоящему перед курсором. После нажатия этих клавиш можно использовать CTRL-N и CTRL-P для перемещения вперед и назад соответственно. Нажмите ENTER, чтобы выбрать соответствие.

Обратите внимание, что термин «ключевые слова» используется достаточно свободно. Это могут быть ключевые слова, известные программистам, а также любые другие слова в файле. Слова определяются как последовательный набор символов в опции `iskeyword`. По умолчанию эта опция установлена довольно разумно, однако можно переопределить ее, если вы хотите внести новую пунктуацию или убрать старую. Символы в `iskeyword` можно указать непосредственно (например, `a-z`) или по коду ASCII (к примеру, для представления a–z используется `97-122`).

Например, установки по умолчанию допускают подчеркивание как часть слова, но точку или минус считают разделителями. В C-подобных языках это работает хорошо, но в других окружениях может подвести.

## Завершение по словарю

CTRL-X CTRL-K ищет ключевые слова, соответствующие ключевому слову перед курсором в файлах, определенных в опции `dictionary`.

В установке по умолчанию опция `dictionary` не определена. Существуют предустановленные места размещения файлов со словарями, но также можно определить свой собственный словарь. Наиболее популярными словарями являются следующие файлы:

- `/usr/dict/words` (Cygwin в XP)
- `/usr/share/dict/words` (FreeBSD)
- `$HOME/.mydict` (персональный список словарных слов)

Обратите внимание, что в Windows XP словарный файл предоставляется в Cygwin (*http://www.cygwin.com/*) – свободной программе, эмулирующей комплект утилит UNIX. Хотя установка Cygwin не затрагивается в этой книге, стоит упомянуть, что можно выборочно установить отдельные части Cygwin (можно установить только ту ее часть, где содержатся словари).

## Завершение по тезаурусу

CTRL-X CTRL-T ищет ключевые слова, соответствующие ключевому слову перед курсором в файлах, определенных в опции `thesaurus`.

В этом методе есть один интересный момент. Когда Vim находит соответствие (и если строка в файле тезауруса содержит больше одного слова), Vim включает все эти слова в качестве кандидатов на завершение.

Можно предположить (и это отражено в названии метода), что он предлагает синонимы, позволяя при этом определить свои собственные. Рассмотрим пример файла, содержащего следующие строки:

```
fun enjoyable desirable
funny hilarious lol rotfl lmao
retrieve getchar getcwd getdirentries getenv getgrent ...
```

Первые две строки – это типичные синонимы английских слов (соответствующие словам «fun» и «funny»), а вот третья может быть полезной для программистов на C, которые постоянно вставляют имена функций, начинающихся на `get`. Для всех этих функций мы задали синоним «retrieve».

На практике лучше отделить английский тезаурус от тезауруса языка C, ведь Vim может искать в нескольких тезаурусах.

В режиме ввода введите слово `fun`, затем CTRL-X CTRL-T. Рисунок 14.20 показывает, что у вас отобразится в `gvim`.

Обратите внимание на следующее:

- Vim считает соответствующими *все* слова, которые может найти в записи тезауруса, а не только первое слово в этой строке.
- Он включает слова-кандидаты из всех строк в тезаурусе, соответствующие ключевому слову перед курсором. Так, в нашем случае он предлагает соответствия как для «fun», так и для «funny».

Другим интересным и, возможно, непредвиденным действием тезауруса является поиск в качестве соответствия слов, *отличных* от первого слова в строке файла тезауруса. Например, в строке файла из предыдущего примера:

```
funny hilarious lol rotfl lmao
```

Если вы введете `hilar` и завершите его, Vim включит в список все слова, начиная от `hilarious` в этой строке, то есть «hilarious», «lol», «rotfl» и «lmao». Здорово!

Заметили дополнительную информацию в списке кандидатов на завершение? Если добавить в опцию `completeopt` значение `preview`, то из всплывающего меню можно будет узнать о том, где Vim нашел соответствие.

Рассмотрим пример с использованием предыдущего файла, но на сей раз вводимым фрагментом слова будет `retrie`. Оно соответствует «retrieve» – синониму, который мы выбрали для «get» и вписали все имена функций-«получателей» в качестве синонимов. Теперь сочетание CTRL-X CTRL-T выдаст всплывающее меню (в `gvim`), где присутствуют все эти функции на выбор. См. рис. 14.21.

Как и в любом другом методе завершения, для подтверждения выбора нажмите ENTER.

```
1674         Unix</emphasis>.  While installatio
1675 fun
1676 fun       c:\home\ehannah\.mythesaurus otin
1677 enjoyable c:\home\ehannah\.mythesaurus mpha
1678 desirable c:\home\ehannah\.mythesaurus y.</
1679 funny     c:\home\ehannah\.mythesaurus
1680 hilarious c:\home\ehannah\.mythesaurus
1681 lol       c:\home\ehannah\.mythesaurus
1682 rotfl     c:\home\ehannah\.mythesaurus
1683 lmao      c:\home\ehannah\.mythesaurus
1684         <title><emphasis>completion method<
1685         (<keycap>CTL-X CTL-T</keycap>) </ti
```

***Рис. 14.20.*** *Завершение для «fun» из тезауруса*

```
46   printf ("02 some line\n");
47   retrieve
48   retrieve          c:\home\ehannah\.mythesaurus
49   getchar           c:\home\ehannah\.mythesaurus
50   getcwd            c:\home\ehannah\.mythesaurus
51   getdirentries     c:\home\ehannah\.mythesaurus
52   getenv            c:\home\ehannah\.mythesaurus
53   getgrent          c:\home\ehannah\.mythesaurus
54   getgrgid          c:\home\ehannah\.mythesaurus
55   getgrnam          c:\home\ehannah\.mythesaurus
56   gethostbyaddr     c:\home\ehannah\.mythesaurus
57   gethostbyname     c:\home\ehannah\.mythesaurus
58   getmntent         c:\home\ehannah\.mythesaurus
59   getnetbyaddr      c:\home\ehannah\.mythesaurus
60   getnetbyname      c:\home\ehannah\.mythesaurus
61   getnetent         c:\home\ehannah\.mythesaurus
62   getopt            c:\home\ehannah\.mythesaurus
63   getopt_long       c:\home\ehannah\.mythesaurus
64   getpass           c:\home\ehannah\.mythesaurus
65   getprotobyname    c:\home\ehannah\.mythesaurus
66   getprotobynumber  c:\home\ehannah\.mythesaurus
67   getprotoent       c:\home\ehannah\.mythesaurus
```

***Рис. 14.21.*** *Завершение по тезаурусу для строки «retrie»*

## Завершение по ключевому слову в текущем и внешних файлах

CTRL-X CTRL-I ищет ключевые слова, соответствующие ключевому слову перед курсором в текущем файле и в подключаемых внешних (include) файлах. Этот метод отличается от метода «поиска в текущем файле» (CTRL-X CTRL-P) тем, что Vim ищет в текущем файле *ссылки на внешние файлы* и производит поиск также и в них.

Vim использует значение опции `include` для определения строк, ссылающихся на *внешние* файлы. По умолчанию в ней задан шаблон, предписывающий редактору искать строки, соответствующие стандартной конструкции в C:

```
# include <somefile.h>
```

В этом случае Vim будет искать соответствия в файле `somefile.h` в стандартных системных каталогах заголовочных файлов. Он также использует опцию `path` в качестве списка каталогов для поиска внешних файлов.

## Завершение по тегу

CTRL-X CTRL-] ищет ключевые слова, соответствующие *тегу* в текущем и включенных файлах (за информацией о тегах обратитесь к разделу «Использование тегов» на стр. 147).

## Завершение по имени файла

CTRL-X CTRL-F ищет имена файлов, соответствующие ключевому слову перед курсором. Обратите внимание, что здесь Vim завершает ключевое слово *именем файла*, а не содержащимся в файле словом.

Vim 7.1 ищет файлы для возможного соответствия имени файла *только* в текущем каталоге. Это поведение отличается от многих других функций Vim, где для поиска файлов используется опция `path`. Встроенная документация редактора подсказывает, что это временное явление и отмечает, что `path` «пока еще» не используется[1].

## Завершение по макросу и именам определений

CTRL-X CTRL-D ищет в текущем и во внешних файлах имена макросов и макроопределений, задаваемых директивой `#define`. В этом методе также производится поиск внешних ссылок в текущем файле и поиск в найденных файлах.

## Метод завершения по командам Vim

Этот метод, вызываемый через CTRL-X CTRL-V, подразумевает его использование в командной строке Vim и пытается угадать наилучшее завершение для слов. Данный контекст предназначен для помощи пользователям при разработке скриптов Vim.

## Завершение по пользовательским функциям

Метод вызывается по CTRL-X CTRL-U и позволяет определять метод завершения вашими собственными функциями. Для завершения Vim использует функцию, на которую ссылается опция `completefunc`. За информацией о скриптах и написании функций Vim обратитесь к главе 12.

## Завершение по omni-функции[2]

Этот метод, вызываемый через CTRL-X CTRL-O, использует пользовательские функции примерно так же, как предыдущий. Важное отли-

[1] Такая ситуация сохраняется и в Vim 7.3. – *Прим. науч. ред.*

[2] Во многих IDE данная функция называется «интеллектуальным завершением»: она подсказывает имена полей классов, методов и т. п. – *Прим. науч. ред.*

чие состоит в том, что данный метод полагает, что функция предназначена для файлов определенного типа и, следовательно, определяется и загружается при загрузке файла. Файлы omni-завершения уже доступны для C, CSS, HTML, JavaScript, PHP, Python, Ruby, SQL и XML. Во встроенной документации Vim упоминается, что вскоре для версии Vim 7.1 будет доступно еще больше скриптов, включая файл функций omni для C++. Мы рекомендуем поэкспериментировать с ними.

## Завершение для исправления орфографии

Метод вызывается по CTRL-X CTRL-S. Слово перед курсором используется как основа, для которой Vim предлагает варианты замены. Если слово написано неправильно, редактор предлагает «более правильный» вариант.

## Завершение с опцией complete

Это обобщенный вариант, вызываемый по CTRL-N и сочетающий в себе все остальные методы. Для большинства пользователей этот вариант будет самым оптимальным, поскольку он практически не требует разбираться в нюансах более узких методов.

В опции `complete` устанавливается, где и как действует завершение. Для этого нужно прописать список доступных источников, разделенных запятыми. Каждый источник представлен одной буквой. Выбор включает в себя:

`.` (*точка*)

: Поиск в текущем буфере.

`w`

: Поиск в буферах в других окнах (внутри экрана, содержащего сеанс работы Vim).

`b`

: Поиск в других загруженных буферах в списке буферов (которые могут не быть видимыми ни в одном окне Vim).

`u`

: Поиск в выгруженных буферах из списка буферов.

`U`

: Поиск в буферах, *не* представленных в списке буферов.

`k`

: Поиск в файлах словарей (перечисленных в опции `dictionary`).

`kspell`

: Использует текущую схему проверки орфографии (это единственная из опций, которая содержит больше одной буквы).

`s`

: Поиск в файлах тезауруса (перечисленных в опции `thesaurus`).

`i`

Поиск в текущем и внешних файлах.

`d`

Поиск по макросам, определенным в текущем и внешних файлах.

`t, ]`

Поиск завершения по тегу.

## Заключительные замечания по автозавершению в Vim

Мы рассмотрели огромный материал, связанный с автозавершением, но это далеко не все. Методы автозавершения с лихвой окупают время, затраченное на их изучение. Если вам приходится *много* редактировать и если имеется *любой* вид обобщения или контекста для завершения, выберите наиболее соответствующий вашему случаю метод и освойте его.

И последний совет. При нажатии двух клавиш (если вы пользователь UNIX, то подсчитываете комбинации клавиш как «больше чем одна») возможны ошибки, особенно при использовании сочетаний с клавишей CTRL. Если вы планируете часто использовать автозавершение, то стоит подумать об отображении любимой команды автозавершения на одну кнопку или сочетание клавиш.

Следующий пример показывает, почему мы считаем настройку столь важной. Как упоминалось ранее, я отобразил общий поиск соответствия ключевого слова на клавишу Tab. При редактировании этой книги с помощью тегов DocBook я вводил (как показал простой `grep` по файлам) слово «emphasis» более 1200 раз! При использовании автозавершения по ключевым словам я знаю, что «emph» всегда соответствует нужному мне тегу «emphasis». Таким образом, для каждого вхождения этого слова я экономлю по крайней мере три нажатия клавиши (считая, что первые три буквы вводятся правильно). Это дало экономию, по крайней мере, 3600 нажатий!

Приведем другой способ измерения эффективности этого метода: я знаю, что ввожу примерно четыре символа в секунду. Таким образом, экономия при вводе *только одного* ключевого слова составит 3600/4, то есть 15 *сэкономленных минут*. На тех же файлах DocBook я так же использовал завершение других 20 или 30 ключевых слов. Оцените рост экономии времени!

# Стеки тегов

Стеки тегов описываются в разделе «Стеки тегов» на стр. 157. Кроме перемещения вперед и назад по тегам, по которым ведется поиск, можно выбирать среди нескольких найденных тегов. Также можно выбирать теги и разделять окно всего одной командой. Команды режима `ex` в Vim для работы с тегами приведены в табл. 14.1.

*Таблица 14.1. Команды работы с тегами в Vim*

| Команда | Функция |
|---|---|
| ta[g][!] [*tagstring*] | Редактирование файла, содержащего *tagstring*, как задано в файле тегов. Знак ! вынуждает Vim переключиться в новый файл, если текущий буфер изменен, но не сохранен. Файл может быть записан либо нет в зависимости от опции autowrite. |
| [*count*]ta[g][!] | Переход на *count*-ю новую запись в стеке тегов. |
| [*count*]po[p][!] | Извлекает позицию курсора из стека, восстанавливая курсор в его предыдущей позиции. Если стоит *count*, то переход будет на *count*-ю старую позицию. |
| tags | Отображает содержимое стека тегов. |
| ts[elect][!] [*tagstring*] | Выводит список тегов, соответствующих *tagstring*, используя информацию в файлах тегов. Если *tagstring* не задан, используется последнее имя из стека тегов. |
| sts[elect][!] [*tagstring*] | Подобна :tselect, но разделяет окно для выбранных тегов. |
| [*count*]tn[ext][!] | Переход на *count*-й следующий соответствующий тег (по умолчанию – на первый). |
| [*count*]tp[revious][!]<br>[*count*]tN[ext][!] | Переход на *count*-й предыдущий соответствующий тег (по умолчанию – на первый). |
| [*count*]tr[ewind][!] | Переход на первый соответствующий тег. Если стоит *count*, то переход будет на *count*-й соответствующий тег. |
| tl[ast][!] | Переход на последующий соответствующий тег. |

Как правило, Vim показывает, на какой тег из какого их общего количества был переход. Например:

```
tag 1 of >3
```

Знак больше (>) указывает на то, что еще не все соответствия использовались. Команды :tnext и :tlast применяются для перебора тегов. Если подобное сообщение не отображается из-за какого-то другого сообщения, введите :0tn, чтобы его увидеть.

Ниже приведен вывод команды :tags, а текущее положение отмечено знаком больше (>):

```
  # TO tag   FROM line in file
  1 1 main           1 harddisk2:text/vim/test
> 2 2 FuncA         58 -current-
  3 1 FuncC        357 harddisk2:text/vim/src/amiga.c
```

Команда :tselect позволит выбрать из более чем одного соответствующего тега. «Приоритет» (поле pri) указывает на качество соответствия (глобальный, а не статический; не регистронезависимость, а точный регистр символов; и т. д.). Более подробно это описано в документации Vim.

```
  nr pri kind tag              file ~
   1 F   f    mch_delay        os_amiga.c
                 mch_delay(msec, ignoreinput)
 > 2 F   f    mch_delay        os_msdos.c
                 mch_delay(msec, ignoreinput)
   3 F   f    mch_delay        os_unix.c
                 mch_delay(msec, ignoreinput)
Enter nr of choice (<CR> to abort):
```

Командам :tag и :tselect можно задать аргумент, который начинается с /. В этом случае команда использует его как регулярное выражение, и Vim найдет все теги, соответствующие данному регулярному выражению. Например, :tag /normal найдет макрос NORMAL, функцию normal_cmd и т. д. Введите :tselect /normal и укажите количество нужных вам тегов.

Команды для командного режима vi описаны в табл. 14.2. Как и в других редакторах, при наличии поддержки мыши в вашем Vim ее можно использовать наравне с клавиатурой.

*Таблица 14.2. Команды работы с тегами в командном режиме Vim*

| Команда | Функция |
|---|---|
| ^]<br>g <LeftMouse><br>CTRL-<LeftMouse> | Ищет положение стоящего под курсором идентификатора в файле тегов и переходит на это место. Текущая позиция заносится в стек тегов. |
| ^T | Возврат к предыдущему положению в стеке тегов, то есть извлечение одного элемента. Предшествующее число указывает на количество элементов в стеке тегов, которое нужно вынуть. |

Опции Vim, влияющие на поиск, описаны в табл. 14.3.

*Таблица 14.3. Опции Vim для управления тегами*

| Опция | Функция |
|---|---|
| taglength, tl | Задает количество значимых символов в теге, по которым будет производиться поиск. Значение по умолчанию, равное нулю, указывает, что важны все символы. |
| tags | Значением является список файлов, в которых следует искать теги. В особом случае, когда имя файла начинается с ./, точка заменяется на каталог в пути текущего файла, что позволяет делать поиск и в других каталогах. Значение по умолчанию: "./tags,tags". |
| tagrelative | Если установить эту опцию в true (значение по умолчанию) и использовать файл тегов из другого каталога, то имена файлов в этих файлах тегов рассматриваются относительно каталога, где расположен файл тегов. |

Vim может использовать файлы etags в стиле Emacs, но они нужны только для обратной совместимости. Этот формат не описан в документации Vim, а использование файлов etags не приветствуется.

Наконец, Vim также смотрит на все слово, содержащее курсор, а не только на его часть от положения курсора.

# Подсветка синтаксиса

Подсветка синтаксиса – одно из самых главных усовершенствований в Vim по сравнению с `vi`. Синтаксическое форматирование в Vim во многом опирается на использование цвета, хотя это практически бессмысленно на черно-белых мониторах. В этом разделе мы обсудим три темы: начало работы, настройка и создание своей расцветки. Подсветка синтаксиса для Vim содержит функции, выходящие за рамки этой книги, так что мы сосредоточимся на предоставлении достаточного количества ознакомительной информации, которую потом вы сможете расширить для своих потребностей.

Поскольку воздействие подсветки синтаксиса в Vim наиболее наглядно в цвете, а эта книга не цветная, мы настоятельно рекомендуем испробовать подсветку, чтобы полностью оценить всю силу цвета в определении контекста. Я еще не встречал пользователя, который, испытав ее, отказался бы от ее применения.

## Как начать

Отобразить файл с подсветкой синтаксиса очень просто. Выполните команду:

```
:syntax enable
```

Если все хорошо и вы редактировали файл с формальным синтаксисом, таким как язык программирования, то вы увидите текст, раскрашенный в разные цвета, каждый из которых определяется контекстом и синтаксисом. Если ничего не изменилось, попробуйте включить *syntax* так:

```
:syntax on
```

Включение синтаксиса через `enable` должно работать, однако мы сталкивались с ситуацией, когда для его активации требовалась дополнительная команда.

Если вы все еще не видите подсветки синтаксиса, то, наверное, Vim не знает тип вашего файла, поэтому он не может решить, какой синтаксис будет подходящим. Причины для этого могут быть разные.

Например, если вы создали новый файл и используете нераспространенное расширение либо не ставите вообще никакого расширения, то Vim не может определить тип файла, так как это новый и, следовательно, пустой файл. Например, я пишу скрипты для оболочки без расширения `.sh`. Каждый новый скрипт этой оболочки начинает свою жизнь без подсветки синтаксиса. К счастью, как только в файле появляется

код, Vim уже знает, как распознать тип файла, и подсветка синтаксиса работает правильно.

Также возможно (хотя и маловероятно), что Vim не распознает тип вашего файла. Это бывает очень редко, и обычно в подобном случае просто нужно явно указать тип файла, потому что файл синтаксиса для этого языка кем-то уже написан. К сожалению, создание такого файла с нуля – сложное занятие, хотя позже в этой главе мы дадим несколько советов на этот счет.

Можно принудить Vim использовать выбранную вами подсветку синтаксиса, если установить его вручную из командной строки. Например, начиная редактировать новый скрипт оболочки, я всегда определяю синтаксис с помощью:

```
:set syntax=sh
```

В разделе «Динамическая конфигурация типов файлов при помощи скриптов» на стр. 236 показан хитроумный способ избежать этого шага.

После включения подсветки синтаксиса Vim настраивает ее, проходя по списку проверок. Не углубляясь в технические детали, скажем, что в конце концов Vim определяет тип вашего файла, находит файл определений для соответствующего синтаксиса и подгружает его. Стандартное расположение файлов синтаксиса – каталог `$VIMRUNTIME/syntax`.

Чтобы почувствовать всю широту набора определений синтаксиса, добавим, что каталог Vim содержит *почти 500 файлов синтаксиса*. Доступные определения охватывают широкую гамму от языков программирования (C, Java, HTML) до контента (`calendar`) и широко распространенных конфигурационных файлов (`fstab`, `xinetd`, `crontab`). Если Vim не распознал тип вашего файла, посмотрите в каталог `$VIMRUNTIME/syntax` и отыщите там файл с синтаксисом, наиболее близким к вашему.

## Настройка

Начав использовать подсветку синтаксиса, вы можете обнаружить, что некоторые цвета не работают. Также они могут быть плохо различимыми или просто вам не нравятся. В Vim есть несколько способов настройки цветов.

Рассмотрим несколько вещей, которые стоит проделать перед принятием более радикальных мер (например, написанием собственного синтаксиса, как описано в следующей главе), чтобы подсветка синтаксиса вам подошла.

Ниже приведены два самых распространенных и броских симптома того, что подсветка «сошла с ума»:

- Плохая контрастность, цвета похожи, их трудно отличить.
- Большое количество цветов, что придает тексту неприятный вид.

Хотя это субъективные недостатки, но сама возможность вносить пользовательские правки радует. Две команды, `colorscheme` и `highlight`, и одна опция, `background`, возможно, помогут внести правильный цветовой баланс.

Есть еще несколько команд и опций, с помощью которых можно настроить подсветку синтаксиса. После короткого введения в *группы* синтаксиса мы поговорим об этих командах и опциях в последующих разделах, но упор сделаем на три уже упомянутые.

## Группы синтаксиса

Vim классифицирует разные типы текста по группам. Этим группам присваиваются цвет и определения подсветки. Кроме того, редактор позволяет объединять группы в группы. Определения можно назначать разным уровням. Если вы присвоили определение группе, содержащей подгруппы, то, если не задано обратное, каждая подгруппа унаследует определения у родительской группы.

При подсветке синтаксиса используются следующие группы высокого уровня:

*Comment* (*комментарии*)

Комментарии для конкретного языка программирования, например:

```
// I am both a C++ and a JavaScript comment
```

*Constant* (*константа*)

Любая постоянная, например `TRUE`.

*Identifier* (*идентификатор*)

Имена функций и переменных.

*Type* (*тип*)

Объявления, такие как `int` и `struct` в C.

*Special* (*специальные*)

Специальные символы, например разделители.

Если взять группу «специальные» из последнего списка, то можно получить следующие примеры подгрупп:

- SpecialChar (специальный символ).
- Tag (тег).
- Delimiter (разделитель).
- SpecialComment (специальный комментарий).
- Debug (отладка).

Базового представления о подсветке синтаксиса, группах и подгруппах достаточно для изменения подсветки синтаксиса так, чтобы она удовлетворяла нашим нуждам.

## Команда colorscheme

Эта команда меняет цвет для разных расцветок синтаксических конструкций, таких как комментарии, ключевые слова или строки, путем переопределения этих групп синтаксиса. Vim поставляется со следующими выборами цветовых схем:

- `blue`
- `darkblue`
- `default`
- `delek`
- `desert`
- `elflord`
- `evening`
- `koehler`
- `morning`
- `murphy`
- `pablo`
- `peachpuff`
- `ron`
- `shine`
- `slate`
- `torte`
- `zellner`

Эти файлы расположены в каталоге `$VIMRUNTIME/colors`. Активировать любой из них можно командой:

```
:colorscheme schemeName
```

В неграфическом Vim вы можете быстро перебрать разные цветовые схемы следующим образом: введите часть команды `:color`, нажмите клавишу Tab для включения автозавершения команды, затем пробел, после чего нажимайте Tab для прокручивания различных вариантов.

В `gvim` можно сделать еще проще. Зайдите в меню Edit, переместите указатель мыши на подменю Colorscheme и «оторвите» меню (черта с ножницами). Теперь вы можете просмотреть каждый вариант, нажав на соответствующую кнопку.

## Установка опции background

При назначении цвета Vim первым делом пытается определить тип цвета фона вашего экрана. В этом редакторе есть всего две категории фона: темный и светлый. В зависимости от результатов назначения он устанавливает цвета по-разному. Будем надеяться, итогом станет сочетание цветов, которое хорошо смотрится на вашем фоне (с хорошей контраст-

ностью и совместимостью цветов). Хотя Vim при этом делает все возможное, правильное присваивание довольно сложно, а отнесение к темному или светлому субъективно. Иногда контраст такой, что в сеансе неудобно работать или сложно что-либо распознать.

Если цвета выглядят неподобающим образом, попробуйте явно выбрать установку background. Сначала посмотрите на текущий выбор:

```
:set background?
```

Таким образом, вы будете знать, что установка изменится. После этого выполните команду типа:

```
:set background=dark
```

Используйте опцию background совместно с командой colorscheme для тонкой настройки цветов экрана. Вместе они обычно дают подходящую цветовую палитру, на которую комфортно смотреть.

## Команда highlight

Команда highlight в Vim позволяет управлять разными группами и контролировать их вид в сеансе редактирования. Это мощная команда. Установки для различных групп можно проверять либо в виде списка, либо запросив специальную информацию о подсветке группы. Например:

```
:highlight comment
```

в моем сеансе редактирования соответствует рис. 14.22.

```
:highlight Comment
Comment        xxx term=bold ctermfg=4 guifg=Blue
Press ENTER or type command to continue
```

*Рис. 14.22. Подсветка комментария*

Вывод показывает вид комментариев в файле. На странице xxx выглядит темно-серым, но на экране он синий. Вывод term=bold означает, что на черно-белых терминалах комментарии будут выделены жирным. ctermfg=4 означает, что на цветном терминале, например под xterm на цветном мониторе, цвет текста для комментария будет соответствовать цвету DOS dark blue. Наконец, guifg=Blue означает, что графический интерфейс будет показывать комментарии синим цветом.

Цветовая схема DOS имеет более ограниченный набор цветов, чем современные графические интерфейсы. В DOS их восемь: black, red, green, yellow, blue, magenta, cyan и white. Каждый из них можно назначать в качестве цвета текста или фона и (необязательно) помечать как «bright» для большей яркости на экране. Vim использует аналогичное отображение для определения цветов текста в неграфических окнах, например в xterm.

> Графические окна предоставляют почти неограниченный набор цветов. Vim позволяет определить некоторые цвета общепринятыми названиями, например Blue, но также можно задать эти цвета значениями красного, зеленого и синего. Формат имеет вид #rrggbb, где # — сам символ, а rr, gg и bb — шестнадцатеричные числа, представляющие интенсивность каждого цвета. Например, красный цвет задается как #ff0000.

Команда highlight используется для изменения параметров группы, цвета которой вас не устраивают. Например, можно выяснить, что идентификаторы в этом файле раскрашены в интерфейсе GUI темно-голубым, как показывает вывод на рис. 14.23.

```
:highlight identifier
```

Цвет для идентификаторов можно переопределить командой:

```
:highlight identifiers guifg=red
```

Сейчас все идентификаторы на экране красного цвета (выглядит убого). Такой вид настройки не является гибким: он применяется ко всем типам файлов и не учитывает разные фоны и цветовые схемы.

```
[No Name][+]    09/02/2007 07:06:56 PM
:hi Identifier
Identifier     xxx term=underline ctermfg=6 guifg=DarkCya
Press ENTER or type command to continue
```

***Рис. 14.23.*** *Подсветка идентификаторов*

Чтобы увидеть, сколько существует различных определений подсветки и чему равны их значения, снова вызовем highlight:

```
:highlight
```

Рисунок 14.24 показывает небольшой пример результатов выполнения команды highlight.

```
Constant       xxx term=underline ctermfg=1 guifg=Magenta
Special        xxx term=bold ctermfg=5 guifg=SlateBlue
Identifier     xxx term=underline ctermfg=6 guifg=DarkCyan
Statement      xxx term=bold ctermfg=3 gui=bold guifg=Brov
PreProc        xxx term=underline ctermfg=5 guifg=Purple
type           xxx term=underline ctermfg=2 gui=bold guifc
Underlined     xxx term=underline cterm=underline ctermfg=
Ignore             ctermfg=15 guifg=bg
Error          xxx term=reverse ctermfg=15 ctermbg=9 guifc
String         xxx links to Constant
Character      xxx links to Constant
Number         xxx links to Constant
Boolean        xxx links to Constant
Float          xxx links to Number
Function       xxx links to Identifier
```

***Рис. 14.24.*** *Часть вывода результатов команды highlight*

Обратите внимание, что некоторые строки содержат полные определения (где приводится term, ctermfg и т. д.), а остальные имеют в себе атрибуты от своих родительских групп (например, String отсылает к Constant).

### Отмена действия файлов синтаксиса

В предыдущем разделе мы изучили, как определять атрибуты группы синтаксиса для всех экземпляров группы. Предположим, вы хотите изменить группу только в одном или нескольких определениях синтаксиса. Vim позволит сделать это с помощью каталога after. В нем можно создать сколько угодно файлов синтаксиса, которые Vim будет обрабатывать *после* обычного файла синтаксиса.

Чтобы создать его, просто включите команды подсветки (или любые команды обработки – понятие «после» является универсальным) в специальном файле в каталоге под названием after, который содержится в опции runtimepath. Теперь, когда Vim начнет устанавливать подсветку синтаксиса для данного типа файла, он также выполняет пользовательские команды из файла after.

Например, применим настройку к файлам XML, использующим синтаксис xml. Это значит, что Vim загрузил определения синтаксиса из файла xml.vim из каталога syntax. Как и в предыдущем примере, мы хотим, чтобы идентификаторы всегда были красными. Для этого создадим собственный файл с именем xml.vim в каталоге ~/.vim/after/syntax и вставим в него строку:

```
highlight identifier ctermfg=red guifg=red
```

Перед запуском этой настройки нужно убедиться, что ~/.vim/after/syntax есть в пути runtimepath:

```
:set runtimepath+=~/.vim/after/syntax
```
*В нашем .vimrc*

Конечно, для сохранения изменений эта строка должна попасть в файл .vimrc.

Теперь Vim, каждый раз подгружая определения синтаксиса для xml, заменит определение для *идентификатора* на то, что прописано нами.

## Создание собственной схемы

При наличии «строительных кирпичиков», рассмотренных в предыдущем разделе, мы располагаем достаточным количеством знаний для написания собственного простого файла синтаксиса. Тем не менее для его полной разработки нужно изучить еще много аспектов.

Создадим файл синтаксиса пошагово. Поскольку определения синтаксиса могут быть чрезвычайно сложными, рассмотрим что-нибудь подходящее для легкого понимания, но и достаточно сложное, чтобы показать его мощь.

Обратимся к отрывку из сгенерированного файла на латыни `loremipsum.latin`:

```
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Proin eget
tellus. Suspendisse ac magna at elit pulvinar aliquam. Pellentesque
iaculis augue sit amet massa. Aliquam erat volutpat. Donec et dui at
massa aliquet molestie. Ut vel augue id tellus hendrerit porta. Quisque
condimentum tempor arcu. Aenean pretium suscipit felis. Curabitur semper
eleifend lectus. Praesent vitae sapien. Ut ornare tempus mauris. Quisque
ornare sapien congue tortor.

In dui. Nam adipiscing ligula at lorem. Vestibulum gravida ipsum iaculis
justo. Integer a ipsum ac est cursus gravida. Etiam eu turpis. Nam laoreet
ligula mollis diam. In aliquam semper nisi. Nunc tristique tellus eu
erat. Ut purus. Nulla venenatis pede ac erat.

 ...
```

Создайте новый файл синтаксиса путем генерации файла с соответствующим названием, в нашем случае — `latin`. При этом файл Vim `latin.vim` можно сгенерировать в вашем личном каталоге запуска редактора `$HOME/.vim`. Затем начнем наши определения, создавая ключевые слова с помощью команды `syntax keyword`. Если в качестве ключевых слов выбрать `lorem`, `dolor`, `nulla` и `lectus`, то файл синтаксиса будет торжественно открыт строкой:

```
syntax keyword identifier lorem dolor nulla lectus
```

Пока при редактировании файла `loremipsum.latin` в нем еще нет подсветки синтаксиса. Чтобы она стала автоматической, нужно проделать еще много работы. Пока же активируем синтаксис командой:

```
:set syntax=latin
```

Поскольку каталог `$HOME/.vim` — один из тех, которые перечислены в опции `runtimepath`, текст приобретет вид, показанный на рис. 14.25.

Может, это сложно заметить, но определенные нами ключевые слова на этом снимке темно-серого, а не черного цвета. Это говорит о том, что их цвет отличается от цвета текста (на самом деле, на экране был черный текст и синие ключевые слова).

```
 1 Lorem ipsum dolor sit amet, consectetuer adipis
 2 tellus. Suspendisse ac magna at elit pulvinar a
 3 iaculis augue sit amet massa. Aliquam erat volu
 4 massa aliquet molestie. Ut vel augue id tellus
 5 condimentum tempor arcu. Aenean pretium suscipi
 6 eleifend lectus. Praesent vitae sapien. Ut orna
 7 ornare sapien congue tortor.
 8
 9 In dui. Nam adipiscing ligula at lorem. Vestibu
```

***Рис. 14.25.*** *Файл на латыни с определенными ключевыми словами*

```
 1 Lorem ipsum dolor sit amet, consectetuer adipiscir
 2 tellus. Suspendisse ac magna at elit pulvinar alic
 3 iaculis augue sit amet massa. Aliquam erat volutpa
 4 massa aliquet molestie. Ut vel augue id tellus her
 5 condimentum tempor arcu. Aenean pretium suscipit f
 6 eleifend lectus. Praesent vitae sapien. Ut ornare
 7 ornare sapien congue tortor.
 8
 9 In dui. Nam adipiscing ligula at lorem. Vestibulur
10 justo. Integer a ipsum ac est cursus gravida. Etia
11 ligula mollis diam. In aliquam semper nisi. Nunc t
12 erat. Ut purus. Nulla venenatis pede ac erat.x
```

***Рис. 14.26.*** *Файл на латыни с определенными ключевыми словами при игнорировании регистра*

Наверно, вы заметили, что первое вхождение Lorem не подсвечено. По умолчанию ключевые слова чувствительны к регистру. Добавьте в самый верх файла синтаксиса строку:

```
:syntax case ignore
```

После этого вы увидите, что Lorem включилось как подсвеченное слово.

Перед следующей попыткой сделаем так, чтобы все работало автоматически. После попытки определить тип файла Vim проверяет другие или даже отменяющие определения (их использовать не рекомендуется) в каталоге под названием ftdetect из вашего runtimepath. Следовательно, создайте такой каталог в $HOME/.vim и файл с именем latin.vim, содержащий всего одну строку:

```
au BufRead,BufNewFile *.latin set filetype=latin
```

Эта строка говорит Vim, что все файлы с расширением .latin являются латинскими файлами и, следовательно, Vim должен обработать файл синтаксиса $HOME/.vim/syntax/latin.vim перед выводом этих файлов.

Сейчас при редактировании loremipsum.latin вы увидите рис. 14.26.

Во-первых, обратите внимание, что синтаксис сразу стал активным, так как Vim распознал его тип у файла latin. Сейчас ключевые слова распознаются независимо от их регистра.

Чтобы было интереснее, зададим шаблон match и присвоим его группе Comment. Для определения текста для подсветки шаблон использует регулярное выражение. Например, мы определим все слова, начинающиеся на s и заканчивающиеся на t, как принадлежащие синтаксису Comment (помните, это только пример!). Регулярное выражение примет следующий вид: \<s[^\t ]*t\> (просто поверьте). Также мы определим область и подсветим ее как Number. Области определяются по регулярным выражениям start и end.

Наша область начинается с Suspendisse и заканчивается на sapien\.. Добавим еще один финт. Пусть ключевое слово lectus *лежит* внутри нашей области. Сейчас файл синтаксиса latin.vim имеет вид:

```
syntax case ignore
syntax keyword identifier lorem dolor nulla lectus
syntax keyword identifier lectus contained
syntax match comment /\<s[^\t ]*t\>/
syntax region number start=/Suspendisse/ end=/sapien\./ contains=identifier
```

При редактировании `loremipsum.latin` мы увидим картинку, аналогичную рис. 14.27.

Добавим несколько замечаний, на которые легче обратить внимание, если вы сами проделаете этот пример и посмотрите на результат в цвете:

- Появилась подсветка нового шаблона. В первой строке `sit` подсвечен голубым цветом, так как он удовлетворяет регулярному выражению `match`.
- Появилась подсветка области. Весь фрагмент абзаца, начинающийся с `Suspendisse` и до слова `sapien.`, подсвечен пурпурным цветом (ой!).
- Ключевые слова подсвечиваются так же, как и раньше.
- Внутри подсвеченной области ключевое слово `lectus` все еще подсвечено зеленым цветом, поскольку мы определили группу `identifier` как `contained`, а область — как `contains identifier`.

Данный пример — только начало потока богатых возможностей подсветки синтаксиса. Хотя этот конкретный пример довольно бесполезен, мы надеемся, что он хорошо продемонстрировал свою мощь и воодушевил вас на эксперименты в создании собственных определений синтаксиса.

***Рис. 14.27.*** *Новая подсветка синтаксиса латинского текста*

# Компиляция и поиск ошибок в Vim

Vim не является интегрированной средой разработки (IDE), но старается сделать жизнь программистов немного проще, включив в сеанс редактирования компиляцию и предоставив быстрый и простой способ находить и исправлять ошибки.

Кроме того, редактор предлагает несколько функций для удобства отслеживания и *перемещения* по позициям в файле. Мы рассмотрим простой пример: цикл правка-компиляция-правка с помощью функций,

встроенных в Vim, некоторые из связанных с ними команд и опций, а также функции, делающие работу удобнее. Все это зависит от одного и того же окна Vim – `Quickfix List`.

Начнем с того, что Vim позволяет компилировать файлы с использованием `make` каждый раз при их изменении. Программа использует действия по умолчанию для управления результатами сборки, чтобы вы могли легко переключаться между правкой и компиляцией. Ошибки компиляции появляются в специальном окне `Quickfix List`, где их можно просмотреть, перейти к ним и исправить.

Для этой темы мы возьмем небольшую программу на C, которая генерирует числа Фибоначчи. В правильном и компилируемом виде код выглядит так:

```
# include <stdio.h>

int main(int argc, char *argv[])
  {
  /*
   * arg 1: starting value
   * arg 2: second value
   * arg 3: number of entries to print
   *
  */

  if (argc - 1 != 3)
    {
    printf ("Three command line args: (you used %d)\n", argc);
    printf ("usage: value 1, value 2, number of entries\n");
    return (1);
    }

/* count = how many to print */
int count = atoi(argv[3]);

/* index = which to print */
long int index;

/* first and second passed in on command line */
long int first, second;

/* these get calculated */
long int current, nMinusOne, nMinusTwo;

first = atoi(argv[1]);
second = atoi(argv[2]);
printf("%d fibonacci numbers with starting values: %d, %d\n", count, first,
    second);
printf("=======================================\n");

/* print the first 2 from the starter values */ printf("%d %04d\n", 1, first);
printf("%d %04d ratio (golden?) %.3f\n", 2, second, (double) second/first);

nMinusTwo = first;
nMinusOne = second;
```

```
for (index=1; index<=count; index++)
  {
  current = nMinusTwo + nMinusOne;
  printf("%d %04d ratio (golden?) %.3f\n",
          index,
          current,
          (double) current/nMinusOne);
  nMinusTwo = nMinusOne;
  nMinusOne = current;
  }
}
```

Скомпилируйте программу из Vim (полагаем, что имя файла — `fibonacci.c`) командой:

```
:make fibonacci
```

По умолчанию редактор передает команду `make` на внешнюю оболочку и переводит результаты в специальное окно `Quickfix List`. После компиляции предыдущего кода экран в `Quickfix List` выглядит примерно как на рис. 14.28.

Далее мы изменим в программе некоторые строки, чтобы получилось значительное количество ошибок.

Измените:

```
long int current, nMinusOne, nMinusTwo;
```

на неправильное объявление:

```
longish int current, nMinusOne, nMinusTwo;
```

Измените:

```
nMinusTwo = first;
nMinusOne = second;
```

на неправильно набранные переменные `xfirst` и `xsecond`:

```
nMinusTwo = xfirst;
nMinusOne = xsecond;
```

Измените:

```
printf("%d %04d ratio (golden?) %.3f\n", 2, second, (float) second/first);
```

на следующий код с пропущенными запятыми:

```
printf("%d %04d ratio (golden?) %.3f\n", 2 second (float) second/first);
```

Теперь снова откомпилируйте программу. На рис. 14.29 показано текущее содержимое окна `Quickfix List`.

Строка 1 окна `Quickfix List` показывает выполненную команду компиляции. Если бы ошибок не было, это была бы единственная строка в окне. Но поскольку сейчас есть ошибки, строка 3 начинается со списка ошибок и их контекста.

```
21   /* count = how many to print */
22   int count = atoi(argv[3]);
23
24   /* index = which to print */
25   long int index;
26
27   /* first and second passed in on command line */
fibonacci.c   09/03/2007 01:10:10 PM
  1 || cc     fibonacci.c   -o fibonacci
~
~
~
~
~
~
~
~
~
[Quickfix List][-]   09/03/2007 01:06:58 PM
```

*Рис. 14.28. Окно Quickfix List после чистой компиляции*

```
47     {
48     current = nMinusTwo + nMinusOne;
fibonacci.c   09/03/2007 01:31:48 PM
  1 || cc     fibonacci.c   -o fibonacci
  2 || fibonacci.c: In function `main':
  3 fibonacci.c|31| error: `longish' undeclared (first use in this
  4 fibonacci.c|31| error: (Each undeclared identifier is reported
  5 fibonacci.c|31| error: for each function it appears in.)
  6 fibonacci.c|31| error: parse error before "int"
  7 fibonacci.c|40| error: parse error before "second"
  8 fibonacci.c|42| error: `nMinusTwo' undeclared (first use in th
  9 fibonacci.c|42| error: `xfirst' undeclared (first use in this
```

*Рис. 14.29. Окно Quickfix List после компиляции с ошибками*

Vim перечисляет все ошибки в окне `Quickfix List` и дает доступ к коду, где ошибки отмечены несколькими способами. Обеспечивая удобство в работе, редактор подсвечивает первую ошибку в окне `Quickfix List`. Затем он меняет позицию в файле исходного кода (сделав прокрутку при необходимости) и помещает курсор на начало строки исходного кода, соответствующей ошибке.

Исправив ее, вы сможете перейти на следующую несколькими способами: вводом команды `:cnext` либо поместив курсор на строку с ошибкой в окне `Quickfix List` и нажав ENTER. Опять же при необходимости Vim прокрутит файл исходного кода и поместит курсор на начало «неисправной» строки.

После того как вы внесли изменения и удовлетворены своими исправлениями, можно запускать цикл компиляция-правка заново, используя те же приемы. При наличии стандартной среды разработчика (для компьютеров с UNIX/Linux она почти всегда есть) Vim по умолчанию

поддерживает цикл «правка-компиляция-правка», как описано выше, без дополнительной настройки.

Если по умолчанию Vim использует неправильную программу компиляции, вы можете указать расположение нужных утилит в опциях, после чего все заработает. Подробности, касающиеся сред программирования и компиляторов, лежат за рамками этой книги, но здесь мы представим те опции Vim, которые послужат вам отправной точкой в случае, если придется настраивать свою среду:

`makeprg`

: Опция, содержащая имя программы для `make` или `compile` в среде разработчика.

`:cnext`, `:cprevious`

: Команды, перемещающие курсор на положение *следующей* и *предыдущей* ошибок, указанных в окне `Quickfix List`, соответственно.

`:colder`, `:cnewer`

: Vim запоминает последние 10 списков ошибок. Эти команды загружают в окно `Quickfix List` следующий *более старый* или следующий *более ранний* список ошибок. Каждая команда имеет необязательный аргумент *n* для загрузки *n*-го по давности списка ошибок.

`errorformat`

: Опция, определяющая формат, которому следует Vim при поиске ошибок после компиляции. Встроенная документация программы дает более детальную информацию о том, как нужно задавать эту опцию, но установка по умолчанию работает почти всегда. Если вам нужно подстроить данную опцию, получите подробную информацию путем ввода:

```
:help errorformat
```

## Другие способы использования окна Quickfix List

Vim также позволяет собрать свой список позиций в файлах, где эти позиции указаны в синтаксисе в стиле `grep`. Окно `Quickfix List` возвращает запрошенные результаты в формате, похожем на строки, выдаваемые в описанном выше процессе компиляции.

Эта функция может пригодиться в таких задачах, как рефакторинг. В качестве примера мы написали эту рукопись в формате DocBook, сходном с XML. В какой-то момент написания книги мы сменили обозначение «vim» с `<emphasis>` на `<literal>`. Так, каждое вхождение типа:

```
<emphasis>vim</emphasis>
```

необходимо изменить на:

```
<literal>vim</literal>
```

После выполнения следующей команды:

```
:vimgrep /<emphasis>vim<\/emphasis>/ *.xml
```

окно `Quickfix List` будет содержать информацию, как на рис. 14.30.

```
169 ch09.xml|62 col 39| executables and enjoy all
170 ch09.xml|65 col 24| stripped down <emphasis>vi
171 ch09.xml|67 col 31| <para>Users may need <empha
172 ch09.xml|75 col 32| install full featured <emp
173 ch09.xml|78 col 37| re-compile, and re-install
174 ch09.xml|82 col 20| <para><emphasis>vim</empha
175 ch09.xml|119 col 23| version, <emphasis>vim</e
176 ch09.xml|127 col 53| <para>As mentioned in the
177 ch09.xml|130 col 15| from <emphasis>vim</empha
178 ch09.xml|133 col 10| <emphasis>vim</emphasis>'
179 ch09.xml|137 col 55| understanding <emphasis>v
180 ch09.xml|165 col 27| <para>Thankfully <emphasis
181 ch09.xml|172 col 48| branch out by using TAB c
```

***Рис. 14.30.*** *Окно Quickfix List после команды :vimgrep*

После этого перемещаться по всем вхождениям и менять значения на новые очень легко.

Может показаться, что задача этого примера может решаться проще следующей простой командой:

```
:%s/<emphasis>vim<\/emphasis>/<literal>vim<\/literal>/g
```

Однако помните, что команда `vimgrep` является более общей и обрабатывает несколько файлов. Мы показали работу `vimgrep`, а не единственный способ решить эту задачу. В Vim обычно существует множество путей выполнения задачи.

# Заключительные соображения о написании программ

В этой главе мы рассмотрели множество мощных функций. Уделите немного времени на изучение этих техник, и вы получите огромное преимущество в производительности. Если вы пользуетесь `vi` долгое время, то уже взобрались вверх по крутой кривой обучения. Новый рывок в изучении дополнительных функций Vim достоин второй кривой.

Если вы программист, то мы надеемся, что эта глава показала возможности Vim для решения задач программирования. Мы советуем испытать эти функции и даже расширить Vim под свои потребности. Быть может, вы сами создадите расширения и опубликуете их в сообществе Vim. За работу!

# 15

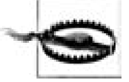
# Другие полезности в Vim

Главы с 10 по 14 охватывают мощные функции Vim и техники, которые, как мы считаем, необходимо знать для эффективного использования редактора. Глава 15 бросает более легкомысленный взгляд на Vim. Здесь рассмотрение некоторых функций, не вошедших в предыдущие разделы, подходы к редактированию и философия Vim, а также некоторые смешные факты о нем (конечно, предыдущие главы тоже были забавными!).

## Редактирование двоичных файлов

Как и `vi`, Vim официально является *текстовым* редактором. Однако в случае крайней необходимости программа позволяет редактировать файлы, содержащие нечитаемые человеком данные.

Зачем вам вообще может понадобиться редактировать двоичный файл? Может, он не зря был сделан двоичным? Ведь двоичные файлы создаются приложением в четко определенном формате, не так ли?



Хотя мы и представляем радостно функцию редактирования двоичных файлов в Vim, мы не останавливаемся на многих серьезных моментах, которые следует иметь в виду. Например, некоторые двоичные файлы содержат цифровые подписи или контрольные суммы, гарантирующие целостность файла. Редактирование этих файлов чревато потерей этой целостности, что может привести их в непригодное состояние. Следовательно, не надо рассматривать эту возможность как поощрение повседневных правок двоичных файлов вручную.

Действительно, двоичные файлы обычно создаются компьютерным или аналоговым процессом и не предназначены для ручного редактирова-

ния. Например, цифровые камеры часто хранят изображения в формате JPEG – сжатом двоичном файле для цифровых фотографий. Хоть они и двоичные, но содержат хорошо определенные разделы (или блоки), где хранится стандартная информация (файлы содержат их, если соответствуют спецификации). Цифровые изображения в формате JPEG содержат мета-информацию об изображении (время, разрешение, настройки камеры, дата и прочее) в зарезервированных блоках, отделенных от собственно сжатых данных изображения. Практическим применением функции редактирования двоичных файлов в Vim может стать исправление поля *год* (*year*) в блоке «создано» в каталоге снимков JPEG для изменения поля «дата создания».

На рис. 15.1 показан сеанс редактирования файла JPEG. Обратите внимание, как расположен курсор в поле даты. Изменив эти поля, можно вручную редактировать информацию об изображении.

***Рис. 15.1.*** *Редактирование двоичного файла JPEG*

Для опытных пользователей, знакомых с конкретным двоичным форматом, Vim может стать чрезвычайно полезным для непосредственного исправления, иначе придется обращаться к скучной повторяющейся работе в других программах.

Существует два способа отредактировать двоичный файл. Можно в командной строке Vim установить опцию `binary`:

```
set binary
```

Или запустить Vim с опцией `-b`.

Чтобы облегчить двоичное редактирование *и* защитить файл от порчи со стороны Vim, программа делает следующее:

- Опции `textwidth` и `wrapmargin` устанавливаются равными 0. Это предотвращает вставку в файл побочных символов новой строки.
- Опции `modeline` и `expandtab` сбрасываются (`nomodeline` и `noexpandtab`). Это предотвращает разворачивание табуляций в пробелы `shiftwidth`

и интерпретирование команд в строке modeline[1], что потенциально может установить опции, приводящие к неожиданным и ненужным побочным эффектам.

В двоичном режиме при переходе от окна к окну или от буфера к буферу будьте внимательны. Vim использует события входа и выхода для задания и установки опций и может ненароком сбросить некоторые из только что перечисленных защитных установок. При редактировании двоичных файлов мы рекомендуем пользоваться однооконным, однобуферным режимом.

## Диграфы: не-ASCII символы

Полагаете, автор «*Мессии*» – George Frideric *Händel* (Георг Фридрих *Гендель*), а не George Frideric *Handel*? Считаете, что ваше *résumé* передает больше, чем *resume*? Используйте диграфы в Vim для ввода специальных символов.

Даже в тексте на английском языке иногда встречаются специальные символы, особенно в ссылках на глобализованный мир. Текстовые файлы на других языках требуют кучу специальных символов.

Vim позволяет ввести специальный символ несколькими способами, два из которых интуитивны и просты. Оба опираются на определение диграфа через префикс (CTRL-K) или используют клавишу BS (Backspace) между двумя символами клавиатуры. (Другие методы больше подходят при вводе символов по их «сырым» числовым значениям, заданным в виде десятичного, шестнадцатеричного или восьмеричного числа. Эти мощные методы сами по себе не обеспечивают простой мнемоники диграфов.)

Термин *диграф* традиционно описывает двухбуквенное сочетание, представляющее единый фонетический звук, например *ph* в «digraph» или «phonetic». Vim перенял понятие «двухбуквенного» сочетания для описания механизма ввода символов со специальными характеристиками, как правило, ударением и другими знаками, например умляутом в *ä*. Эти специальные знаки правильно называются *диакритикой*, или *диакритическими знаками*. Другими словами, Vim использует диграфы для создания диакритики. Мы рады, что смогли это объяснить.

Первый метод ввода диакритики – это трехбуквенное сочетание, содержащее CTRL-K, базовый символ и знак препинания, указывающий на

[1] Особым образом отформатированный комментарий, вставляемый в редактируемый файл, чтобы определить специфичные для него настройки (величину отступа и т. п.). – *Прим. науч. ред.*

акцент или знак, который нужно поставить. Например, чтобы набрать «с» с седилем (ç), введите CTRL-K `c,`. Для создания «а» с грависом (à) введите CTRL-K `a!`.

Греческие буквы можно набрать по соответствующей латинской букве, после которой надо ввести звездочку (например, введите CTRL-K `p*` для строчной π). Русские буквы можно создать по соответствующей латинской, после которой следует поставить знак равно[1] или, в некоторых случаях, знак процента. Используйте CTRL-K `?I` (убедитесь, что `I` прописная) для ввода перевернутого знака вопроса (¿) и CTRL-K `ss`, чтобы ввести немецкий эсцет (ß).

Чтобы использовать второй метод Vim, установите опцию `digraph`:

```
set digraph
```

Теперь создайте специальный символ, введя первый символ двухсимвольной комбинации, затем backspace (BS), а после этого – знак препинания, задающий знак. Так, для ç введите `c`BS`,`, а для à – `a`BS`!`.

Установка опции `digraph` не мешает вам вводить диграфы методом CTRL-K. Полагаться *только* на метод CTRL-K стоит, *если* ваши навыки набора не особо выдающиеся. Иначе вы можете обнаружить, что непреднамеренно вводите диграфы из-за использования Backspace при исправлении ошибок.

Наберите команду `:digraph`, чтобы показать все установки по умолчанию; более подробное описание можно получить с помощью `:help digraph-table`. Рисунок 15.2 показывает частичный список от команды `digraph`.

```
SH ^A   1  SX ^B   2  EX ^C   3  ET ^D   4
VT ^K  11  FF ^L  12  CR ^M  13  SO ^N  14
NK ^U  21  SY ^V  22  EB ^W  23  CN ^X  24
US ^_  31  SP     32  Nb #   35  DO $   36
(! {  123  !! |  124  !) }  125  '? ~  126
NL █  133  SA █  134  ES █  135  HS █  136
S3 █  143  DC █  144  P1 ‘  145  P2 ’  146
GC █  153  SC █  154  CI █  155  ST █  156
Pd £  163  Cu ¤  164  Ye ¥  165  BB ¦  166
-- -  173  Rg ®  174  'm ¯  175  DG °  176
.M ·  183  ',  ¸ 184  1S ¹  185  -o º  186
A' Á  193  A> Â  194  A? Ã  195  A: Ä  196
E: Ë  203  I! Ì  204  I' Í  205  I> Î  206
O? Õ  213  O: Ö  214  *X ×  215  O/ Ø  216
ss ß  223  a! à  224  a' á  225  a> â  226
e' é  233  e> ê  234  e: ë  235  i! ì  236
o' ó  243  o> ô  244  o? õ  245  o: ö  246
```

***Рис. 15.2.*** *Диграфы в Vim*

[1] Разумеется, как русскоговорящий пользователь, вы можете просто переключить раскладку клавиатуры. – *Прим. науч. ред.*

В выводе каждый диграф digraph представлен тремя столбцами. Дисплей немного «комкается», поскольку Vim «втискивает» столько трехстолбцовых комбинаций в каждой строке, сколько позволит экран. Для каждой из групп первый столбец показывает двухсимвольную комбинацию диграфа, второй – сам диграф, а третий – десятичное значение Unicode для диграфа.

Для удобства в табл. 15.1 приведены знаки препинания, используемые в качестве последнего символа в последовательности, для наиболее употребительных ударений и диакритических знаков.

*Таблица 15.1. Как вводить акценты и другие знаки*

| Знак | Символ, вводимый как часть диграфа |
|---|---|
| Акут (fiancé) | Апостроф (') |
| Знак краткости (publică) | Левая скобка ( |
| Гачек (Dubček) | Знак меньше (<) |
| Седиль (français) | Запятая (,) |
| Циркумфлекс (portugus) | Знак больше (>) |
| Гравис (voilà) | Восклицательный знак (!) |
| Макрон (ātmā) | Минус (-) |
| Перечеркивание (Søren) | Косая черта (/) |
| Тильда (señor) | Вопросительный знак (?) |
| Умляут или диерезис (Noël) | Двоеточие (:) |

# Редактирование файлов из других мест

Благодаря гладкой интеграции сетевых протоколов Vim позволяет редактировать файлы на удаленных машинах, как если бы они были на локальном компьютере! Если вы просто укажете URL вместо имени файла, Vim откроет его в окне и запишет изменения на удаленную систему (конечно, при наличии прав доступа). Например, следующая команда редактирует файл .vimrc, принадлежащий пользователю ehannah на системе mozart. Удаленная машина предоставляет безопасный доступ по протоколу SSH на порту 122 (это нестандартный порт, что дает еще больше защиты из-за неизвестности):

```
$ vim scp://ehannah@mozart:122//home/ehannah/.vimrc
```

Поскольку мы редактируем файл в домашнем каталоге ehannah на удаленной машине, то можем сократить URL, используя простое имя файла. Оно рассматривается как путь по отношению к домашнему каталогу пользователя на удаленной системе:

```
$ vim scp://ehannah@mozart:122/.vimrc
```

Рассмотрим URL отдельно и изучим запись URL для конкретного окружения:

*scp:*

Первая часть, до двоеточия, представляет транспортный протокол. В этом примере им является `scp` – протокол копирования файлов, основанный на протоколе SSH (Secure Shell). Последующее двоеточие (`:`) обязательно.

*//*

Предваряет информацию о хосте, которая в большинстве транспортных протоколов имеет вид [*user*@]*hostname*[:*port*].

*ehannah@*

Эта часть необязательна. Для безопасных протоколов, таких как `scp`, определяет, какого пользователя зарегистрировать как удаленного пользователя. Если ее опустить, по умолчанию будет использоваться имя пользователя на локальной машине. На запрос ввести пароль следует ввести пароль пользователя на удаленной машине.

*mozart*

Это символическое имя удаленной машины, которое можно задать и в цифровом виде, например `192.168.1.106`.

*:122*

Эта необязательная часть задает порт, на котором обслуживается протокол. Номер порта отделяется от предшествующего имени хоста двоеточием. Все стандартные протоколы используют вполне определенные порты, так что обычно этот элемент URL можно опустить. В нашем примере порт 122 *не является* стандартным для протокола `scp`. Таким образом, поскольку администратор системы `mozart` разместил сервис на порту 122, такое указание обязательно.

*//home/ehannah/.vimrc*

Это файл на удаленной машине, который мы хотим отредактировать. Начинается с двойной косой черты, поскольку мы задаем абсолютный путь. Относительный путь или просто имя файла требует только одинарной косой черты, отделяющей имя файла от имени хоста. Относительный путь берется относительно домашнего каталога пользователя, под которым был выполнен вход в систему. Так, в нашем примере относительный путь будет браться относительно домашнего каталога `ehannah`, например `/home/ehannah`.

Ниже приведен частичный список поддерживаемых протоколов:

- *ftp:* и *sftp:* (обычный FTP и безопасный FTP).
- *scp:* (безопасное удаленное копирование по SSH).
- *http:* (передача файла по стандартному протоколу браузера).
- *dav:* (сравнительно новый, но популярный открытый стандарт для передачи через веб).
- *rcp:* (удаленное копирование).

Вышесказанного достаточно для получения разрешения на удаленное редактирование, однако сам процесс может быть менее прозрачным, чем редактирование локального файла. То есть из-за промежуточного требования перемещать данные с удаленного хоста вас могут попросить ввести пароль во время работы. Это способно утомить, особенно если вы привыкли периодически записывать файл на диск при редактировании, поскольку каждое из таких «сохранений» прерывается вводом пароля для завершения передачи.

Все транспортные протоколы из вышеприведенного списка позволяют настраивать службу так, чтобы доступ был беспарольным, однако детали этого процесса будут отличаться. За подробностями и настройками каждого конкретного протокола обратитесь к документации службы.

## Переход и смена каталогов

Если вы долго пользуетесь Vim, то, возможно, уже обнаружили, что он позволяет просматривать каталоги и перемещаться по ним с использованием клавиш, применяемых в файлах.

Рассмотрим каталог `ex-050325`, содержащий множество файлов `.c` (это каталог с компилируемыми исходниками для оригинального редактора `vi`). Отредактируйте `ex-050325` с помощью:

```
$ vim ex-050325
```

На рис. 15.3 представлена часть снимка экрана, показывающая примерно то, что вы можете увидеть.

Vim отображает информацию трех типов: вводные комментарии (перед ними стоят знаки равенства), каталоги (после имени каталога стоит косая черта) и файлы. Каждый каталог или файл выводится в своей строке.

Есть много способов использовать эту особенность, однако при помощи стандартных команд перемещения Vim (например, `w` для перехода на следующее слово, `j` или стрелки вниз для перехода на строку ниже и т. д.) и кликов мышью по элементам можно достаточно быстро и интуитивно достичь высокой продуктивности при минимальных усилиях.

Рассмотрим некоторые особенности режима каталогов:

- Когда курсор стоит на имени каталога, нажатие ENTER перемещает внутрь этого каталога.
- Когда курсор стоит на имени файла, нажатие ENTER открывает этот файл для редактирования.

Если нужно оставить окно с каталогами для дальнейшей работы с файлами, нажатием `o` откройте файл под курсором. При этом Vim разделит окно, а файл будет редактироваться в созданном новом окне (то же относится к переходу в другой каталог: если курсор стоял на имени каталога, Vim разделяет окно и «редактирует» каталог, в который вы перешли, в новом окне).

```
" ============================================================================
" Netrw Directory Listing                                        (netrw v98)
"   c:/home/ehannah/downloads/ex-050325
"   Sorted by      name
"   Sort sequence: [\/]$,*,\.bak$,\.o$,\.h$,\.info$,\.swp$,\.obj$
"   Quick Help: <F1>:help  -:go up dir  D:delete  R:rename  s:sort-by  x:exec
" ============================================================================
../
./
catd/
libterm/
libuxre/
Changes
LICENSE
Makefile
README
TODO
ex.1
ex.c
```

***Рис. 15.3.*** *«Редактирование» каталога ex-050325 в Vim*

***Рис. 15.4.*** *Приглашение при переименовании в режиме «редактирования каталогов»*

- Можно удалять и переименовывать файлы и каталоги. Переименование осуществляется нажатием прописной `R`. Возможно, это выглядит несколько странно, но Vim создает приглашение командной строки, где и происходит переименование. Это выглядит примерно как на рис. 15.4.

  Для завершения процесса отредактируйте второй аргумент командной строки.

  Удаление файла работает аналогично: поместите курсор над именем удаляемого файла и нажмите прописную `D`. Vim покажет диалог и попросит подтвердить удаление файла. Как и переименование, процесс происходит в области командной строки.

- Еще одно интересное преимущество редактирования каталогов – быстрый доступ к файлам по функции поиска. Пусть нам нужно отредактировать файл `expreserve.c` из упомянутого выше каталога `ex-050325`. Чтобы быстро перейти и открыть этот файл, найдите его по имени файла или его части:

  ```
  /expreserve.c
  ```

  а после установки курсора на имя этого файла нажмите ENTER или `o`.

Если вы почитаете онлайн-справку по редактированию каталогов, то увидите, что Vim описывает эту функцию как часть средств для редактирования файлов через сетевые протоколы, рассмотренные в предыдущем разделе книги. Мы вынесли изменение каталогов в отдельный раздел, поскольку это важная тема, а в большом количестве деталей сетевых протоколов легко запутаться.

# Резервные копии в Vim

Vim защищает от непреднамеренной потери данных, позволяя создавать резервные копии редактируемых файлов. Если во время правки все пошло не так, то они пригодятся для восстановления предыдущей версии файла.

Резервные копии управляются заданием двух опций: `backup` и `writebackup`. Место и время создания резервных копий определяется четырьмя опциями: `backupskip`, `backupcopy`, `backupdir` и `backupext`.

Если обе опции — `backup` и `writebackup` — выключены (установлены `nobackup` и `nowritebackup`), Vim не делает резервных копий во время сеанса редактирования. Если `backup` включена, программа удаляет старые резервные копии и создает копию для текущего файла. Если `backup` выключена, но включена `writebackup`, Vim создает резервную копию на время сеанса редактирования и удаляет ее после окончания работы.

`Backupdir` — это разделенный запятыми список каталогов, внутри которых Vim создает резервные копии. Например, если вы хотите, чтобы они всегда создавались в системном временном каталоге, установите `backupdir` равным `"C:\TEMP"` для Windows или `"/tmp"` для UNIX и Linux.

Если необходимо, чтобы резервная копия всегда создавалась в текущем каталоге, в качестве каталога резервных копий можно прописать «`.`» (точку). Есть вариант, когда копию сначала пытаются создать в скрытом подкаталоге, если таковой существует, а затем в текущем каталоге. Это можно осуществить, задав значение `backupdir` равным чему-то вроде `"./.mybackups,."` (последняя точка означает текущий каталог для файла). Это очень гибкая опция, позволяющая задавать несколько стратегий определения местоположений резервной копии.

Если вы хотите создавать резервные копии не для всех файлов во время работы, используйте опцию `backupskip`, в которой определите список шаблонов, разделив их запятыми. Vim не будет создавать резервные копии для файлов, удовлетворяющих этим шаблонам. Например, вам не нужно создавать резервные копии файлов, редактируемых в каталогах `/tmp` или `/var/tmp`. Запрет накладывается путем установки значения `"/tmp/*,/var/tmp/*"` в `backupskip`.

По умолчанию Vim создает файл резервной копии с именем оригинала, добавляя ~ (тильду) в конце. Это вполне безопасный суффикс, поскольку имена файлов с этим символом на конце встречаются редко. Опция `backupext` позволяет изменить эту добавку по своему усмотрению. Например, если вы хотите, чтобы резервные копии имели расширение `.bu`, установите `backupext` равным `".bu"`.

Наконец, опция `backupcopy` определяет, *как* создается резервная копия. Мы рекомендуем установить ее равной `"auto"`, чтобы Vim сам вычислял наилучший метод создания копии.

# Создание HTML из текста

У вас возникала потребность представлять свой код или текст другим людям? Вы когда-нибудь пытались изучить код, используя чужую конфигурацию Vim, с которой не смогли разобраться? Попробуйте преобразовать ваш текст или код в HTML и просмотреть его в браузере.

Vim предлагает три способа генерации HTML из имеющегося текста. Все они создают новый буфер с именем, аналогичным оригинальному файлу, но с расширением `.html`. Vim разделяет окно текущего сеанса и показывает HTML-версию файла в новом окне.

`gvim` «*Convert to HTML*»

Это самый дружественный метод, встроенный в графический редактор `gvim` (про который рассказывалось в главе 13). Откройте в `gvim` меню Syntax и выберите «Convert to HTML».

`2html.vim`, *скрипт*

Это скрипт, лежащий в основе пункта меню «Convert to HTML», то есть описанного выше способа. Он вызывается командой:

```
:runtime!syntax/2html.vim
```

Скрипт не понимает диапазонов: обрабатывается весь буфер.

`TOhtml`, *команда*

Это более гибкая команда по сравнению со скриптом `2html.vim`, поскольку здесь можно задавать точный диапазон строк, которые надо преобразовать. Например, чтобы конвертировать строки с 25 по 44 из буфера, введите:

```
:25,44TOhtml
```

Преимуществом использования `gvim` при преобразовании в HTML является то, что GUI позволяет точно распознать цвета и задать правильные директивы HTML. Эти методы работают и в неграфическом контексте, но результаты не будут настолько же точными и могут не принести пользы.

Вы сами решаете, что делать с новым файлом. Vim не будет сохранять его сам, а просто создаст буфер. Мы рекомендуем придерживаться определенной политики для сохранения и синхронизации HTML-версий ваших текстовых файлов. Например, для создания и сохранения файлов HTML можно использовать автокоманды.

Сохраненный файл HTML можно просмотреть в любом браузере. Некоторые из вас могут быть не знакомы со способами открытия файлов в браузере на локальных системах. Это довольно легко: теоретически во всех браузерах есть пункт меню Open File в меню File, после чего появляется диалог, в котором можно перейти в каталог с HTML-файлом. Если вы планируете выполнять эти действия регулярно, рекомендуем создать набор закладок на свои файлы.

# В чем разница?

Различия в разных версиях одного файла часто трудно обнаружить, поэтому утилита для просмотра всех отличий одним разом может сэкономить часы работы. Известная команда UNIX `diff` интегрирована в Vim посредством изощренного интерфейса визуализации, который вызывается командой `vimdiff`.

Для вызова этой функции есть два эквивалентных способа: отдельная команда и опция для Vim:

```
$ vimdiff old_file new_file
$ vim -d old_file new_file
```

Как правило, первый сравниваемый файл представляет старую версию, а второй — более новую, но это только соглашение. На самом деле, всегда можно поменять порядок.

На рис. 15.5 показан пример вывода команды `vimdiff`. Из-за ограниченности места мы уменьшили ширину и выключили опцию `wrap`, чтобы проиллюстрировать отличия.

Хотя рисунок не передает всего эффекта визуализации содержимого (в частности, из-за представления цветов в оттенках серого), на нем показаны ключевые действия:

- В строке 4 слева вы можете заметить темный блок, которого нет справа. Это подсвеченное слово, указывающее на отличия в строках. Аналогично в строке 32 справа содержится подсвеченное слово, которое слева отсутствует.
- На строке 11 с обеих сторон Vim создал 15-строчную *свертку*. Эти 15 строк одинаковы в обоих файлах. Vim свернул их, чтобы на экране было как можно больше информации о «разницах».

- Строки 41–42 слева подсвечены, тогда как на соответствующих позициях справа стоят минусы (-), которые указывают, что справа этих строк нет. Начиная с этого места нумерация строк отличается, поскольку правая часть на две строки меньше. Тем не менее соответствующие друг другу строки из обоих файлов по-прежнему выровнены по горизонтали.

Функция `vimdiff` имеется во всех установках Vim в UNIX-системах, поскольку `diff` входит в стандарт UNIX. Установки редактора не под UNIX должны поставляться со своей версией `diff`. Vim допускает замену команды `diff` на другие, если они тоже вырабатывают стандартный вывод `diff`.

Переменная `diffexpr` определяет выражение, заменяющее собой стандартную команду, используемую `vimdiff` для вычисления различий между двумя файлами. Обычно это выражение реализуется через скрипт, работающий со следующими переменными:

`v:fname_in`

Первый сравниваемый файл.

`v:fname_new`

Второй сравниваемый файл.

`v:fname_out`

Файл для содержимого вывода `diff`.

```
 4 # know what          terminal is on it+|  4 # know what terminal is on it.  The c
 5 # terminal are the lowest common deno+|  5 # terminal are the lowest common denor
 6 # The last one, "other", is like unkn+|  6 # The last one, "other", is like unkno
 7 # that insists that a "real" unknown +|  7 # that insists that a "real" unknown t
 8 #                                      |  8 #
 9                                        |  9
10 dumb:\                                 | 10 dumb:\
11 +-- 15 lines: :am:\-------------------+| 11 +-- 15 lines: :am:\------------------
26         :co#132:li#66:\                | 26         :co#132:li#66:\
27         :bl=^G:cr=^M:do=^J:ff=^L:le=^+| 27         :bl=^G:cr=^M:do=^J:ff=^L:le=^H
28                                        | 28
29 #### ANSI terminals and terminal emul+| 29 #### ANSI terminals and terminal emula
30 #                                      | 30 #
31 # See near the end of this file for d+| 31 # See near the end of this file for de
32 # Don't mess with these entries!  Lot+| 32 # Don't    mess with these entries!  L
33 #                                      | 33 #
34 # This section lists entries in a lea+| 34 # This section lists entries in a leas
35 # if you're in doubt about what `ANSI+| 35 # if you're in doubt about what `ANSI
36 # order and back off from the first t+| 36 # order and back off from the first th
37                                        | 37
38 # (ansi: changed ":pt:" to ":it#8:" -+| 38 # (ansi: changed ":pt:" to ":it#8:" --
39 ansi-mini|any ansi terminal with pess+| 39 ansi-mini|any ansi terminal with pessi
40         :am:bs:\                       | 40         :am:bs:\
41         :co#80:it#8:li#24:\            |    --------------------------------------
42         :ce=\E[K:cl=\E[;H\E[2J:cm=\E[+|    --------------------------------------
43         :ho=\E[H:le=\E[D:nd=\E[C:up=\+| 41         :ho=\E[H:le=\E[D:nd=\E[C:up=\E
44                                        | 42
45 # Color controls corresponding to the+| 43 # Color controls corresponding to the
```

***Рис. 15.5.*** *Результаты vimdiff*

# Отмена отмен

Кроме отмены неограниченного количества правок Vim имеет интересную особенность, называемую *ветвлением* отмен.

Чтобы использовать эту возможность, нужно сначала решить, каким количеством отмен вы хотите управлять. Опция `undolevels` определяет число отменяемых изменений, которые можно делать во время сеанса работы. По умолчанию значение равно 1000. Скорее всего, этого более чем достаточно для большинства пользователей. Чтобы обеспечить совместимость с `vi`, установите `undolevels` равным 0.

```
:set undolevels=0
```

По сути, в `vi` команда отмены `u` переключает между текущим состоянием файла и его самой последней версией. Первая *отмена* переключает в состояние перед последней правкой, а следующая *отмена* восстанавливает отмененное состояние. Vim ведет себя совсем иначе, и, следовательно, эти команды работают по-разному.

Вместо переключения между состояниями с последней правкой повторный вызов отмены в Vim откатывает состояние файла назад к наиболее ранним состояниям до того, которое задает опция `undolevels`. Поскольку команда отмены `u` перемещает только назад, нам потребуется команда для возврата и «повторного применения» изменений. Vim выполняет это действие командой `:redo` или сочетанием клавиш CTRL-R. Комбинация CTRL-R может предваряться числовым аргументом, чтобы восстановить несколько изменений сразу.

При переходе вперед и назад по изменениям командами возврата (CTRL-R) и отмены (`u`) Vim поддерживает карту состояний файла и знает, когда произошла последняя возможная отмена. Когда все отмены исчерпаны, программа меняет состояние «*измененный*» у файла, что позволит выйти без дополнительного суффикса `!` (восклицательного знака). Хотя пользователю от этого мало выгоды, данный факт можно использовать при написании скриптов, где модифицированное состояние файла является важным.

Функций отмены и возврата будет достаточно для работы большинства пользователей. Однако рассмотрим более сложную ситуацию. Что если вы сделали семь исправлений в файле, а отменили только три? Пока все хорошо, нет ничего необычного. А теперь представим, что, отменив три из семи действий, вы проделали изменение, отличное от того, что стоит следующим в коллекции действий Vim. Программа определяет это действие как точку изменения истории в виде новой *ветки*, в которой происходят свои события. По этому пути вы тоже можете перемещаться вперед и назад по хронологии событий, а на точке ветвления – перейти вперед по любому из путей сохраненных изменений.

Для полного ознакомления с перемещением по изменениям как по дереву используйте команду справки Vim:

```
:help usr_32.txt
```

# На чем я остановился?

Большинство редакторов начинают сеанс работы на строке 1, столбец 1. То есть каждый раз при запуске программы загружается файл, и редактирование начинается с первой строки. Если вы долго правите текст, двигаясь по нему вперед, то будет удобнее, если новый сеанс редактирования начнется с места окончания предыдущего сеанса. Vim позволяет это сделать.

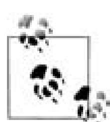
Есть два способа сохранять информацию о сеансе редактирования для будущих целей: опция `viminfo` и команда `mksession`.

## Опция viminfo

Vim использует опцию `viminfo`, чтобы определить, что, как и где сохранять в виде информации о сеансе. Она представляет собой строку с разделенными запятыми параметрами, которые сообщают Vim, какую информацию сохранять и где это делать. Некоторые из подопций `viminfo` определяются следующим образом:

`<n`

: Указывает Vim сохранять строки для каждого регистра, вплоть до максимума из *n* строк.



Если этот параметр не задан, будут сохранены *все* строки. Хотя на первый взгляд это может показаться естественным желанием, подумайте о последствиях при редактировании больших файлов и большом количестве исправлений. Например, если вы обычно редактируете файл из 10000 строк, удаляете все строки (возможно, чтобы воспрепятствовать его быстрому росту, вызванному другой программой), а затем сохраняете файл, то все 10000 строк регистра будут сохранены в файле `viminfo`. Если это проделывается многократно для нескольких файлов, то размер `viminfo` сильно увеличится. Если вы заметите долгие задержки при запуске даже небольших файлов в Vim, то, возможно, это связано с необходимостью обработать файл `viminfo` при старте редактора.

Мы рекомендуем указать какой-нибудь разумный, но пригодный предел. Автор этих строк использует 50.

`/n`

: Количество сохраняемых пунктов в истории поиска. Если параметр не задан, Vim использует значение опции `history`.

`:n`

Максимальное количество сохраняемых команд из истории командной строки. Если параметр не задан, Vim использует значение опции `history`.

`'n`

Максимальное количество файлов, для которых Vim сохраняет информацию. Если вы задаете опцию `viminfo`, этот параметр является обязательным.

В файле `viminfo` сохраняется следующее:

- История командной строки
- История строк поиска
- История строк ввода[1]
- Регистры
- Метки в файлах (то есть сохраняются метки, созданные командой `mx`, после чего при повторном редактировании файла на них можно будет перейти с помощью `'x`)
- Последний поиск и шаблоны подстановки
- Список буферов
- Глобальные переменные

Эта опция очень удобна для поддержания непрерывности в работе. Например, при редактировании большого файла, в котором делаются изменения по шаблону, сохраняется шаблон поиска и положение курсора в файле. Чтобы продолжить редактирование в новом сеансе, нужно лишь ввести `n` для перехода на следующее вхождение шаблона поиска.

## Команда mksession

Командой `mksession` Vim сохраняет всю информацию, касающуюся редактирования в данном сеансе работы. Опция `sessionoptions` содержит строку с запятыми в качестве разделителей, в которой определяются компоненты сеанса для сохранения. Этот способ сохранения информации о сеансе более полный, но и более специфичный, нежели `viminfo`. Он затрагивает файлы, буферы, окна и так далее, поэтому `mksession` сохранит информацию с возможностью полного восстановления сеанса, включая все редактировавшиеся файлы и все установки всех опций, даже размеры окон. В этом и состоит отличие от опции `viminfo`, которая сохраняет информацию по одному файлу.

Чтобы сохранить сеанс таким способом, введите:

```
:mksession [filename]
```

---

[1] Строки, введенные в ответ на вызов функции `input()`; подробности ищите в онлайн-справке. — *Прим. науч. ред.*

где *filename* задает файл, в который будет импортирована информация о сеансе. Vim создает файл скрипта, восстанавливающий сеанс редактирования при его последующем выполнении командой `source` (если имя файла не указано, то по умолчанию присваивается `Session.vim`). Таким образом, если вы сохраняете сеанс командой:

```
:mksession mysession.vim
```

то впоследствии сможете заново воссоздать его, выполнив:

```
:source mysession.vim
```

Ниже мы перечислим компоненты, которые можно сохранять из сеанса с соответствующим параметром для опции `sessionoptions`:

`blank`

Пустые окна.

`buffers`

Скрытые и выгруженные буферы.

`curdir`

Текущий каталог.

`folds`

Созданные вручную и открытые/закрытые свертки, а также локальные опции сверток.

Нет необходимости сохранять другие свертки, кроме созданных вручную. Свертки, созданные автоматически, будут автоматически воссозданы!

`globals`

Глобальные переменные, которые начинаются с прописной буквы и содержат по крайней мере одну строчную.

`help`

Окно справки.

`localoptions`

Опции, определенные локально для окна.

`options`

Опции, заданные с помощью `:set`.

`resize`

Размер окна Vim.

`sesdir`

Каталог, где лежит файл сеанса.

`slash`

Обратные косые черты в файле, замененные на прямые косые черты.

`tabpages`

Все страницы вкладок.

Если это не определить специально в строке `sessionoptions`, то будет сохранен сеанс только для текущей вкладки. В результате можно определять сеанс для одной вкладки либо глобально для всех вкладок.

`unix`

Формат конца строки UNIX.

`winpos`

Положение окна Vim на экране.

`winsize`

Размер окон буферов на экране.

Например, если вы хотите сохранить сеанс, чтобы удержать всю информацию о всех буферах, всех вкладках, глобальных переменных, а также о размере и положении окна, необходимо задать опцию `sessionoptions` следующим образом:

```
:set sessionoptions=buffers,folds,globals,options,resize,winpos
```

# На какой я строке?

Vim допускает строки почти неограниченной длины. Их можно либо переносить на несколько строк экрана, чтобы видеть без горизонтальной прокрутки, либо отображать на экране только начало каждой строки, позволяя ей уйти вправо за пределы экрана.

Если вы предпочитаете, чтобы одной строке экрана соответствовала одна строка текста, отключите опцию `wrap`:

```
set nowrap
```

При `nowrap` Vim отображает столько символов в строке, сколько позволяет ширина экрана. Представьте, что экран – это смотровое отверстие или окно, через которое рассматривается длинная строка. Например, строка из 100 символов содержит на 20 символов больше, чем может вместить экран, состоящий из 80 столбцов. В зависимости от того, какой символ стоит в первом столбце экрана, Vim определяет, какие символы в 100-символьной строке не отображаются. Например, если в первом столбце находится 5-й символ строки, то символы 1–4 расположены за левым краем экрана, поэтому их не видно. Символы 5–84 видны на экране, а оставшиеся символы, с 85 по 100, выходят за правый край экрана, поэтому их также не видно.

В Vim можно управлять отображением строки при перемещении по ней влево и вправо. Минимальное количество символов, на которое Vim смещает строку налево и направо, равно `sidescroll`. Это значение можно задать так:

```
set sidescroll=n
```

где *n* – число прокручиваемых столбцов. Мы рекомендуем установить sidescroll равным 1, поскольку современные ПК обладают мощностью процессора, достаточной для гладкого смещения всего экрана на один символ за один раз. Если ваш экран тормозит и имеет большое время отклика, можно увеличить значение для уменьшения числа перерисовок экрана.

Опция sidescroll определяет *минимальный* сдвиг. Как можно ожидать, Vim перемещает курсор настолько далеко, насколько это требуется для выполнения заданной вами команды перемещения. Например, ввод w переместит курсор на следующее слово в строке. Однако обработка перемещений в Vim немного мудреная.

Если следующее слово видно не полностью (справа), редактор переместит вас на первый символ этого слова, не смещая экран. Следующее выполнение команды w сместит строку влево на такое расстояние, чтобы курсор попал на первый символ следующего слова, но чтобы был виден только этот первый символ.

Подобное поведение можно контролировать опцией sidescrolloff. Она определяет максимальное количество столбцов, содержащихся справа и слева от курсора. Так, если задать sidescrolloff равной 10, Vim сохраняет по крайней мере 10 символов окружения, когда курсор находится близко к краю экрана. Сейчас, когда вы перемещаетесь по строке налево и направо, курсор не подойдет к каждому из краев экрана ближе, чем (в нашем случае) на 10 символов, так как программа начнет смещать текст для просмотра этого контекста. Возможно, это наилучший способ настроить Vim в режиме nowrap.

Редактор предоставляет удобные визуальные подсказки с помощью опции listchar. Она задает способ отображения символов при установленной опции list. В Vim также есть два параметра этой опции, которые определяют, использовать ли дополнительные символы, чтобы показать наличие символов за пределами левого и правого краев экрана в длинной строке. Например:

```
set listchars=extends:>
set listchars+=precedes:<
```

предписывает Vim показывать < в столбце 1, если длинная строка содержит другие символы слева от видимого экрана, а > – в последнем столбце, чтобы показать присутствие других символов справа от видимого экрана. Пример представлен на рис. 15.6.

```
10
11 <ext This is a very long line exceeding width of screen. text >
12
```

***Рис. 15.6.*** *Длинная строка в режиме nowrap*

Напротив, если вам хочется видеть всю строку без прокрутки, опцией `wrap` можно задать перенос строк в Vim.

```
set wrap
```

Теперь строка выглядит, как на рис. 15.7.

```
11 text text This is a very long line exceeding width of screen. t
   ext text more text than a line should ever have unless you're j
   ust doing it for the sake of an example but even in that case i
   t's an awful lot of text for just one line! :-)
12
```

***Рис. 15.7.*** *Длинная строка в режиме wrap*

Очень длинные строки, которые не влезают полностью на экран, представляются единственным символом `@` на первой позиции, пока курсор и файл не будут расположены так, чтобы строка влезла полностью. Строка на рис. 15.7 будет выглядеть, как на рис. 15.8, если расположится внизу экрана.

```
 10
@
@
@
```

***Рис. 15.8.*** *Индикатор длинной строки*

# Сокращения команд и опций Vim

В Vim существует много команд и опций, поэтому мы рекомендуем сначала выучить их названия. Почти все команды и опции (по крайней мере, с длинными именами) имеют связанную с ними короткую форму. Это поможет сэкономить время, *но нужно быть уверенным*, что вы знаете эту аббревиатуру! Автор иногда получал смущающие и неожиданные результаты при использовании неправильной короткой формы команды.

Если вы освоились в любимом подмножестве команд и опций Vim, то использование сокращенных форм сэкономит ваше время. Как правило, для опций редактор пытается взять сокращения типа UNIX, а для команды берет начальную уникальную подстроку ее имени.

Приведем некоторые сокращения для часто используемых команд:

```
n    next
prev previous
q    quit
se   set
w    write
```

И опций:

```
ai   autoindent
bg   background
ff   fileformat
ft   filetype
ic   ignorecase
li   list
nu   number
sc   showcommand (а не showcase)
sm   showmatch
sw   shiftwidth
wm   wrapmargin
```

Сокращенная форма экономит время, если вы достаточно хорошо знаете команды и опции. Однако при написании скриптов и при настройке сеанса командами в файле .vimrc или .gvimrc вы, скорее всего, в перспективе сэкономите время, если будете использовать полные имена команд и опций, т. к. это облегчает чтение и отслеживание файлов и скриптов.

Обратите внимание, что в комплекте файлов скриптов из дистрибутива Vim (syntax, autoindent, colorscheme и т. д.) предпринят противоположный подход, и мы ничего не имеем против него. Тем не менее для удобства использования собственных скриптов мы рекомендуем указывать их полные имена.

## Несколько мелочей (не обязательно для Vim)

Мы предложим несколько приемов. Некоторые из них, имеющиеся как в исходном vi, так и в Vim, стоит запомнить и использовать для облегчения работы:

*Быстрая перестановка*

Довольно часто встречается такая ошибка, как ввод двух символов в неправильном порядке. Поместите курсор на первый из «капризных» знаков и введите xp (удалить символ, вставить символ).

*Другая быстрая перестановка*

У вас есть две строки, которые нужно поменять местами? Поместите курсор на верхнюю и введите ddp (удалить строку, вставить строку после текущей).

*Быстрая справка*

Не забывайте о встроенной справке в Vim. Нажатие клавиши F1 разделит окно и отобразит содержимое онлайновой справки.

*Что это за команду я использовал?*

В общем случае Vim открывает доступ к недавно вызванным командам путем использования клавиш со стрелками в командной строке.

Нажимая стрелки вверх и вниз, вы сможете перемещаться по недавним командам из списка и редактировать их. Независимо от того, правились ли команды в истории Vim или нет, можно вызвать любую из них нажатием на ENTER.

Можно лучше изучить эту особенность, если вызвать встроенное в Vim редактирование истории команд. Это делается нажатием CTRL-F в командной строке. Откроется небольшое «командное» окно (его высота по умолчанию равна 7), в котором вы сможете перемещаться обычными командами перемещения Vim. Также здесь можно выполнять поиск, как в обычном буфере Vim, и редактировать команды.

В окне редактирования команды можно легко отыскать недавнюю команду, при необходимости изменить ее и выполнить нажатием на ENTER. Этот буфер можно записать в отдельный файл, чтобы сохранить историю команд для будущего использования.

*Немного юмора*

Попробуйте ввести

```
:help sure
```

и почитайте, что ответит Vim.

# Другие ресурсы

Здесь мы приведем две ссылки на HTML-копии встроенной справки Vim для двух самых последних основных релизов Vim:

*Vim 6.2*

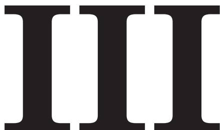
*http://www.vim.org/htmldoc/help.html*

*Vim 7*

*http://vimdoc.sourceforge.net/htmldoc/usr_toc.html*

Кроме того, *http://vimdoc.sourceforge.net/vimfaq.html* – это FAQ (часто задаваемые вопросы) для Vim. В нем не связаны ответы и вопросы, но все они находятся на одной странице. Мы рекомендуем пролистать страницу вниз до ответов и начать чтение оттуда.

Раньше на официальной странице Vim можно было послать советы по программе, однако из-за проблем со спамерами администраторы переместили советы в wiki, где со спамом бороться проще. Адрес страницы в wiki: *http://vim.wikia.com/wiki/Category:Integration*.



# Другие клоны vi

В части III рассматриваются другие популярные клоны `vi`, которые развивались параллельно с Vim. Эта часть содержит следующие главы:

- Глава 16 «nvi: новый vi»
- Глава 17 «elvis»
- Глава 18 «vile: vi Like Emacs»

# 16

# nvi: новый vi

nvi – сокращение от «new vi» (новый vi). Изначально этот редактор был разработан в Калифорнийском университете в Беркли (University of California at Berkeley) (UCB), на родине Berkeley Software Distribution (BSD) UNIX. Он и был использован для написания этой главы.

## Автор и история

Оригинальный vi был создан в UCB в конце 1970-х годов Биллом Джойем (Bill Joy), тогда еще студентом отделения компьютерных наук, который позже стал основателем и вице-президентом Sun Microsystems.

До vi Билл Джой (Bill Joy) сначала написал ex, основываясь на сильно улучшенном шестом издании редактора ed. Первым усовершенствованием стал открытый режим, который он сделал совместно с Чаком Хали (Chuck Haley). Между 1976 и 1979 годами ex превратился в vi. Затем в Беркли пришел Марк Хортон (Mark Horton), добавил макросы «и другие функции»[1] и проделал над программой много работы, после чего редактор стал запускаться на огромном числе терминалов и UNIX-систем. К версии 4.1 BSD (1981) он уже содержал в себе все функции, описанные в части I этой книги.

Несмотря на все перемены, ядром vi был (и остается) изначальный UNIX-редактор ed. Из-за этого его код нельзя было свободно распространять. К началу 1990-х годов, когда шла работа над 4.4 BSD, разработчики BSD захотели иметь версию vi, которая могла бы свободно распространяться в виде исходного кода.

---

[1] Из справочного руководства vi. К сожалению, какие именно функции, не говорится.

Кейт Бостич (Keith Bostic) из UCB начал с elvis 1.8[1] – свободно распространяемой модификации vi – и стал превращать его в «поошибочно совместимую» разновидность изначального редактора. nvi также следует стандартам POSIX Command Language и Utilities Standard (IEEE P1003.1) везде, где это имеет смысл.

Будучи уже не связанным с UCB, Кейт Бостич (Keith Bostic) продолжает распространять nvi. На момент написания книги текущая версия имела номер 1.79.

nvi важен, поскольку это «официальная» берклеевская версия vi. Это часть 4.4 BSDLite II, и именно она используется в популярных разновидностях BSD, таких как NetBSD и FreeBSD.

## Важные аргументы командной строки

В чистом окружении BSD nvi устанавливается под именами ex, vi и view. Обычно все они ссылаются на один и тот же исполняемый файл, и nvi смотрит, как он был вызван, чтобы вести себя соответствующим образом. В нем допускается команда Q режима vi для перехода в режим ex. Вариант view похож на vi, за исключением изначально установленной опции readonly.

nvi поддерживает определенный набор опций командной строки. Приведем наиболее часто употребляемые:

-c *command*

При старте выполняет команду *command*. Это POSIX-версия исторически сложившегося синтаксиса +*command*, однако nvi не ограничен позиционными аргументами (старый синтаксис тоже работает).

-F

Не копировать весь файл при начале редактирования. Это может увеличить скорость работы, но при этом допускается возможность ситуации, когда в редактируемый вами файл кто-то другой тоже вносит изменения.

-r

Восстановить указанные файлы или, если файлы не указаны в командной строке, вывести список всех файлов, которые можно восстановить.

-R

Запуск в режиме «только для чтения» и установка опции readonly.

-s

Вход в пакетный (скриптовый) режим. Предназначено только для ex и для запуска скриптов редактирования. Отключены все подсказки

[1] Хотя от первоначального кода elvis едва ли что-то осталось.

и сообщения, кроме сообщений об ошибках. Это POSIX-версия для исторического аргумента «-»; nvi поддерживает оба варианта.

-S

Запуск с установленной опцией secure, которая запрещает доступ к внешним программам[1].

-t *tag*

Начать редактирование на указанном теге *tag*.

-w *size*

Установка размера внутреннего окна равным *size* строк.

# Онлайн-справка и другая документация

nvi поставляется с довольно полной распечатываемой документацией. Она доступна в виде исходника troff, в формате ASCII и в PostScript и включает следующие документы:

*Справочное руководство vi*

Справочное руководство для nvi, описывающее все опции командной строки nvi, команды, опции и команды ex.

*Man-страницы vi*

Страницы руководства (manpage) для nvi.

*Учебник vi*

Этот документ является вводным курсом в редактирование в vi.

*Справочное руководство для ex*

Представляет собой исходный вариант документации по ex и является немного устаревшим ввиду новых возможностей в nvi.

Кроме того, сюда включены файлы ACSII, в которых задокументированы некоторые внутренние особенности nvi, а также дается список полезных функций и файлы, которые можно использовать в качестве онлайн-справки vi.

Встроенная в nvi онлайн-справка минимальна и содержит две команды: :exusage и :viusage. Они предоставляют однострочные резюме для каждой команды ex и vi. Обычно этого достаточно, чтобы вспомнить, как работает та или иная команда, но не подходит для изучения новых или непонятных функций nvi.

В качестве аргумента для :exusage и :viusage можно задавать команду. В этом случае nvi покажет справку только для нее и выведет одну стро-

[1] Как и на все, что имеет в названии «secure» (безопасный), полностью полагаться на это не стоит. Тем не менее Кейт Бостич (Keith Bostic) утверждает, что опции secure в nvi *можно* доверять.

ку, объясняющую назначение этой команды, и одну, где показано ее использование.

## Инициализация

Если указаны опции -s или «-», nvi игнорирует все инициализации. Иначе программа произведет следующие шаги:

1. Считает и выполнит файл /etc/vi.exrc. Его владельцем должны быть либо вы, либо root.
2. Вычислит значение переменной окружения NEXINIT при ее наличии, иначе использует переменную EXINIT, если она существует. Используется только одна из этих переменных. Игнорируется[1] выполнение $HOME/.nexrc или $HOME/.exrc.
3. Если существует файл $HOME/.nexrc, то он считывается и выполняется. Иначе считывается и выполняется $HOME/.exrc. Обрабатываться будет только один файл.
4. Если установлена опция exrc, то редактор ищет и выполняет файл ./.nexrc, а если такового нет, то./.exrc. Обрабатываться будет только один из этих файлов.

nvi не будет обрабатывать файл, права на запись в который имеет кто-то еще кроме его владельца.

В документации nvi предлагается поместить общие действия инициализации (то есть опции и команды для UNIX vi) в ваш файл .exrc и вызывать команду :source .exrc в вашем файле .nexrc перед или после инициализаций, специфичных для nvi.

## Многооконное редактирование

Чтобы создать новое окно в nvi, используйте версию одной из команд редактирования ex, написанную с прописной буквы: Edit, Fg, Next, Previous, Tag или Visual (как обычно, можно применять аббревиатуры этих команд). Если курсор находится в верхней половине экрана, то новое окно появится в нижней, и наоборот. Переключение на другое окно происходит с помощью CTRL-W:

```
<preface id="VI6-CH-0">
<title>Preface </title>

<para>
Text editing is one of the most common uses of any computer system, and
<command>vi</command> is one of the most useful standard text editors
on your system.
With <command>vi</command> you can create new files, or edit any existing
```

[1] Если переменные NEXINIT или EXINIT определены. – *Прим. науч. ред.*

```
Unix text file.
</para>
ch00.sgm: unmodified: line 1
# Makefile for vi book
#
# Arnold Robbins

CHAPTERS = ch00_6.sgm ch00_5.sgm ch00.sgm ch01.sgm ch02.sgm ch03.sgm \
    ch04.sgm ch05.sgm ch06.sgm ch07.sgm ch08.sgm
APPENDICES = appa.sgm appb.sgm appc.sgm appd.sgm

POSTSCRIPT = ch00_6.ps ch00_5.ps ch00.ps ch01.ps ch02.ps ch03.ps \
    ch04.ps ch05.ps ch06.ps ch07.ps ch08.ps \
Makefile: unmodified: line 1
```

В этом примере показано, как nvi редактирует два файла: ch00.sgm и Makefile. Разделение экрана произошло в результате ввода nvi ch00.sgm и последовавшего за ним :Edit Makefile. Последняя строка каждого файла работает как строка состояния, и именно там выполняются команды с двоеточием для этого окна. Строки состояния показаны инвертированным цветом.

Команды создания окон в режиме ex и их действия описаны в табл. 16.1.

*Таблица 16.1. Команды управления окнами в nvi*

| Команда | Функция |
|---|---|
| bg | Скрывает текущее окно. Его можно вызвать повторно командами fg и Fg. |
| di[splay] b[uffers] | Показывает все буферы, включая именованные, неименованные и цифровые. |
| di[splay] s[creens] | Показывает все имена файлов окон, расположенных в фоне. |
| Edit *filename* | Редактирует файл *filename* в новом окне. |
| Edit /tmp | Создает новое окно, где показывается пустой буфер. /tmp специально интерпретируется как создание нового временного файла. |
| fg *filename* | Возвращает файл *filename* в текущее окно. Предыдущий файл отходит на фон. |
| Fg *filename* | Возвращает файл *filename* в новом окне. Текущее окно разделяется, вместо того чтобы место на экране перераспределилось между всеми открытыми окнами. |
| Next | Редактирует следующий файл из списка аргументов в новом окне. |
| Previous | Редактирует предыдущий файл из списка аргументов в новом окне (в nvi есть соответствующая команда previous, перемещающая пользователя обратно к предыдущему файлу, но в vi ее нет). |
| resize ±*nrows* | Увеличивает или уменьшает размер текущего окна на *nrows* строк. |
| Tag *tagstring* | Редактирует файл, содержащий *tagstring*, в новом окне. |

Команда CTRL-W циклически перемещает между окнами, сверху вниз. Команды :q и ZZ – это выход из текущего окна.

Можно иметь несколько окон, в которых открыт один файл. Изменения, сделанные в одном окне, отражаются в другом, хотя исправления, сделанные в режиме вставки nvi, не будут видны в другом окне, пока вы не завершите правку нажатием ESC. Пользователя не будут приглашать сохранить изменения до вызова команды, закрывающей последнее окно у файла.

# Графические интерфейсы

nvi не имеет версии с графическим пользовательским интерфейсом (GUI).

# Расширенные регулярные выражения

Мы познакомились с расширенными регулярными выражениями еще раньше, в разделе «Расширенные регулярные выражения» на стр. 152. Сейчас мы обобщим метасимволы, предоставляемые nvi. Программа также поддерживает групповые выражения POSIX, [[:alnum:]] и прочие.

Для вызова поиска по расширенным регулярным выражениям нужно выполнить :set extended:

|

Указывает на альтернативу. Левая и правая стороны не обязаны быть одиночными символами.

(...)

Используется для группировки, чтобы допустить применение дополнительных операторов работы с регулярными выражениями.

Когда установлен extended, текст, сгруппированный круглыми скобками, работает как текст, сгруппированный в \(...\) в обычном vi. К собственно тексту можно обратиться в строке замены с помощью \1, \2 и т. д. В этом случае \( представляет собой буквально левую скобку.

+

Соответствует одному или нескольким вхождениям предшествующего регулярного выражения. Им может быть один символ либо группа символов, заключенных в круглые скобки.

?

Соответствует одному или ни одному вхождению предшествующего регулярного выражения.

{...}

Определяет *интервальное выражение*. Оно описывает счетное число повторений. В следующем описании *n* и *m* представляют собой целочисленные константы.

{*n*}

Соответствует *n* вхождениям предшествующего регулярного выражения.

{*n*,}

Соответствует не менее чем *n* вхождениям предшествующего регулярного выражения.

{*n*,*m*}

Соответствует количеству вхождений предшествующего регулярного выражения от *n* до *m*.

Если опция `extended` не установлена, `nvi` предоставляет те же самые возможности, но с использованием `\{` и `\}`.

Как можно ожидать, если `extended` установлена, то для буквального соответствия метасимволам перед ними нужно ставить обратную косую черту.

# Улучшения в редактировании

Этот раздел описывает особенности `nvi`, которые еще более облегчают работу и расширяют возможности простого редактирования текста.

## История и завершение командной строки

`nvi` сохраняет команды `ex` и дает возможность редактировать их для последующего ввода.

Эта способность управляется опцией `cedit`, значение которой – строка.

При вводе первого символа этой строки в командной строке с двоеточием `nvi` открывает новое окно с историей команд, которую можно редактировать. Если на какой-нибудь строке нажать ENTER, программа выполнит ее. Этой опции удобно поставить в соответствие ESC (для его ввода используйте `^V ^[`).

Так как нажатие ENTER запускает выполнение команды, для перемещения вниз по строкам необходимо пользоваться клавишами `j` или ↓.

Помимо редактирования командной строки можно выполнять подстановку в именах файлов. Эта возможность управляется опцией `filec`.

При вводе ее первого символа в командной строке с двоеточием `nvi` рассматривает отделенное пробелами слово перед курсором, как если бы к нему была приписана `*` и производит подстановку имени файла, как в командной строке. Этой опции также удобно поставить в соответствие ESC (для его ввода используйте `^V ^[`).

Если вы поставили этой опции в соответствие тот же символ, что и для опции `cedit`, редактирование командной строки происходит только то-

гда, когда первый введенный символ значения опции cedit одновременно является первым в командной строке с двоеточием.

Документация nvi указывает, что TAB – еще один распространенный выбор для опции filec. Чтобы это заработало, нужно ввести :set filec=\TAB. В любом случае на практике использование ESC хорошо работает для обоих вариантов.

Проще всего установить эти опции в файле .nexrc:

```
set cedit=^[
set filec=^[
```

## Стеки тегов

Теги были описаны ранее в разделе «Стеки тегов» на стр. 157. Стек тегов в nvi – самый простой среди четырех модификаций оригинального редактора. В табл. 16.2 и 16.3 показаны используемые для него команды.

*Таблица 16.2. Команды работы с тегами в nvi*

| **Команда** | **Функция** |
|---|---|
| di[splay] t[ags] | Отобразить стек тегов. |
| ta[g][!] *tagstring* | Редактировать файл, содержащий *tagstring*, как определено в файле tags. ! принуждает nvi переключиться в новый файл, даже если текущий буфер был изменен, но не сохранен. |
| Ta[g][!] *tagstring* | Действует как :tag, только в данном случае файл редактируется в новом окне. |
| tagp[op][!] *tagloc* | Извлекает из стека все теги вплоть до указанного или до последнего использовавшегося тега, если *tagloc* отсутствует. Позицией может быть как имя файла интересующего нас тега, так и номер, присвоенный тегу в стеке. |
| tagt[op][!] | Переходит на самый старый тег в стеке. При этом стек очищается. |

*Таблица 16.3. Команды работы с тегами для командного режима nvi*

| **Команда** | **Функция** |
|---|---|
| ^] | В файле tags ищет положение идентификатора, на котором стоит курсор, и переходит на это место. Текущее положение автоматически заносится в стек тегов. |
| ^T | Возврат к предыдущему положению в стеке тегов, то есть извлечение элемента из стека. |

Можно установить опцию tags как список имен файлов, где nvi будет искать тег. Такой подход позволяет задать упрощенную схему путей поис-

ка. Значение по умолчанию – "tags /var/db/libc.tags /sys/kern/tags", при котором в системе 4.4 BSD сначала идет поиск в текущем каталоге, а затем в файлах библиотеки С и в исходном коде операционной системы.

Опция taglength управляет количеством символов, которое значимо в строке тега. Значение по умолчанию, равное нулю, означает, что нужно использовать все символы.

nvi ведет себя примерно как vi: использует «слово», на котором стоит курсор, начиная с текущей позиции курсора. Если курсор стоит на букве *i* в слове *main*, то nvi ищет идентификатор *in*, а не *main*.

nvi опирается на традиционный формат файла tags. К сожалению, этот формат сильно ограничен. В частности, в нем нет концепции *области видимости* языка программирования, позволяющей одному идентификатору использоваться в разных контекстах и закреплять за собой разные понятия. Эта проблема усугубляется в C++, который допускает явную *перегрузку* имени функции, то есть использование одного и того же имени для разных функций.

nvi обходит ограничения, накладываемые файлом tags, используя совершенно другой механизм[1] – программу cscope. Будучи когда-то проприетарной, сейчас cscope является программой с открытым кодом, доступной в проекте Bell Labs World-Wide Exptools (зайдите на *http://www.bell-labs.com/project/wwexptools/*). Она считывает исходные файлы С и создает базу данных, описывающую программу. В nvi есть команды, осуществляющие запросы и позволяющие обрабатывать результаты. Поскольку cscope доступна не везде, мы не станем рассматривать ее здесь. Подробности про команды nvi изложены в его документации.

Формат файла расширенных тегов, который генерируется программой Exuberant ctags, не привел ни к каким ошибкам в nvi 1.79; однако nvi никак не пользуется преимуществами этого формата.

## Бесконечная отмена

Как правило, в vi команда «точка» (.) работает в качестве команды «повторить еще раз»: она повторяет последнее действие редактирования, будь то удаление, вставка или замена.

nvi возводит эту команду в полноценную «redo», применяя ее даже в тех случаях, когда последней командой была u (или отмена).

Таким образом, чтобы начать серию команд «отмены», просто введите u. Затем для каждой вводимой точки nvi продолжит отменять изменения, постепенно возвращая файл к первоначальному состоянию.

В конце концов первоначальное состояние файла будет достигнуто. После этого нажатие . приведет к системному сигналу (или миганию экра-

[1] Ради полноты картины заметим, что поддержка cscope есть и в Vim. – *Прим. науч. ред.*

на). Теперь можно вводить u, чтобы «отменить отмены», а точку использовать для применения последующих правок.

Заметим, nvi не позволяет снабжать команды u и . числовыми аргументами.

### Строки произвольной длины и двоичные данные

nvi может редактировать файлы со строками произвольной длины и с произвольным числом строк.

Он автоматически поддерживает двоичные данные. Для этого не требуется специальных опций командной строки или команды ex. Для ввода в файл любого 8-разрядного символа наберите ^X, а затем одну или две шестнадцатеричных цифры.

### Инкрементный поиск

Инкрементный поиск в nvi можно включить с помощью :set searchincr.

При наборе курсор перемещается по файлу и всегда попадает на первый символ текста, соответствующего вводу.

### Прокрутка влево-вправо

В nvi прокрутка влево-вправо включается через :set leftright. Значение sidescroll определяет количество символов, на которое nvi смещает экран при прокрутке влево и вправо.

## Помощь программисту

В nvi нет специальных функций для помощи программисту.

## Интересные функции

Из всех клонов nvi является самым минималистичным, так что огромное количество дополнительных функций не поддерживается. Однако существует несколько важных особенностей, про которые стоит упомянуть:

*Поддержка интернационализации*

Большую часть информационных сообщений и предупреждений в nvi можно заменить их переводом на другие языки, используя функцию под названием «каталоги сообщений» (message catalog). nvi реализует эту функцию собственными средствами, используя механизм, который описан в файле catalog/README дистрибутива nvi. Каталоги сообщений (message catalogs) поддерживаются для голландского, английского, французского, немецкого, русского, испанского и шведского языков.

*Произвольные имена для буферов*

Исторически сложилось так, что имена буферов в vi ограничены 26 буквами алфавита. nvi позволяет использовать в качестве имени буфера любые символы.

*Специальная интерпретация /tmp*

Для любой команды ex, требующей имя файла в качестве аргумента, при использовании специального имени /tmp nvi заменит его на имя уникального временного файла.

## Исходный код и поддерживаемые операционные системы

nvi доступен по адресу *http://www.bostic.com/vi*. С этой веб-страницы можно скачать текущую версию[1] и подписаться на почтовую рассылку уведомлений о новых версиях и функциях программы.

Исходный код редактора распространяется свободно. Условия лицензии описаны в файле LICENSE дистрибутива и позволяют распространение в виде исходников и бинарных файлов.

nvi собирается и работает под UNIX. Его также можно собрать под LynxOS 2.4.0 и, возможно, для более поздних версий. Также его можно собрать и запустить под другими POSIX-совместимыми системами, хотя в документации нет списка поддерживаемых операционных платформ.

Компиляция nvi происходит непосредственным образом. Скачайте дистрибутив по ftp, разархивируйте его, запустите программу configure, а после нее — make:

```
$ gzip -d < nvi.tar.gz | tar -xvpf -
...
$ cd nvi-1.79; ./configure
...
$ make
...
```

nvi должен настроиться и собраться без проблем. Для установки выполните make install.

Если вам нужно сообщить об ошибке или проблеме в nvi, обратитесь к Кейту Бостичу (Keith Bostic) на *bostic@bostic.com*.

[1] Графическая версия nvi находится в состоянии разработки; если вам интересно, обратитесь на веб-страницу за контактами.

# 17

# elvis

elvis был написан и поддерживается Стивом Киркендаллом (Steve Kirkendall). Ранняя версия этой программы послужила основой для nvi. Данная глава была изначально написана с использованием elvis.

## Автор и история

Благодарим за помощь Стива Киркендалла и предоставляем ему право самому рассказать об истории редактора:

> «Я начал писать elvis 1.0 после сбоя более ранней модификации vi под названием stevie, из-за чего я потерял несколько часов работы и полностью утратил доверие к этой программе. Также stevie содержал весь буфер редактирования в ОЗУ, что было не совсем удобно в Minix. Так я начал работу над собственным клоном, который хранил бы буфер редактирования в файле. Даже если моя программа закончится аварийно, редактируемый текст можно взять из этого файла.
>
> elvis 2.x рос отдельно от 1.x. Я написал свою вторую модификацию vi, поскольку первая унаследовала слишком много ограничений от настоящего vi и от Minix. Самыми большими изменениями стали поддержка нескольких буферов и многооконность. Ни то, ни другое нельзя было легко внести в 1.x. Мне также хотелось убрать ограничение на длину строки и иметь онлайн-справку, написанную на HTML.»

Что касается имени «elvis», Стив говорит, что была по крайней мере одна причина выбрать это имя – чтобы понять, сколько людей будет об этом спрашивать[1]! Кроме того, достаточно традиционно использовать сочетание «vi» в названиях разновидностей vi.

---

[1] ☺ За восемь лет я был только четвертым! – А.Р.

# Важные аргументы командной строки

`elvis` обычно не устанавливается в системе как `vi`, хотя можно делать и так. Если его вызвать как `ex`, он работает как строковый редактор, а команда `Q` из режима `vi` переключает его в режим `ex`.

В `elvis` есть несколько поддерживаемых ключей командной строки. Наиболее употребляемыми являются:

`-a`

Загружает в отдельном окне каждый файл, перечисленный в командной строке.

`-c` *command*

Выполняет команду *command* при старте и представляет собой POSIX-версию исторически сложившегося синтаксиса +command (он также поддерживается).

`-f` *filename*

Использует для файла сеанса имя *filename* вместо имени по умолчанию. Файлы сеанса обсуждаются позже в этой главе.

`-G` *gui*

Использование заданного интерфейса. Интерфейсом по умолчанию служит `termcap`. Другими вариантами являются `x11`, `windows`, `curses`, `open` и `quit`. Ваша версия `elvis` может быть скомпилирована не со всеми интерфейсами.

`-i`

Начать сеанс в режиме ввода, а не в командном режиме. Может быть, привычнее для новых пользователей.

`-o` *logfile*

Перенаправить стартовые сообщения в файл вместо `stdout/stderr`. Ключ имеет решающую важность для пользователей MS Windows, поскольку эта операционная система отбрасывает все, что записывается в стандартный вывод, а это делает диагностику проблем в конфигурации WinElvis практически невозможной. С помощью `-o` *filename* вы сможете послать диагностическую информацию в файл и просмотреть его позже.

`-r`

Произвести восстановление после сбоя.

`-R`

Начать редактирование каждого файла в режиме «только для чтения».

`-s`

Считать скрипт `ex` из стандартного ввода и выполнить его (синтаксис POSIX). Все скрипты инициализации будут проигнорированы.

-S

Устанавливает опцию security=safer для всего сеанса, а не только для выполнения файлов .exrc. В некоторой степени это увеличит защиту, но слепо доверять данному ключу нельзя.

-SS

Устанавливает опцию security=restricted, еще более «параноидальную», чем security=safer.

-t *tag*

Начать редактирование на указанном теге *tag*.

-V

Вывести более подробную информацию о состоянии. Ключ полезен при диагностике проблем с файлами инициализации.

-?

Выводит все возможные опции.

# Онлайн-справка и другая документация

elvis в этом отношении очень интересен. Онлайн-справка написана на HTML и является всеобъемлющей. Ее можно читать в вашем любимом веб-браузере. В редакторе также есть режим просмотра HTML (о нем позже), позволяющий легко и доступно просматривать справку прямо из elvis.

При чтении файлов HTML для перемещения по разным темам и возврата обратно можно использовать команды работы с тегами (^] и ^T), в результате чего обзор справки станет удобнее. Наши аплодисменты этому нововведению.

Конечно, elvis поставляется со страницами руководства (manpages) UNIX.

# Инициализация

В этом разделе описаны файлы сеанса elvis и расписаны по пунктам шаги, которые он выполняет при инициализации.

## Файл сеанса

В конечном итоге, elvis должен соответствовать стандартам Common Open System Environment (COSE). Это требует способности программы сохранять свое состояние и возвращаться к нему позже.

Для этого редактор сохраняет все свое состояние в файле сеанса. Как правило, он создает его при старте и удаляет при завершении работы, однако в случае сбоя во время работы оставшийся файл сеанса можно использовать для восстановления редактировавшихся файлов.

## Шаги инициализации

1. Инициализировать все жестко закодированные (hardcoded) опции.

Выбрать интерфейс из тех, с которыми elvis был скомпилирован. Он выберет «лучший» из скомпилированных и работоспособных. Например, считается, что интерфейс X11 лучше termcap, но он может оказаться непригодным, если X Window System в настоящий момент не запущена.

Выбранный интерфейс может обрабатывать командную строку и выбирать оттуда опции, предназначенные специально для него.

2. Создать файл сеанса при его отсутствии, иначе считать его (в рамках подготовки к восстановлению).
3. Инициализировать опцию elvispath из переменной окружения ELVISPATH, иначе присвоить ей значение по умолчанию. Обычно таковым является "~/.elvislib:/usr/local/lib/elvis", однако в реальности это значение зависит от порядка настройки и сборки редактора.
4. Искать в elvispath скрипт ex под названием elvis.ini и выполнить его. Имеющийся по умолчанию файл elvis.ini выполняет следующие действия:
   - Выбирает таблицу диграфов на основе текущей операционной системы (диграфы – это способ определения расширенного набора символов ASCII для системы и порядок ввода этого набора).
   - Устанавливает опции, основанные на имени программы (например, режим ex, а не vi).
   - Производит зависимые от системы подстройки, такие как выбор цветов для X11 и добавление меню к интерфейсу.
   - Выбирает имя файла инициализации: либо .exrc для UNIX, либо elvis.rc в не-UNIX системах. Назовем этот файл f.
   - Если существует переменная окружения EXINIT, выполняет ее значение. Иначе выполняется :source ~/f, где f – имя файла, выбранное на предыдущем шаге.
   - Если установлена опция exrc, выполняет команду safely source для f в текущем каталоге.
   - В X11 устанавливает обычный, полужирный и курсивный шрифты, если они еще не установлены.
5. Загрузить командные файлы пред- и постчтения, а также пред- и постзаписи. Затем загрузить файл elvis.msg. Про все эти файлы рассказывается ниже.
6. Загрузить и отобразить первый файл, указанный в командной строке.
7. Если задана опция -a, загрузить и показать остальные файлы, каждый в своем окне.

## Многооконное редактирование

Для создания в elvis нового окна используется команда :split. После этого для редактирования файла нужно применить одну из команд ex, например :e *filename* или :n. Это самый простой способ. Другие более быстрые методы описываются ниже в этой главе. Перемещаться по окнам вперед и назад можно с помощью CTRL-W CTRL-W:

```
<preface id="VI6-CH-0">
<title>Preface </title>

<para>
Text editing is one of the most common uses of any computer system, and
<command>vi</command> is one of the most useful standard text editors
on your system.
With <command>vi</command> you can create new files, or edit any
existing Unix text file.
_____________________________________________________________________
# Makefile for vi book
#
# Arnold Robbins

CHAPTERS = ch00_6.sgm ch00_5.sgm ch00.sgm ch01.sgm ch02.sgm ch03.sgm \
        ch04.sgm ch05.sgm ch06.sgm ch07.sgm ch08.sgm
APPENDICES = appa.sgm appb.sgm appc.sgm appd.sgm

POSTSCRIPT = ch00_6.ps ch00_5.ps ch00.ps ch01.ps ch02.ps ch03.ps \
        ch04.ps ch05.ps ch06.ps ch07.ps ch08.ps \
        appa.ps appb.ps appc.ps appd.ps
```

Разделенный таким образом экран – результат выполнения elvis ch00.sgm, а потом – :split Makefile.

Как и nvi, elvis выделяет собственную строку состояния каждому окну. Уникальность этой программы состоит в том, что для строки состояния он использует подсвеченную строку из знаков подчеркивания, а не инверсные цвета. Команды ex с двоеточием вводятся в строке состояния каждого окна.

Таблица 17.1 описывает команды работы с окнами в режиме ex и их действия.

*Таблица 17.1. Команды управления окнами в elvis*

| **Команда** | **Функция** |
|---|---|
| sp[lit] [*file*] | Создает новое окно и загружает в него файл *file*, если таковой указан. Иначе в новом окне отображается текущий файл. |
| new<br>sne[w] | Создает новый пустой буфер, а затем окно, где показан этот буфер. |

| Команда | Функция |
|---|---|
| sn[ext] [*file...*] | Создается новое окно, где показан следующий файл из списка аргументов. Команда не влияет на текущий файл. |
| sN[ext] | Создает новое окно, где показан *предыдущий* файл из списка аргументов. Команда не влияет на текущий файл. |
| sre[wind][!] | Создает новое окно, в котором показан первый файл из списка аргументов. «Текущим» (с точки зрения команды :next) устанавливается первый файл в списке аргументов. На текущий файл (в смысле редактирования) команда не влияет. |
| sl[ast] | Создает новое окно, где показан последний файл из списка аргументов. Команда не влияет на текущий файл. |
| sta[g][!] *tag* | Создает новое окно, где показан файл и обнаружен указанный тег *tag*. |
| sa[ll] | Создает новые окна для всех файлов, указанных в списке аргументов, но не имеющих своего окна. |
| wi[ndow] [*target*] | Если аргументов нет, выводится список всех окон. Возможные значения *target* перечислены в табл. 17.2. |
| close | Закрывает текущее окно. Команда не действует на буфер, который отображался в окне. Если он был изменен, команда выхода из elvis предотвратит выход, пока вы не сохраните его явно или не откажетесь от изменений в этом буфере. |
| wquit | Записывает буфер обратно в файл и закрывает окно. Файл сохраняется независимо от того, были в нем изменения или нет. |
| qall | Выполняет команду :q для каждого окна. Буферы без окон не затрагиваются. |

Таблица 17.2 описывает аргументы работы с окнами для ex и их значения.

elvis предоставляет несколько команд перемещения между окнами для режима vi. Они приведены в табл. 17.3.

*Таблица 17.2. Аргументы для команды window*

| Аргумент | Значение |
|---|---|
| + | Переход на следующее окно, аналогично ^W k. |
| ++ | Циклический[a] переход на следующее окно, аналогично ^W^W. |
| - | Переход на предыдущее окно, аналогично ^W j. |
| -- | Циклический переход на предыдущее окно. |
| *num* | Переход на окно, у которого windowid=*num*. |
| *buffer-name* | Переход на окно, где редактируется заданный буфер. |

[a] Следующим за последним окном является первое, и наоборот. – *Прим. науч. ред.*

*Таблица 17.3. Команды работы с окнами elvis для режима vi*

| Команда | Функция |
|---|---|
| ^W c | Скрывает буфер и закрывает окно. Эквивалентна команде :close. |
| ^W d | По возможности переключает режим отображения между синтаксическим режимом и другими (html, man, tex). При этом редактирование веб-страниц становится более удобным. Опция определяется для окна. Про режимы отображения рассказывается в разделе «Режимы отображения» на стр. 376. |
| ^W j | Перемещает в следующее окно ниже текущего. |
| ^W k | Перемещает в следующее окно выше текущего. |
| ^W n | Создает новое окно и новый буфер, который показывается в этом окне. Аналогична команде :snew. |
| ^W q | Сохраняет буфер и закрывает окно, равносильна ZZ. |
| ^W s | Разделяет текущее окно. Эквивалентна :split. |
| ^W S | Включает режим wrap. Эта опция определяет, переносятся ли длинные строки или же весь экран прокручивается вправо. Определяется для окна. Она рассматривается подробнее в разделе «Прокрутка влево-вправо» на стр. 370. |
| ^W ] | Создает новое окно, затем ищет тег под курсором. Аналогична команде :stag. |
| [*count*] ^W ^W | Переходит в следующее или *count*-ое по счету окно. |
| ^W + | Увеличивает размер текущего окна (только для интерфейса termcap). |
| ^W - | Уменьшает размер текущего окна (только для интерфейса termcap). |
| ^W \ | Максимально увеличивает размер текущего окна (только для интерфейса termcap). |

# Графические интерфейсы

Снимки экрана и разъяснения для этого раздела были любезно предоставлены Стивом Киркендаллом, за что ему большое спасибо.

Интерфейс X11 для elvis предоставляет полосы прокрутки и поддержку мыши и также позволяет выбрать используемые шрифты. После создания первого окна шрифты уже не изменить. Все они должны быть моноширинными. Обычно это какая-либо вариация гарнитуры Courier.

Интерфейс X11 для elvis поддерживает несколько шрифтов и цветов, мерцание курсора, который меняет свою форму для указания текущего режима редактирования (командный режим или режим вставки), полосы прокрутки, сглаженный текст, графический файл, который можно использовать в качестве фона (его по желанию можно «подцветить»),

пользовательский значок и действия мышью. Мышь можно использовать для выделения текста, вырезания и вставки текста между программами и поиска по тегам. Кроме того, есть настраиваемая панель инструментов, диалоговые окна, строка состояния и флаг `-client`.

Интерфейс GUI для MS Windows также поддерживает фоновый рисунок с помощью той же команды и файла формата XPM, так что в обоих окружениях можно использовать одно и то же изображение.

## Основное окно

Основное окно `elvis` показано на рис. 17.1.

В `elvis` есть отдельное всплывающее диалоговое окно поиска, которое показано на рис. 17.2. Его внешний вид напоминает Motif, хотя на самом деле редактор не использует эти библиотеки.

Параметры командной строки позволяют выбрать четыре различных шрифта, используемых в `elvis`: обычный, курсивный, полужирный и «для управления», то есть шрифт, использующийся для текста панели инструментов и подписей на кнопках. Также можно определять цвет текста и фона, изначальную геометрию окна и обязанность программы запускаться в свернутом виде.

```
Quit Edit Split Save Save as Reload · Prev Next Alt Back · Make Err Search · Normal Hex Syntax Other Display Options · XV Man Help
                /* :help exname */
                section = "elvisex.html";
        }
        else
        {
                /* Can't tell what user is looking for; perhaps the user
                 * doesn't know the syntax of :help ?  Teach them!
                 */
                topic = toCHAR("help");
                section = "elvisex.html";
        }

        /* if help text not found, then give up */
        buf = bufpath(o_elvispath, section, toCHAR(section));
        if (!buf)
        {
                msg(MSG_ERROR, "[s]help not available; couldn't load $1", sectio
n);
                return RESULT_ERROR;
        }

        /* help text uses "html" display mode */
        if (optflags(o_bufdisplay(buf)) & OPT_FREE)
        {
                safefree(o_bufdisplay(buf));
                optflags(o_bufdisplay(buf)) &= ~OPT_FREE;
        }
        o_bufdisplay(buf) = toCHAR("html");

        /* combine section name and topic name to form a tag */
        if (topic)
        {
ex_help                                                    760,16   Command
```

***Рис. 17.1.*** *Графическое окно в elvis*

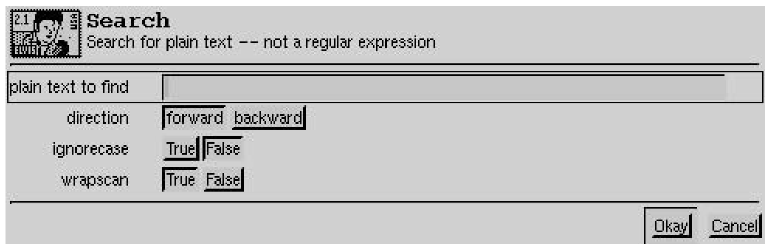



*Рис. 17.2. Диалог поиска в elvis*

Опция `-client` требует, чтобы редактор искал уже запущенный процесс `elvis` и посылал ему сообщение с запросом на редактирование файлов, указанных в командной строке. Таким образом, вы можете использовать скопированный текст и другую информацию совместно – в уже редактируемых в `elvis` и в новых файлах.

Кроме панели инструментов имеется также строка состояния, где показываются сообщения о состоянии и любая другая информация о кнопках панели инструментов.

## Поведение мыши

В своем поведении мышь стремится удержать баланс между `xterm` и полезными в редакторе функциями. Чтобы проделать это корректно, `elvis` различает щелчок и перетаскивание мышью.

Перетаскивание всегда выделяет текст. Перетаскивание с кнопкой 1 выделяет символы, с кнопкой 2 – прямоугольную область, а с кнопкой 3 – строки целиком. (Кнопки 1, 2 и 3 для пользователя-правши соответствуют левой, средней и правой кнопкам. У левшей порядок кнопок меняется на обратный.) Эти операции соответствуют командам `elvis` `v`, `^V` и `V` (они описаны ниже в этой главе). Когда вы отпускаете кнопку в конце перемещения, выделенный текст автоматически копируется в буфер X11, после чего можно вставлять его в другие приложения, например в `xterm`. Текст остается выделенным, и его можно обрабатывать разными командами.

Нажатие кнопки 1 отменит любое незаконченное выделение и переместит курсор на введенный символ. Нажатие кнопки 3 переместит курсор без снятия незаконченного выделения. Это действие используется для расширения незаконченного выделения.

Нажатие кнопки 2 вставляет текст из буфера X11 (например, из `xterm`). Если вы находитесь в командной строке `ex`, текст будет вставлен в командную строку, как при ручном вводе. Если вы находитесь в визуальном режиме или в режиме вставки, текст будет вставлен в буфер редактирования. Неважно, где именно была нажата кнопка мыши, – `elvis` всегда вставляет текст на позиции курсора в файле.

Двойной щелчок по кнопке 1 симулирует нажатие клавиш ^], что приведет к поиску по тегам слова, на котором вы щелкнули. Если elvis отображает HTML-документ, то поиск по тегу проследует по гипертекстовым ссылкам, позволяя щелкать по любому подчеркнутому слову, чтобы увидеть тему с его описанием. Двойной щелчок кнопкой 3 симулирует нажатие клавиш ^T, перенося пользователя в место вызова последнего поиска по тегу.

## Панель инструментов

Интерфейс X11 поддерживает настраиваемую пользователем панель инструментов. По умолчанию панель включена, если только в файле ~/.exrc не содержится команда set notoolbar.

По умолчанию в панели инструментов уже определены несколько кнопок. Для ее перенастройки используется команда :gui.

Она поддерживает несколько команд. В частности, можно перенастроить панель инструментов, чтобы она отвечала вашим вкусам, удаляя одну или все имеющиеся кнопки и управляя промежутками между кнопками или группами кнопок. Приведем простой пример:

```
:gui Make:make
:gui Make " Rebuild the program
:gui Quit:q
:gui Quit?!modified
```

Эти команды добавляют две новые кнопки. В первой строке появляется кнопка под названием Make, по нажатию выполняющая команду :make (про эту команду рассказывается ниже). Во второй строке задается текст описания для кнопки Make, который будет отображаться в строке состояния при нажатии на кнопку. В данном случае символ " — это не начало комментария, а оператор для команды :gui.

Вторая кнопка, названная Quit, создается в третьей строке. Она осуществляет выход из программы. Четвертая строка меняет ее поведение. Если условие (!modified) выполняется, кнопка будет работать как обычно. Однако если условие не выполняется, кнопка проигнорирует любые нажатия мышью и будет «плоской» вместо обычного 3D-вида. Таким образом, если текущий файл изменен, вы не сможете воспользоваться кнопкой Quit для выхода из программы.

Можно создавать всплывающие диалоги, которые появляются при нажатии на кнопки панели. Эти диалоги могут устанавливать значение (или значения) предопределенных переменных (опций). Эту возможность можно протестировать в команде ex, связанной с этой кнопкой. Существует 26 предопределенных переменных с именами от a до z, зарезервированных для пользовательских «программ» такого вида. В следующем примере диалог связывается с новой кнопкой Split:

```
:gui Split"Create a new window, showing a given file
```

```
:gui Split;"File to load:" (file) f = filename
:gui Split:split (f)
```

Первая команда связывает текст описания с кнопкой Split, вторая создает всплывающий диалог, содержащий приглашение «File to load:» для установки опции filename. (file) указывает, что можно ввести любую строку, а для завершения имени файла нажать TAB. f = filename копирует значение filename в f. Наконец, третья команда выполняет :split со значением переменной f, что представляет собой имя нового файла, введенное пользователем.

Эта функция довольно гибкая. За подробностями обратитесь к файлу справки.

## Опции

Интерфейс X11 управляется огромным числом опций. Обычно они устанавливаются в файле .exrc. Есть разнообразные опции и аббревиатуры для задания различных шрифтов, а также для включения и настройки панели инструментов, строки состояния, полос прокрутки и курсора. Также существуют опции, управляющие поведением курсора при переходе между окнами с помощью ^W ^W и возвращающие его в изначальный xterm после выхода из elvis.

Онлайн-документация описывает все опции ex, связанные с X11. Здесь мы приведем только самые интересные из них:

autoiconify

Обычно, когда команда ^W ^W переносит фокус на свернутое (iconified) окно, это окно разворачивается. Если autoiconify имеет значение true, elvis сворачивает старое окно в значок, так что количество открытых окон в редакторе остается постоянным.

blinktime

Значение – число между 0 и 10, определяющее время отображения и скрытия курсора при мерцании в десятых долях секунды. Значение 0 выключает мерцание.

firstx, firsty, stagger

firstx и firsty определяют положение первого окна, создаваемого в elvis. Если они не установлены, положение задается опцией -geometry или менеджером окон. Если значение stagger не равно нулю, то новое окно создается на указанное количество пикселов ниже и правее от текущего. Если значение этой опции равно нулю, то положение будет определяться менеджером окон.

stopshell

Здесь хранится команда, запускающая интерактивную оболочку для команд ex :shell и :stop и для команды ^Z визуального режима. По умолчанию задано xterm &, поэтому интерактивный эмулятор терминала будет запускаться в новом окне.

xscrollbar

Значения left и right помещают полосу прокрутки на соответствующую сторону окна, а none выключает полосы. По умолчанию задано right.

В elvis есть настройка посредством ресурсов X[1]. Их значения можно переопределить переключателями командной строки либо явным выполнением команд :set или :color в скриптах инициализации. Ресурсы elvis перечислены в табл. 17.4.

*Таблица 17.4. Ресурсы X для elvis*

| Класс ресурса (имя – имя класса в нижнем регистре) | Тип | Значение по умолчанию |
|---|---|---|
| Elvis.Toolbar | Boolean | true |
| Elvis.Statusbar | Boolean | true |
| Elvis.Font | Font | fixed |
| Elvis.Geometry | Geometry | 80x34 |
| Elvis.Foreground | Color | black |
| Elvis.Background | Color | gray90 |
| Elvis.MultiClickTimeout | Timeout | 3 |
| Elvis.Control.Font | Font | variable |
| Elvis.Cursor.Foreground | Color | red |
| Elvis.Cursor.Selected | Color | red |
| Elvis.Cursor.BlinkTime | Timeout | 3 |
| Elvis.Tool.Foreground | Color | black |
| Elvis.Tool.Background | Color | gray75 |
| Elvis.Scrollbar.Foreground | Color | gray75 |
| Elvis.Scrollbar.Background | Color | gray60 |
| Elvis.Scrollbar.Width | Number | 11 |
| Elvis.Scrollbar.Repeat | Timeout | 4 |
| Elvis.Scrollbar.Position | Edge | right |

Тип «Timeout» задает время в десятых долях секунды, а «Edge» – положение полосы прокрутки: left, right или none.

[1] X-ресурсы – это способ настроить приложения X11 на основе пар «имя/значение», которые хранятся в памяти X-сервером. Они нечасто используются в современных полновесных окружениях рабочего стола, таких как KDE и GNOME. Тем не менее их можно задавать с помощью команды xrdb.

Например, если база данных ресурсов X содержит строку `elvis.font: 10x20`, то значением шрифта текста по умолчанию станет `10x20`. Именно оно будет использоваться при сбросе опции `normalfont`.

# Расширенные регулярные выражения

Мы уже познакомились с расширенными регулярными выражениями в разделе «Расширенные регулярные выражения» на стр. 152. В `elvis` доступны следующие дополнительные метасимволы:

`\|`

Указывает на альтернативу.

`\(...\)`

Используется для группировки, позволяя применять к регулярным выражениям дополнительные операторы.

`\+`

Соответствует одному или более предшествующим регулярным выражениям.

`\?`

Соответствует одному или ни одному предшествующему регулярному выражению.

`\@`

Соответствует слову под курсором.

`\=`

Указывает, куда следует поместить курсор при обнаружении соответствия. Например, `hel\=lo` поставит курсор на вторую *l* в следующем вхождении *hello*.

`\{...\}`

Описывает интервальное выражение, например `x\{1,3\}`, которое соответствует *x*, *xx* или *xxx*. Групповые выражения POSIX (классы символов и т. д.) также поддерживаются.

# Улучшенные возможности редактирования

В этом разделе описаны функции `elvis`, ускоряющие и упрощающие правку текста.

## История командной строки и автозавершение

Все, что вы вводите в командной строке `ex`, сохраняется в буфер под названием `Elvis ex history`. К нему можно обратиться как к любому буферу `elvis`, но из его непосредственного просмотра в окне не удастся извлечь особой пользы.

Чтобы обратиться к истории, используйте курсорные клавиши для просмотра и редактирования предыдущих команд. Кнопки ↑ и ↓ нужны для перемещения по списку, а ← и → – для движения по командной строке. Вставлять символы можно простым набором, а удалять – клавишей Backspace. Как и при редактировании в обычном буфере vi, Backspace удалит символы, хотя строка не будет обновляться по мере ввода, так что будьте внимательны!

При вводе текста в буфер Elvis ex history (то есть в командную строку с двоеточием) клавишу TAB можно использовать для дополнения имени файла. Вводимое слово рассматривается как его часть, и elvis ищет все подходящие файлы. Если соответствий несколько, он вводит максимально возможное количество символов в именах файлов и выдает звуковой сигнал. Если же для соответствия именам файлов не требуется дополнительных символов, программа выводит все подходящие имена файлов и снова показывает командную строку. Если соответствие всего одно, elvis завершает имя и приписывает символ табуляции. Если никаких соответствий нет, редактор просто вставляет символ табуляции.

Чтобы ввести настоящий символ табуляции, перед ним необходимо нажать ^V. Также можно полностью выключить завершение имен файлов, если установить опцию inputtab в tab для буфера Elvis ex history с помощью следующей команды:

```
:(Elvis ex history)set inputtab=tab
```

## Стеки тегов

Стеки тегов были ранее описаны в разделе «Стеки тегов» на стр. 157. В elvis стеки тегов организованы весьма очевидным образом; они приведены в табл. 17.5 и 17.6.

*Таблица 17.5. Команды работы с тегами в elvis*

| Команда | Функция |
|---|---|
| ta[g][!] [*tagstring*] | Редактирует файл, содержащий *tagstring*, как определено в файле tags. Восклицательный знак ! вынуждает программу перейти на новый файл, даже если текущий буфер был изменен, но не сохранен. |
| stac[k] | Отображает текущий стек тегов. |
| po[p][!] | Извлекает позицию курсора из стека, перемещая курсор в его предыдущее положение. |

В отличие от традиционного vi, при вводе ^] elvis ищет слово, на котором стоит курсор, целиком, а не его часть между положением курсора и концом слова.

В режиме HTML (о нем расскажем чуть позже в разделе «Режимы отображения» на стр. 376) команды работают так же, за исключением того,

что :tag ожидает получить URL, а не имя тега. Адреса URL не зависят от наличия файла tags, так что в режиме HTML файл tags игнорируется. elvis поддерживает URL-адреса типа file:, http: и ftp:. Также он может записывать данные через протокол FTP. Для этого просто укажите URL вместо имени файла. Чтобы получить доступ к собственной учетной записи на FTP-сайте (вместо анонимной), каталог в URL нужно начинать с /~. Программа считает ваш файл ~/.netrc, в котором отыщет нужные имя пользователя и пароль. Здесь режим отображения html хорошо работает (функции работы с сетью также действуют в Windows и OS/2)!

*Таблица 17.6. Команды работы с тегами в командном режиме elvis*

| Команда | Функция |
|---|---|
| ^] | Ищет положение идентификатора, на котором расположен курсор, в файле tags и переходит на эту позицию. Текущее положение автоматически заносится в стек тегов. |
| ^T | Возвращает предыдущее положение из стека тегов, то есть извлекает оттуда один элемент. |

Несколько опций :set влияют на взаимодействие elvis с тегами. Они приведены в табл. 17.7.

*Таблица 17.7. Опции elvis для управления тегами*

| Опция | Функция |
|---|---|
| taglength, tl | Определяет количество значимых символов в теге, по которым будет вестись поиск. Значение по умолчанию, равное нулю, указывает на то, что все символы имеют значение. |
| tags, tagpath | Значение – список каталогов и/или имен файлов, в которых нужно искать файлы tags. elvis ищет файл с именем tags в каждом пункте списка, являющегося каталогом. Элементы списка разделяются двоеточием (или точкой с запятой в DOS/Windows), позволяя использовать пробелы в именах каталогов. По умолчанию в списке содержится только tags, то есть поиск файла с именем tags будет проводиться только в текущем каталоге. Эта опция переопределяется заданием переменной окружения TAGPATH. |
| tagstack | Если опция установлена равной true, elvis помещает каждую позицию в стек тегов. Для выключения используйте :set notagstack. |

elvis поддерживает файлы тегов расширенного формата, который был описан в главе 8. Редактор поставляется со своей собственной версией программы ctags (названной elvtags, чтобы избежать конфликта имен со стандартной версией). Программа генерирует файлы описанного выше расширенного формата. Приведем пример создаваемых им специальных строк !_TAG_:

```
!_TAG_FILE_FORMAT       2       /supported features/
!_TAG_FILE_SORTED       1       /0=unsorted, 1=sorted/
!_TAG_PROGRAM_AUTHOR    Steve Kirkendall        /kirkenda@cs.pdx.edu/
!_TAG_PROGRAM_NAME      Elvis Ctags     //
!_TAG_PROGRAM_URL       ftp://ftp.cs.pdx.edu/pub/elvis/README.html   //
!_TAG_PROGRAM_VERSION   2.1     //
```

В elvis каждое окно имеет свой стек тегов.

Вообще говоря, elvis добавил тегам некоторые новшества. При чтении перегруженных тегов он пытается догадаться, что именно вы ищете, и выводит первым наиболее подходящий вариант. Если вы его отвергаете (повторным вводом ^[ или :tag), он показывает следующий подходящий и т. д. Также программа запоминает атрибуты отвергнутых или принятых тегов и использует их для эвристических догадок при последующем поиске.

Синтаксис команды расширен для поиска по свойствам тегов, отличным от их имен. Здесь можно копать очень глубоко. Почитайте соответствующую главу онлайн-справки, где описывается работа с тегами.

Также существует команда :browse, которая находит все подходящие теги и строит для них таблицу HTML. Из этой таблицы можно переходить по гипертекстовым ссылкам на любой из нужных тегов.

Наконец, существует опция tagprg, которая после установки отменяет встроенный алгоритм поиска по тегам, а вместо него запускает внешнюю программу, производящую поиск.

## Бесконечная отмена

В elvis для получения возможности отменять и повторять изменения различного уровня необходимо установить опцию undolevels на допустимое количество уровней «отмены». Отрицательное значение запрещает любые отмены (не слишком полезный вариант). В документации редактора предупреждается, что каждый уровень отмены использует примерно 6 Кбайт в файле сеанса (файл, в котором описывается ваш сеанс редактирования), поэтому может очень быстро «съесть» дисковое пространство. Рекомендуется устанавливать в undolevels значение не больше и, «возможно, сильно меньше» 100.

Если undolevels присвоено ненулевое значение, то текст вводится обычным путем. Затем каждая последующая u отменит одно действие. Для возобновления («отмена отмены») используется (довольно мнемоническая) команда ^R (Ctrl-R).

По умолчанию в elvis undolevels равен нулю, что делает эту программу похожей на UNIX-vi. Данная опция действует в одном редактируемом буфере. Порядок ее назначения для всех файлов описан в разделе «Шаги инициализации» на стр. 357.

После установки опции undolevels добавление числа к командам u или ^R отменит или восстановит заданное число изменений.

## Строки произвольной длины и двоичные данные

elvis может редактировать файлы с любым количеством строк произвольной длины.

В UNIX elvis не рассматривает двоичный файл как особенный по сравнению с другими файлами. В других системах он использует файл elvis.brf для установки опции binary. Это предотвращает проблемы с различными символами перевода строки. 8-разрядный текст можно ввести, если нажать ^X, а затем две шестнадцатеричные цифры. Использование режима шестнадцатеричного отображения – прекрасный способ просматривать двоичные файлы. Файл elvis.brf и режим hex описываются в разделе «Интересные особенности» на стр. 374.

## Прокрутка влево-вправо

В elvis прокрутка влево-вправо включается опцией :set nowrap. Значение sidescroll определяет количество символов, на которое редактор смещает экран при прокрутке влево или вправо. Команда ^W S переключает значение этой опции.

## Визуальный режим

Программа позволяет выделять области по одному символу, одной строке или прямоугольную область с помощью команд, приведенных в табл. 17.8.

*Таблица 17.8. Клавиши команд блочного режима elvis*

| Команда | Функция |
|---|---|
| v | Начать посимвольное выделение области. |
| V | Начать построчное выделение области. |
| ^V | Начать выделение прямоугольной области. |

elvis подсвечивает текст (используя инверсию) при его выделении. Чтобы выполнить выделение, используйте обычные клавиши перемещения. Так выглядит экран с выделенной прямоугольной областью:

```
The 6th edition of <citetitle>Learning the vi Editor</citetitle>
brings the book into the late 1990&rsquo;s.
In particular, besides the &ldquo;original&rdquo; version of
<command>vi</command> that comes as a standard part of every Unix
system, there are now a number of freely available &ldquo;clones&rdquo;
or work-alike editors.
```

Редактор позволяет проделывать над текстом выделенной области лишь несколько операций. Некоторые из них действуют на всю строку, даже если выделенная область содержит только ее часть (см. табл. 17.9).

*Таблица 17.9. Операции в блочном режиме elvis*

| Команда | Операция |
|---|---|
| c, d, y | Изменение, удаление или копирование текста. Отметим, что лишь d работает точно с прямоугольной областью. |
| <, >, ! | Сдвиг текста влево или вправо, а также его фильтрация. Эти операции действуют на все строки, попадающие в выделенную область. |

После применения команды d, удаляющей область, экран будет выглядеть так:

```
The 6th edition of <citetitle>Learning the vi Editor</citetitle>
brings the 90&rsquo;s.
In particulo;original&rdquo; version of
<command>vi as a standard part of every Unix
system, thef freely available &ldquo;clones&rdquo;
or work-alike editors.
```

# Помощь программисту

В этом разделе описаны возможности elvis, помогающие программистам.

## Ускорение цикла редактирование–компиляция

*Таблица 17.10. Команды разработки программ в elvis*

| **Команда** | **Опция** | **Функция** |
|---|---|---|
| cc[!] [*args*] | ccprg | Запускает компилятор C. Опция полезна при перекомпиляции отдельного файла. |
| mak[e][!] [*args*] | makeprg | Заново компилирует все, что требует перекомпиляции (как правило, с помощью make). |
| er[rlist][!] [*file*] | | Переход на позицию следующей ошибки. |

Команда cc перекомпилирует отдельный файл исходного кода. Она выполняется из командной строки с двоеточием. Например, если вы редактируете файл hello.c и вводите :cc, elvis скомпилирует этот файл.

Если команду :cc снабдить дополнительными аргументами, они будут переданы компилятору C. В этом случае нужно указать *все* аргументы, включая имя файла.

Команда :cc запускает текст из опции ccprg. Ее значение по умолчанию – "cc ($1?$1:$2)". Программа подставляет имя текущего файла вместо $2, а вместо $1 – аргументы, заданные команде :cc. Таким образом,

значение ccprg использует ваши аргументы при их наличии, иначе опция просто передает имя текущего файла системной команде cc (конечно, ccprg можно поменять по своему вкусу).

Аналогично команда :make предназначена для перекомпиляции всего, что требует перекомпиляции. Операция осуществляется путем запуска содержимого опции makeprg, по умолчанию равной "make $1". Таким образом, можно ввести :make hello, чтобы собрать программу hello, а :make – для сборки всего подряд[1].

elvis обрабатывает вывод от компиляции или make и ищет там все, что напоминает имена файлов и номера строк. Когда подходящие кандидатуры найдены, он считает их таковыми и переходит на положение первой ошибки. Команда :errlist перемещает по всем ошибкам по очереди. Редактор отображает текст сообщения об ошибке в строке состояния по мере перемещения на новую строку.

Если команде :errlist задать аргумент *filename*, программа загрузит свежий пакет сообщений об ошибках из этого файла и переместит пользователя на первую ошибку.

Команда * (звездочка) режима vi эквивалентна :errlist. Ее удобнее использовать при большом количестве ошибок.

Наконец, последняя приятная особенность elvis – учет изменений в файле. Программа следит за добавлением или удалением строк пользователем, так что при переходе на следующую ошибку вы окажетесь на нужной строке, даже если ее номер не совпадает с номером в сообщении об ошибке компилятора.

## Подсветка синтаксиса

Для включения подсветки синтаксиса в elvis вызовите команду :display syntax. Она действует на одно окно. (Другие режимы отображения elvis описываются в разделе «Режимы отображения» на стр. 376.)

Внешний вид текста указывается непосредственно с помощью команды :color. Сначала задается тип текста, который нужно выделить цветом. Например, в режиме отображения syntax есть следующие возможности:

comment

Задание вида комментариев.

function

Задание вида идентификаторов – имен функций.

keyword

Определение вида идентификаторов – ключевых слов.

---

[1] Хотя такое поведение стандартно, безусловно, оно полностью определяется конкретным Makefile или аналогичным файлом другой системы сборки. – *Прим. науч. ред.*

`prep`

Задать вид директив препроцессора C и C++.

`string`

Определение вида строковых констант (таких как `"Don't panic!"` в `awk`).

`variable`

Задание вида переменных, полей и прочего.

`other`

Задание отображения всего, что не попало в вышеперечисленные категории, но не должно показываться обычным шрифтом (например, имена типов, определенные с помощью конструкции C `typedef`).

После этого нужно указать гарнитуру шрифта, принимающую одно из следующих значений: `normal`, `bold`, `italic`, `underlined`, `emphasized`, `boxed`, `graphic`, `proportional` или `fixed` (их можно сократить до одной буквы). Затем вы можете указать цвет. Например:

```
:color function bold yellow
```

Описание комментариев, функций и ключевых слов для каждого языка хранится в файле `elvis.syn`. Он поставляется с уже определенными в нем стандартами. В качестве примера дадим спецификацию синтаксиса для `awk`:

```
# Awk. This is actually for Thompson Automation's AWK compiler, which is
# somewhat beefier than the standard AWK interpreter.
language tawk awk
extension .awk
keyword BEGIN BEGINFILE END ENDFILE INIT break continue do else for function
keyword global if in local next return while
comment #
function (
string "
regexp /
useregexp (,~
other allcaps
```

В целом, формат файла говорит сам за себя и полностью описан в онлайн-документации `elvis`.

Причина, по которой редактор связывает шрифты и цвета с различными частями синтаксиса файла, связана с его способностью выводить на печать файлы так, как они отображены на экране. (Почитайте обсуждение команды `:lpr` в разделе «Режимы отображения» на стр. 376.)

На нерастровом дисплее, например в консоли Linux, все шрифты отображаются на тот, который используется драйвером консоли. Из-за этого, например, сложно отличить обычный (`normal`) шрифт от курсивного (`italic`). Однако на некоторых дисплеях (таких как консоль Linux) программа компенсирует это сменой цвета различных шрифтов. Если у вас система GNU/Linux с `elvis`, попробуйте отредактировать обычный

файл C, и вы увидите, что различные участки кода окрасятся в разные цвета. Этот эффект довольно приятен; жалко, что его нельзя воспроизвести на печати.

Цвета синтаксиса в elvis являются атрибутами, относящимися к отдельному окну. Вы можете сменить цвет курсивного шрифта в одном окне, и это не повлияет на цвет курсива в другом. Особенность справедлива даже для окон, в которых отображается один и тот же файл.

Расцветка синтаксиса делает редактирование программ намного интереснее и живее, однако при выборе цветов следует проявлять осторожность!

# Интересные особенности

В elvis есть множество интересных особенностей:

*Поддержка интернационализации*

Как и nvi, в рассматриваемой программе имеется доморощенный способ, допускающий перевод сообщений на другие языки. Файл elvis.msg ищется в elvis path и загружается в буфер с названием Elvis messages.

Сообщения имеют вид *«краткое сообщение*:*длинное сообщение»*. Перед их выводом редактор ищет краткую форму. Если присутствует соответствующая длинная форма, то выводится длинная, иначе используется краткая.

*Режимы отображения*

Возможно, это самая интересная особенность elvis. Для файлов определенного типа программа форматирует на экране содержимое файла, что дает на удивление хороший эффект WYSIWYG. Редактор также может использовать это форматирование для печати буфера на принтерах некоторых типов. Режимам отображения посвящен отдельный подраздел этой главы.

*Командные файлы пред- и пост обработки*

elvis загружает (если находит) четыре файла, которые позволяют настроить его поведение до и после считывания и записи файла. Ниже мы подробнее рассмотрим эту особенность.

*Открытый режим*

elvis – единственная из разновидностей vi, в которой по-настоящему реализован открытый режим vi. (Представьте себе открытый режим в vi, но только с однострочным окном. «Преимуществом» открытого режима является возможность его использования на терминалах, где нельзя перемещать курсор.)

*Безопасность*

Команда :safely устанавливает опцию security для выполнения файлов .exrc не из домашнего каталога или любых других файлов, которым не следует доверять. Если установлено security=safer, то «определенные команды не будут выполняться, использование групповых символов в именах файлов отключается, а также блокируются определенные опции (включая security)». В документации elvis приводятся детали. Однако не следует слепо верить, что редактор обеспечивает полную безопасность.

*Встроенный калькулятор*

Программа расширяет язык команд ex и включает встроенный калькулятор (иногда в документации его называют «вычислителем выражений» (expression evaluator). Он понимает синтаксис языка C и чаще всего используется с командами :if, :calc и :eval. В онлайн-справке есть подробности, а в образцах файлов инициализации elvis – примеры использования.

*Отладчик макросов*

В elvis есть отладчик для макросов vi (команда :map). Его можно использовать при написании сложного ввода или отображений команд.

*Макросы для режима* ex

Команда :alias позволяет определять макросы ex. Она является аналогом команды alias в csh. Например, существует псевдоним :safer для команды :safely, обеспечивающий обратную совместимость с ранними версиями elvis.

*Расширение возможностей команды* %

Здесь визуальная команда % расширена. Она понимает директивы #if, #else и #endif, если используется режим отображения syntax.

*Встроенная проверка орфографии*

В режиме отображения syntax проверка орфографии достаточно продвинута, чтобы проверить файл tags на наличие символов программы, а словарь естественного языка – на комментарии. Посмотрите :help set spell.

*Свертка текста*

Сворачивание текста позволяет скрывать и показывать отдельные фрагменты файла. Это очень удобно при работе со структурированным текстом. Посмотрите :help :fold.

*Подсветка выделенных строк*

По словам Стива, «elvis может добавлять подсветку к выделенным строкам. Посмотрите :help :region. Например, последовательное выполнение команд :load since и :rcssince подсветит строки, измененные после фиксации файла в RCS».

## Режимы отображения

В elvis есть несколько режимов отображения. В зависимости от типа файла программа генерирует его отформатированную версию, что дает эффект WYSIWYG. Режимы отображения приведены в табл. 17.11.

*Таблица 17.11. Режимы отображения в elvis*

| Режим | Внешний вид |
|---|---|
| normal | Форматирования нет; текст отображается так, как он присутствует в файле. |
| syntax | Как normal, но со включенной раскраской синтаксиса. |
| hex | Интерактивный режим с шестнадцатеричным выводом, напоминающий шестнадцатеричный дамп на мейнфрейме. Полезен при редактировании двоичных файлов. |
| html | Простое форматирование веб-страницы. Для перемещения по ссылкам и возврата в исходную точку используются команды работы с тегами. |
| man | Простое форматирование man-страницы. Похоже на вывод nroff -man. |
| tex | Простое подмножество команд форматирования TeX. |

Команда :normal переключает режим отображения с форматированного на режим normal. Для обратного переключения используется :display *mode*. В качестве сокращения команда ^W d переключает режимы отображения для окна.

Из всех режимов отображения «самыми WYSIWYG» по своей природе являются html и man. Онлайн-документация четко определяет подмножество языков разметки, понимаемые elvis.

Редактор использует режим html для вывода онлайн-справки, которая написана на HTML и содержит *множество* перекрестных ссылок.

Ниже дан пример отображения правки одного из HTML-файлов справки в elvis. Экран разделен, и оба окна показывают один и тот же буфер. В нижнем окне используется режим отображения html, а в верхнем — normal:

```
<html><head>
<title>Elvis 2.0 Sessions</title>

</head><body>

<h1>10. SESSIONS, INITIALIZATION, AND RECOVERY</h1>

This section of the manual describes the life-cycle of an
edit session. We begin with the definition of an
```

```
<a href="#SESSION">edit session</a> and
what that means to elvis.
This is followed by sections discussing
<a href="#INIT">initialization</a>
and <a href="#RECOVER">recovery after a crash.</a>


----------------------------------------------------------------------

10.0 SESSIONS, INITIALIZATION, AND RECOVERY
This section of the manual describes the life-cycle of an
edit session. We begin with the definition of an edit
session and what that means to elvis. This is
followed by sections discussing initialization and
recovery after a crash.

10.1 Sessions
```

Режим man тоже интересен, поскольку обычно приходится форматировать и распечатывать страницы руководства (manpage), чтобы убедиться, что верстка выполнена нужным образом. Следующая цитата из онлайн-справки кажется подходящей:

> Troff source was never designed to be interactively edited, and although I did the best I could, attempting to edit in man mode is still a disorienting experience. I suggest you get in the habit of using normal mode when making changes, and man mode to preview the effect of those changes. The ^W d command makes switching between modes a pretty easy thing to do.
>
> (*Перевод*: Исходные тексты troff никогда не предназначались для интерактивного редактирования, и хотя я старался как мог, попытки редактировать файл в режиме man все же могут обескураживать. Я рекомендую выработать привычку вносить изменения в режиме normal и переключаться в режим man для предпросмотра изменений. Команда ^W d делает переключение между режимами весьма простой задачей.)

Интересно отметить, что оба режима, как html, так и man, также работают с командой :color, описанной в разделе «Подсветка синтаксиса» на стр. 372. Это особенно удобно для режима man. Например, по умолчанию в консоли Linux можно отличить только жирный текст (.B) от обычного. Однако при включенной расцветке становятся различимы как жирный (.B), так и курсивный (.I) тексты. Команды управления режимом приведены в табл. 17.12.

*Таблица 17.12. Команды для режимов отображения в elvis*

| Команда | Функция |
|---|---|
| di[splay] [*mode* [*lang*]] | Меняет режим отображения на *mode*. *lang* используется в режиме syntax. |
| no[rmal] | Аналогично :display normal, но требует меньше знаков для ввода. |

Опция bufdisplay связана с каждым окном, и ей можно присвоить одно из значений поддерживаемых режимов отображения. Стандартный файл elvis.arf (см. следующий подраздел) будет смотреть на расширение имени файла буфера и попытается установить дисплей на более интересный режим, нежели normal.

Наконец, elvis также применяет форматирование WYSIWYG для печати содержимого буфера. Команда :lpr форматирует диапазон строк (или весь буфер, по умолчанию) для печати. Можно печатать в файл или перенаправлять в другую команду. По умолчанию программа печатает в канал, выполняющий системную команду спулинга печати.

Команда :lpr управляется несколькими опциями, представленными в табл. 17.13.

*Таблица 17.13. Опции elvis для управления печатью*

| Опция | Функция |
|---|---|
| lptype, lp | Тип принтера. |
| lpconvert, lpcvt | Если опции установлены, преобразует расширенный ASCII Latin-8 в расширенный ASCII PC-8. |
| lpcrlf, lpc | В конце каждой строки принтеру нужен CR/LF. |
| lpout, lpo | Файл или команда для печати. |
| lpcolumns, lpcols | Ширина области печати. |
| lpwrap, lpw | Симуляция переноса строк. |
| lplines, lprows | Длина страницы принтера. |
| lpformfeed, lpff | Посылает запрос на подачу страницы после последней страницы. |
| lpoptions, lpopt | Управляет различными особенностями принтера, имеет смысл только для принтеров PostScript. |
| lpcolor, lpcl | Включает цветную печать в принтерах PostScript и MS Windows. |
| lpcontrast, lpct | Управляет тенями и контрастностью; используется с опцией lpcolor. |

Большая часть опций говорит сама за себя. elvis поддерживает несколько типов принтеров, как показано в табл. 17.14.

Если у вас есть принтер PostScript, во что бы то ни стало установите lptype в ps или ps2. Последний лучше для экономии бумаги, что полезно при печати черновиков.

*Таблица 17.14. Значения для опции lptype*

| Имя | Тип принтера |
|---|---|
| ps | PostScript, одна логическая страница на лист бумаги. |
| ps2 | PostScript, две логических страницы на лист бумаги. |
| epson | Большинство матричных принтеров; графические символы не поддерживаются. |
| pana | Матричные принтеры Panasonic. |
| ibm | Матричные принтеры с графическими символами IBM. |
| hp | Принтеры Hewlett-Packard и большая часть не-PostScript лазерных принтеров. |
| cr | Линейно-матричные принтеры; запечатывание при помощи возврата каретки. |
| bs | Запечатывание при помощи символа backspace. Эта установка наиболее близка к традиционному nroff в UNIX. |
| dumb | Простой ASCII, без управления шрифтами. |

## Командные файлы пред- и постобработки

elvis дает возможность управлять своими действиями при считывании и записи файлов четыре раза: перед и после чтения, а также перед и после записи файла. Это осуществляется путем выполнения содержимого четырех скриптов ex в соответствующие моменты времени. Скрипты ищутся в каталогах, перечисленных в опции elvispath:

elvis.brf

Файл выполняется перед чтением вашего файла (.brf – Before Reading a File). По умолчанию elvis сначала смотрит на расширение файла и пытается определить, двоичный ли он. Если да, включается опция binary, чтобы редактор не преобразовывал символы перевода строки (которые могут быть представлены в файле комбинацией CR/LF) в LF.

elvis.arf

Выполняется после чтения файла (.arf – After Reading a File). Версия по умолчанию проверяет расширение файла, чтобы включить подсветку синтаксиса.

elvis.bwf

Выполняется перед записью файла (.bwf – Before Writing a File), точнее, перед полной заменой первоначального файла содержимым буфера. В версии по умолчанию реализовано копирование первоначального файла в файл с расширением .bak. Чтобы это работало, нужно установить опцию backup.

elvis.awf

Выполняется после записи файла (.awf – After Writing a File). По умолчанию ничего не происходит, хотя это, возможно, хорошее место для добавления ловушки (hook) системы управления версиями.

Использование командных файлов для управления этими действиями обеспечивает значительную гибкость. Вы сможете настроить поведение elvis под свои цели. В других редакторах гораздо сложнее внести в код эти возможности.

Кроме того, elvis поддерживает автокоманды в стиле Vim при помощи :autocmd. Подробнее об этом написано в онлайн-справке.

## Будущее elvis

Стив Киркендалл (Steve Kirkendall) рассказал нам, что им реализованы, но еще не выпущены[1] следующие возможности:

- Интерфейс к GDB (GNU Debugger) для использования при разработке приложений.
- Сохранение состояния, сходное с файлом viminfo у Vim.
- Способность встраивать один синтаксис внутрь другого, например JavaScript в HTML.

## Исходный код и другие операционные системы

Официальная домашняя страница elvis находится по адресу *http://elvis.vi-editor.org/*. Вы можете скачать дистрибутив редактора оттуда либо взять его с ftp по адресу *ftp://ftp.cs.pdx.edu/pub/elvis/elvis-2.2_0.tar.gz*.

Исходный код elvis распространяется свободно. Программа распространяется на условиях perl-лицензии Artistic License. Эти условия описаны в файле doc/license.html дистрибутива.

elvis работает в UNIX, OS/2, MS-DOS и современных версиях MS Windows. Порты для UNIX и Windows имеют графический пользовательский интерфейс, а в версию для MS-DOS включена поддержка мыши.

Компиляция программы происходит непосредственным образом: возьмите дистрибутив с ftp или через веб-браузер, разархивируйте[2] его и запустите программу configure, а затем make:

```
$ gzip -d < elvis-2.2_0.tar.gz | tar -xvpf -
...
```

[1] Актуальная в настоящий момент версия elvis была выпущена в 2003 году. – *Прим. науч. ред.*

[2] Программа untar.c, доступная на ftp-сайте elvis, широко портируема, а также проста для распаковки tar и сжатых gzip-файлов на не-UNIX системах.

```
$ cd elvis-2.2_0; ./configure
...
$ make
...
```

elvis должен настроиться и собраться без проблем. Для установки выполните make install.

Конфигурация по умолчанию приведет к тому, что редактор установит сам себя в стандартные системные каталоги, такие как /usr/bin, /usr/share и прочие. Если вы хотите установить его в /usr/local, воспользуйтесь опцией --prefix скрипта configure.

Если необходимо сообщить об ошибке или проблеме с elvis, обращайтесь к Стиву Киркендаллу (Steve Kirkendall) на *kirkenda@cs.pdx.edu*.

# 18

# vile: vi Like Emacs (vi как Emacs)

vile означает «vi Like Emacs» (vi как Emacs). Он возник как копия MicroEMACS версии 3.9 – доработанной до совпадения «ощущений пальцев» с vi. Разработчиками стали Томас Дики (Thomas Dickey) и Пол Фокс (Paul Fox). За годы, прошедшие с начала девяностых, участие в проекте принимали и другие, в том числе Кевин Буттнер (Kevin Buettner) и Кларк Морган (Clark Morgan).

Текущая версия имеет номер 9.6, а ее релиз состоялся в конце 2007 года[1]. Снимки экрана для этой главы были сделаны в версии 9.5s (предбета релиз). До конца девяностых номера версий возрастали примерно на единицу в год. Учитывая, что с 1999 года версия возрастает на 0.1 за год, когда-нибудь номер станет равен 10.

Эта глава была изначально написана в vile.

## Авторы и история

Пол Фокс (Paul Fox) описывает раннюю историю vile следующим образом:

> «Цель разработки программы всегда немного отличалась от целей, которые ставились перед другими разновидностями исходного редактора. На самом деле, vile *никогда не пытались* сделать «клоном», хотя многие считают его довольно похожим на vi. Я начал заниматься им в 1990 году, так как хотел редактировать несколько файлов в нескольких окнах. К тому времени я уже 10 лет пользовался vi, и исходные коды MicroEMACS приплыли мне в группу новостей на работе, на которой у меня было предостаточно времени. Я начал менять существующую раскладку

[1] На момент подготовки русскоязычного издания книги – 9.8, вышла 7 августа 2010 года. – *Прим. науч. ред.*

> клавиш очевидным образом и сразу столкнулся с проблемой: «Эй! А где же режим вставки?». Пришлось покопаться в коде еще немножко, а потом еще немножко, и в конце концов, в 1991 или 1992 годах, состоялся релиз. (Вскоре в номерах основных версий стал отмечаться год выпуска: 7.3 был третьим релизом в 1997 году.)
>
> Но моей целью всегда оставалось соответствие «ощущений пальцев» (а не внешнего вида экрана) в целом и, из меркантильных побуждений, ощущений пальцев для часто используемых мною команд. ☺ vile имеет любопытный и хорошо работающий режим ex. Возможно, он неважно *выглядит* и не хватает пары команд, лежащих за пределами текущего синтаксического анализатора. По некоторым причинам vile также не может полностью обработать существующие файлы .exrc, поскольку я не считаю это сильно важным. Он может работать с простыми файлами, но для более изощренных требуется доработка. Однако когда вы окунетесь во встроенный язык команд/макросов vile, то быстро забудете о своем волнении об .exrc.»

В декабре 1992 над vile начал работать Томас Дики (Thomas Dickey). Сначала он просто писал «заплатки», а затем перешел к работе над более важными функциями и расширениями, такими как нумерация строк, автозавершение имен файлов и анимация окна списка буферов. Он объясняет: «интеграцию функций друг с другом я считаю более важной, чем реализацию большого числа функций».

В феврале 1994 года над vile стал работать Кевин Буттнер (Kevin Buettner). Изначально он поставлял исправления ошибок для версии под X11 и xvile, а затем усовершенствования, подобные полосам прокрутки. Позднее его работа развилась в поддержку комплектов виджетов Motif, OpenLook и Athena. Как ни странно, виджеты Athena были «не везде доступны без ошибок», и он написал версию, использовавшую голый инструментарий Xt. В конце концов она обогнала по функциональности версию для Athena. Также Кевин реализовал изначальную поддержку GNU Autoconf в vile.

Порт с Win32 GUI, который называется winvile, был запущен в 1997 году и продолжил развиваться с расширениями, включающими OLE-сервер и дополнение к Visual Studio.

В текущей версии vile интерфейс для perl и основные режимы (о них позже) стабильны. Они используются в качестве базы для других функций, таких как сервер (с использованием интерфейса perl) и подсветка синтаксиса в основных режимах. В перспективных планах стоит работа по улучшению поддержки локалей.

# Важные аргументы командной строки

Хотя vile не может быть запущен как vi или ex, его можно вызвать командой view. При этом каждый файл будет рассматриваться как «только для чтения». В отличие от других модификаций vi, в нем нет режима строкового редактора.

Самые важные аргументы командной строки приведены ниже:

-c *command*, + *command*

vile выполнит заданную команду в стиле ex. Можно задавать любое количество опций -c.

-h

Вызывает vile с файлом справки.

-R

Вызывает редактор в режиме «только для чтения». В этом режиме запись файла запрещена (то же самое относится к vile, запущенному как view, или когда в стартовом файле установлен режим readonly).

-t *tag*

Начинает редактирование на указанном теге *tag*. Этой команде эквивалентна опция -T, поэтому ее можно использовать, когда опцию -t «забирает себе» анализатор командной строки X11.

-v

Запускает vile в режиме «просмотра». Отметим, что в нем не разрешено делать изменения в буферах.

-?

vile выводит небольшое описание использования и прекращает работу.

@*cmdfile*

В качестве файла инициализации программа будет использовать заданный файл и игнорировать любой обычный стартовый файл (то есть .vilerc) или переменную окружения (то есть VILEINIT).

Некоторые часто используемые опции устарели, так как в vile реализована POSIX-опция -c (или +):

-g *N*

Редактор начнет правку первого файла на строке с указанным номером. Также задается как +*N*.

-s *pattern*

В первом файле vile выполнит команду поиска по заданному шаблону. Также задается как +/*pattern*.

# Онлайн-справка и другая документация

На настоящий момент vile поставляется с одним (довольно большим) текстовым файлом ASCII – vile.hlp. Команда :help (которую можно сократить до :h) откроет новое окно с этим файлом. После этого можно искать информацию по конкретной теме с помощью стандартных методов поиска vi. Поскольку файл справки – это простой ASCII, его можно распечатать и читать с листа.

В дополнение к файлу справки vile содержит несколько встроенных команд, отображающих информацию о функциях и состоянии редактора. Приведем самые полезные из них:

:show-commands

Создает новое окно, в котором показывается полный список всех команд vile с кратким описанием каждой из них. Информация помещается в собственный буфер, который можно рассматривать как обычный буфер. В частности, его можно записать в отдельный файл, чтобы затем распечатать.

:apropos

Показывает все команды, имена которых содержат данную подстроку. Это намного удобнее простого поиска информации по интересующей теме в файле справки.

:describe-key

Запрашивает у пользователя клавишу или последовательность клавиш, а затем показывает описание этой команды. Например, клавиша x реализует функцию delete-next-character.

:describe-function

Запрашивает у вас имя функции, а затем показывает ее описание. Например, функция delete-next-character удаляет заданное количество символов справа от позиции курсора.

Команды :apropos, :describe-function и :describe-key обеспечивают описательную информацию, выдают синонимы (для удобства, т. к. функция может иметь более одного имени), все остальные клавиши, к которым приписана данная команда (многие сочетания клавиш могут означать одно и то же действие), и показывают, является ли команда «перемещением» или «оператором». Ниже приведен подходящий пример вывода команды :describe-function next-line:

```
"next-line"                    ^J      ^N      j       #-B
  or    "down-arrow"
  or    "down-line"
  or    "forward-line"
  (motion:  move down CNT lines )
```

Команда вывела все четыре имени функции и сочетания клавиш для нее. (Последовательность #-B – это не зависящее от терминала представление vile для «стрелки вверх». Их полный список можно увидеть с помощью :show-key-names.)

Переменной окружения VILE_STARTUP_PATH можно присвоить пути поиска файла справки, отделенные друг от друга двоеточиями[1]. Переменную

---

[1] Порт для Win32 в качестве разделителя использует точку с запятой, а порт для OpenVMS – запятые.

окружения VILE_HELP_FILE можно применять для переопределения имени файла справки (обычно это vile.hlp).

Если в онлайн-справке применить возможность поиска, встроенные команды, описания клавиш и автозавершение команд, то пользоваться справкой становится просто и удобно.

# Инициализация

xvile производит дополнительную инициализацию меню перед выполнением остальных этапов:

1. (Только в xvile). В качестве имени файла описания меню используется значение переменной окружения XVILE_MENU, если таковая установлена. Иначе использует .vilemenu. Этот файл задает меню по умолчанию для интерфейса X11[1].

После этого различные версии vile, xvile и winvile производят одну и ту же двухэтапную инициализацию. Первый этап использует смесь переменных окружения и файлов:

2. Выполняется файл, который был указан в командной строке опцией @*cmdfile*, если таковая присутствовала. Игнорируются остальные шаги инициализации, которые проделывались бы при отсутствии этой опции.
3. Если существует переменная окружения VILEINIT, выполняется ее значение. Иначе производится поиск файла инициализации.
4. Если существует переменная окружения VILE_STARTUP_FILE, она используется как имя стартового файла. Если нет, UNIX использует .vilerc, а другие системы – файл vile.rc.
5. Стартовый файл ищется в текущем каталоге, а затем в домашнем каталоге пользователя. Используется первый найденный.

На втором этапе применяются следующие команды инициализации:

6. Файл, указанный первым в командной строке, загружается в буфер памяти.
7. Выполняются команды, которые были заданы в опциях -c. По умолчанию они будут работать с первым файлом.

Как и другие модификации, vile позволяет помещать общие действия по инициализации (то есть опции и команды для UNIX vi и/или других клонов) в ваш файл .exrc и выполнять из .vilerc команду :source .exrc до или после установок, специфичных для vile.

[1] Меню для winvile не настраиваются. Их функции можно использовать только в Win32.

# Многооконное редактирование

vile несколько отличается от остальных клонов. Он начал свою жизнь как версия MicroEMACS и только потом превратился в редактор с «ощущениями пальцев» vi.

Все версии Emacs хорошо работали с несколькими окнами и несколькими файлами, поэтому vile стал первой программой типа vi, в которой было несколько окон и несколько буферов.

Как и в программах elvis и Vim, команда :split[1] создает новое окно, а затем позволяет использовать команду ex :e *filename*, чтобы редактировать новый файл в новом окне. После этого начинаются отличия. В частности, команды перемещения между окнами для режима vi совсем другие.

На рис. 18.1 показан разделенный экран, появляющийся в результате ввода ch12.xml[2], а потом :split и :e !zcat chapter.xml.gz в vile.

```
<chapter label="12" id="vi6-ch-12">
<!--
  vile:docbookmode
 -->
<title>vile&mdash;vi Like Emacs</title>

<para><emphasis>vile</emphasis> stands for "vi Like Emacs."
It started out as a copy of Version 3.9 of MicroEMACS
that was modified to have the "finger feel" of <emphasis>vi</emphasis>.
Thomas Dickey and Paul Fox are the maintainers.
=== ch12.xml [docbookmode] ======================================(10,48) 0% ==
<?xml version="1.0"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.4//EN"
"http://www.oasis-open.org/docbook/xml/4.4/docbookx.dtd"  [
<!ENTITY latex "!!LATEX!!">
<!ENTITY tex "!!TEX!!">

<!ENTITY ch12 SYSTEM "ch12.xml">
]>
<book fpi="9780596529833">
<title>Learning the vi and vim Editors</title>
<bookinfo>
    [!zcat chapter.xml.] [xmlmode read-only] is !zcat chapter.xml.gz (2,1) 3%
```

***Рис. 18.1.*** *Редактирование этой главы в vile*

Как в Vim, все окна имеют общую нижнюю строку для выполнения команд ex. Каждое окно содержит собственную строку состояния, а строка состояния текущего окна заполнена знаками минус. Кроме того, при

[1] Это артефакт редактора vile, позволяющего сокращать команды. Правильная команда носит имя split-current-window.

[2] Внимательный читатель заметит, что это уже не глава 12. Главы были перенумерованы при разработке седьмого издания.

активном режиме вставки во втором столбце строки состояния появляется I, а если файл был изменен, но не сохранялся, к его имени приписывается [modified].

Также vile похож на Emacs в плане команд и связанных с ними сочетаний клавиш. Они показаны в табл. 18.1. В некоторых случаях два сочетания выполняют одну и ту же операцию, как у команды delete-other-windows.

*Таблица 18.1. Команды управления окнами в vile*

| **Команда** | **Сочетание клавиш** | **Функция** |
|---|---|---|
| delete-other-windows | ^O, ^X 1 | Удаляет все окна, за исключением текущего. |
| delete-window | ^K, ^X 0 | Удаляет текущее окно, если только оно не последнее. |
| edit-file, E, e | ^X e | Открывает данный (для ^X e – находящийся под курсором) файл или существующий буфер в окне. |
| find-file | ^X e | То же, что edit-file. |
| grow-window | V | Увеличивает размер текущего окна на *count* строк. |
| move-next-window-down | ^A ^E | Прокручивает следующее окно вниз (или буфер – вверх) на *count* строк. |
| move-next-window-up | ^A ^Y | Прокручивает следующее окно вверх (или буфер – вниз) на *count* строк. |
| move-window-left | ^X ^L | Прокручивает окно влево на *count* столбцов или на полэкрана, если *count* не указан. |
| move-window-right | ^X ^R | Прокручивает окно вправо на *count* столбцов, или на полэкрана, если *count* не указан. |
| next-window | ^X o | Переходит на следующее окно. |
| position-window | z *where* | Прокрутить окно, поместив текущую позицию курсора в *where*, где *where* принимает одно из следующих значений: центр (., M, m), верх (ENTER, H, t) или низ (-, L, b). |
| previous-window | ^X O | Переходит на предыдущее окно. |
| resize-window | | Устанавливает размер текущего окна равным *count* строк. *count* указывается как префиксный аргумент. |
| restore-window | | Возврат к окну, сохраненному с помощью save-window. |
| save-window | | Пометка окна для последующего возврата в него с помощью restore-window. |
| scroll-next-window-down | ^A ^D | Прокручивает следующее окно вниз на *count* половинок экрана. *count* задается в виде префиксного аргумента. |

| Команда | Сочетание клавиш | Функция |
|---|---|---|
| scroll-next-window-up | ^A ^U | Прокручивает следующее окно вверх на *count* половинок экрана. *count* задается в виде префиксного аргумента. |
| shrink-window | v | Уменьшает размер текущего окна на *count* строк. *count* задается в виде префиксного аргумента. |
| split-current-window | ^X 2 | Разделяет окно пополам; *count* со значением 1 или 2 определяет, какое из новых окон станет текущим. *count* задается в виде префиксного аргумента. |
| view-file | | Открывает данный файл или существующий буфер в окне и помечает его как «только для чтения». |
| set-window | | Открывает существующий буфер в окне. |
| historical-buffer | _ | Выдает список первых пяти буферов. Цифра переместит в указанный буфер, а символ _ – в самый последний редактировавшийся файл. Tab (и back-tab) прокручивают список, что облегчает перемещение по нему при длинных именах буферов. |
| toggle-buffer-list | * | Показывает/скрывает окно, в котором показаны *все* буферы vile. |

# Графические интерфейсы

Снимки экрана и объяснения для этого раздела были представлены Кевином Буттнером (Kevin Buettner), Томасом Дики (Thomas Dickey) и Полом Фоксом (Paul Fox), за что им большое спасибо.

Для vile существуют несколько интерфейсов X11, каждый из которых применяет свой инструментарий, основанный на библиотеке Xt. Есть простая версия «No Toolkit», где инструментарий не используется, однако в ней присутствует своя полоса прокрутки и виджет «доска обсуждения» (bulletin board) для управления геометрией. Есть версии, использующие инструментарии Motif, Athena или OpenLook[1]. Наиболее хорошо поддерживаются версии Motif и Athena, и в них есть поддержка меню.

Существует GUI для Win32, имеющий вариации с поддержкой OLE и Unicode. Обе вариации выглядят одинаково.

[1] Sun Microsystems прекратила поддержку OpenLook перед выходом Solaris 9 в 2002 году.

К счастью, основной интерфейс для всех версий одинаков: имеется одно окно верхнего уровня, которое можно разбить на две или больше панелей (panes). В свою очередь, панели можно использовать для показа нескольких представлений буфера, нескольких буферов или их смеси. На языке `vile` они называются «окнами», но во избежание путаницы при дальнейшем изложении мы продолжим называть их «панелями».

## Сборка xvile

Хотя для `xvile` имеются двоичные пакеты, вы можете захотеть скомпилировать его на платформе без поддержки пакетов.

При сборке `xvile` необходимо решить, какой из инструментариев вы будете использовать. Это можно сделать при настройке `vile` с помощью команды `configure`[1]. Выбирайте из следующих опций:

`--with-screen=`*`value`*

Определяет драйвер терминала. По умолчанию это `tcap` для драйвера `termcap/terminfo`. Среди других значений есть `curses`, `ncurses`, `ncursesw`, `X11`, `OpenLook`, `Motif`, `Athena`, `Xaw`, `Xaw3d`, `neXtaw` и `ansi`.

`--with-x`

Использовать X Window System. Для версии «No Toolkit».

`--with-Xaw-scrollbars`

Использовать полосы прокрутки `Xaw` вместо собственных полос прокрутки `vile`.

`--with-drag-extension`

Использовать расширение перетаскивания/прокрутки с `Xaw`.

## Базовый внешний вид и функциональность xvile

На следующих рисунках представлен интерфейс Motif у `xvile`. Он похож на интерфейс Athena.

На рис. 18.2 показаны три панели:

1. man-страница `vile`, где демонстрируется использование подчеркивания и полужирного начертания.
2. Буфер `misc.c` из `tin`, где есть подсветка синтаксиса (на этот раз с различными цветами для директив препроцессора, операторов, комментариев и ключевых слов, хотя в книге видны только градации серого).
3. Активная трехстрочная панель `[Completions]` (строка состояния у нее темнее) для завершения имен файлов. Она координируется с минибуфером (командной строкой с двоеточием): в первой строке написа-

[1] Скрипт `configure` работает на любой платформе UNIX или сходной с ней. Для сборки в OpenVMS используйте скрипт `vmsbuild.com`. Инструкции по сборке располагаются в самом начале скрипта.

но Completions prefixed by /usr/build/in/tin-1.9.2+/src/m:, а в мини-буфере — Find file: m. Остальная часть мини-буфера содержит имена файлов, удовлетворяющих запросу. Первая строка [Completions] и содержимое меняются по мере завершения пользователем имени файла (и нажатия TAB, что укажет vile показать уменьшенный набор вариантов).

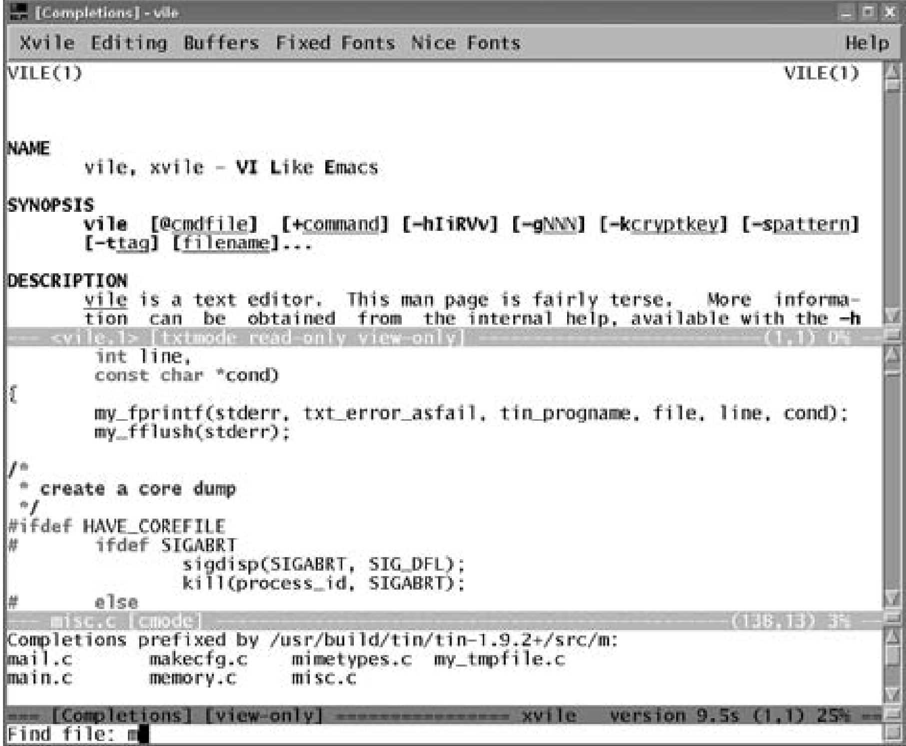



***Рис. 18.2.*** *Графическое окно xvile*

На рис. 18.3 тоже есть три панели:

1. Панель [Help], в которой показана, разумеется, самая важная функция редактора (как выйти из него, не сохраняя изменений). ☺
2. [Buffer List], показывающая, что предыдущим (#) буфером является charset.c. Буфера % (текущего) в списке нет, поскольку в этой копии [Buffer List] отображаются только «видимые» буферы. Если команде * задать аргумент, то она покажет и невидимые. Буферы 0 и 2 – это charset.c и misc.c. Они загружены, а в [Buffer List] показаны их размеры (12425 и 89340). В буфере 1 (<vile.1>) содержится форматированная страница руководства (manpage), сгенерированная макросом. Этому

буферу не соответствует никакой файл[1]. Буфер 3 (`color.c`) не загружен, то есть в первом столбце для него стоит `u`, а его размер – ноль.

3. Буфер `[Completions]` является активным. На этот раз в нем показано завершение по тегам для частичного соответствия *co*, а сообщение *Completions prefixed* не показано, поскольку буфер прокручен пониже, что является побочным действием нажатия TAB: `vile` делает циклическую прокрутку, так что приводятся все варианты, даже если окно маленькое[2].

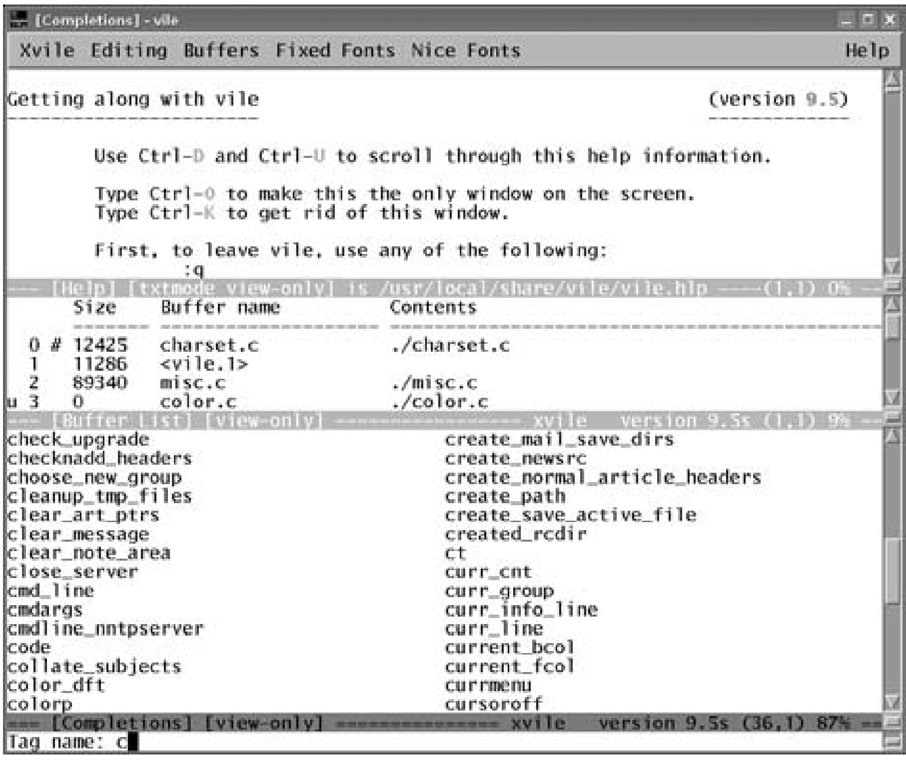



***Рис. 18.3.*** *Буферы и завершение в vile*

Сгенерированные буферы, такие как `[Help]` и `[Buffer List]`, служат «рабочими» (scratch) буферами. Если их свернуть, то они закрываются, а их

---

[1] Угловые скобки вокруг имени `<vile.1>` используются по соглашению, чтобы избежать конфликта имен, поскольку иметь два буфера с одним и тем же именем запрещено.

[2] Размер буфера `[Completions]` устанавливается автоматически. Никогда не показывается больше нужного количества строк. Если буфер слишком велик, `vile` «берет в долг» до ¾ места у соседней панели.

содержимое пропадает. Есть и другие «невидимые» буферы, например буферы, содержащие скрипты. Как правило, ни те, ни другие не отображаются в [Buffer List].

## Полосы прокрутки

Справа от каждой панели находится полоса прокрутки, которую можно использовать для перемещения по буферу стандартным способом. Заметим, что этот стандартный способ различается в разных инструментариях. В версиях Athena и «No Toolkit» средняя кнопка мыши может использоваться для перетаскивания «бегунка» или видимого индикатора, а левая и правая перемещают вниз и вверх (соответственно) по буферу. Количество перемещаемого контента зависит от положения курсора мыши на полосе прокрутки. Если поместить его около верха, перемещение будет происходить построчно, а если около низа – по целым панелям.

Возможно, полоса прокрутки Motif более привычна. Для всех операций используется левая кнопка мыши. Щелчок по маленьким стрелочкам сместит вас вверх или вниз на одну строку. Для быстрого перемещения можно перетаскивать индикатор полосы прокрутки, а если щелкнуть повыше или пониже индикатора, то произойдет перемещение вверх или вниз на всю панель.

Во всех версиях имеется небольшая «рукоятка» выше или ниже полос прокрутки (находится между ними), которую можно использовать для подстройки двух соседних панелей. В версии xvile «No Toolkit» эта рукоятка вписывается в строку состояния двух смежных панелей. В других версиях она более различима, но во всех случаях курсор мыши при его постановке над рукояткой смены размера меняется на большую вертикальную двойную стрелку. Размер окон можно поменять перетаскиванием рукоятки мышью.

Панель можно разбить на две части, если при нажатой клавише Ctrl щелкнуть левой кнопкой мыши на полосе прокрутки. Вы получите два представления одного буфера. Если хотите сменить одно из представлений на другой буфер, используйте другие команды vile. Панели можно удалить, если удерживать клавишу Ctrl и нажать на среднюю кнопку мыши. Если создано много панелей, можно сделать так, чтобы все окно занимала одна панель. Для этого нажмите Ctrl и щелкните правой кнопкой мыши. Все другие панели удалятся, оставив окно xvile с панелью, по которой вы щелкнули. Обзор действий представлен в табл. 18.2.

*Таблица 18.2. Команды vile для управления панелями*

| **Команда** | **Функция** |
|---|---|
| Ctrl-левая кнопка | На полосе прокрутки, разделяет панель. |
| Ctrl-средняя кнопка | Удаляет панель. |
| Ctrl-правая кнопка | Оставляет на экране только ту панель, по которой щелкнули. |

## Установка положения курсора и движения мыши

Внутри текстовой области панели курсор можно установить простым щелчком левой кнопки мыши. Это действие задаст не только положение курсора, но и панель, в которой будет происходить редактирование. Чтобы поменять ее, но сохранить ее старое положение, щелкните по строке состояния под тем текстом, который вы хотите исправить.

Щелчок мышью рассматривается как перемещение, аналогично, например, команде `4j`. Чтобы удалить пять строк, можно ввести `d4j`, удалив текущую строку и четыре под ней. То же самое делается и с помощью мыши. Поместите курсор в место начала удаления и нажмите `d`, после чего щелкните по буферу в том месте, до которого вы хотите удалить. Щелчки мышью являются перемещением курсора, так что они могут использоваться и в других операциях.

## Выделения

Чтобы задать выделение, перетащите мышь с нажатой левой кнопкой. Это называется PRIMARY (первичным) выделением. Если отпустить кнопку мыши, то выделение копируется и становится доступным для вставки. Можно потребовать, чтобы выделенная область была прямоугольной, если удерживать клавишу Ctrl при перетаскивании с нажатой левой кнопкой мыши. Если перетаскивание выходит за пределы окна, текст будет по возможности прокручиваться в нужном направлении, чтобы подогнать выделения, превышающие размер окна. Скорость, при которой идет прокрутка, будет возрастать со временем, что позволяет делать быстрое выделение больших кусков текста.

Отдельные слова или строки можно выделять двойным или тройным щелчком по ним.

С помощью правой кнопки мыши выделение может быть расширено. Как и в случае с левой кнопкой, его можно подстроить или прокрутить, удерживая правую кнопку при перетаскивании. Выделения можно расширять и в других окнах, если в них открыт тот же буфер, что и в окне, в котором было сделано выделение. То есть если у вас есть два представления буфера (на двух разных панелях), причем в одном окне показано начало буфера, а в другом – его конец, можно выделить весь буфер, если нажать левую кнопку в начале панели с началом буфера и правую – в панели с его концом. Также выделения можно делать прямоугольными, сочетая клавишу Ctrl с правой кнопкой мыши.

Средняя кнопка мыши используется для вставки выделения. По умолчанию она вставляет текст в последнюю позицию курсора. Если при ее нажатии удерживать клавишу Shift, то вставка произойдет на месте курсора.

Выделение можно очистить (если оно сделано в редакторе `xvile`), дважды щелкнув по строке состояния.

## Системный буфер (Clipboard)

Можно производить обмен данными с другими приложениями X через PRIMARY-выделение. Выше было описано, как задать выделение и управлять им.

Другие приложения используют выделение CLIPBOARD для обмена данными между собой. На многих клавиатурах Sun выделенный текст попадает в системный буфер при нажатии клавиши COPY, а вставляется оттуда при нажатии на PASTE. Если у вас не получается вставить текст из xvile в другие программы (или наоборот), то это, скорее всего, из-за использования этими программами выделения CLIPBOARD вместо PRIMARY. (Другой механизм, который используется очень старыми приложениями, включает использование кольца буферов.)

В xvile есть две команды для управления системным буфером: copy-to-clipboard и paste-from-clipboard. При вызове copy-to-clipboard содержимое текущего выделения копируется в специальный регистр (clipboard kill register), который обозначается как ; в списке регистров. Когда приложение запрашивает выделение из системного буфера, xvile выдает содержимое этого регистра. Команда paste-from-clipboard запрашивает данные системного буфера у текущего владельца выделения CLIPBOARD.

Возможно, пользователям систем Sun захочется определить следующие сочетания клавиш в файле .vilerc, чтобы можно было использовать кнопки COPY и PASTE:

```
bind-key copy-to-clipboard #-^
bind-key paste-from-clipboard #-*
```

В этой главе мы еще вернемся к привязкам клавиш.

## Ресурсы

xvile содержит множество ресурсов, посредством которых можно управлять внешним видом и поведением программы. Особенно важен выбор шрифта, так как он определяет правильность отображения курсивных и наклонных шрифтов. Документация vile содержит полный список ресурсов, а также образец записей в .Xdefault.

## Добавление меню

В версиях Motif и Athena имеется поддержка меню. Задаваемые пользователем пункты меню считываются из файла .vilemenu текущего или домашнего каталога. xvile допускает пункты меню трех типов:

- Встроенные, то есть специфичные для системы меню, такие как повторное считывание файла .vilerc или создание новой копии xvile.
- Непосредственный вызов встроенных команд (например, показать [Buffer List]).

- Вызов произвольных командных последовательностей (например, запуск интерактивного макроса, такого как команда поиска).

Мы считаем два последних пункта различными, так как авторы предпочли, чтобы `vile` мог проверять правильность команд перед их выполнением.

## Сборка winvile

Для каждого релиза `winvile` доступны бинарные файлы, но вам может захотеться скомпилировать редактор из промежуточных пропатченных версий (interim patch versions). В исходниках есть Make-файлы для компиляторов Microsoft (`makefile.wnt`) и Borland (`makefile.tbc`). В первом есть больше возможностей, например сборка программы с поддержкой `OLE`, `perl` и встроенной подсветкой синтаксиса. GUI для Win32 можно собрать любым компилятором.

## Базовый внешний вид и функциональность winvile

На рис. 18.4 и 18.5 показан графический интерфейс Win32 для `winvile`. На первый взгляд он очень похож на интерфейс «No Toolkit» в X11 с полосами прокрутки. Если копнуть поглубже – а это несложно – он оказывается более замысловатым, чем интерфейс Motif.

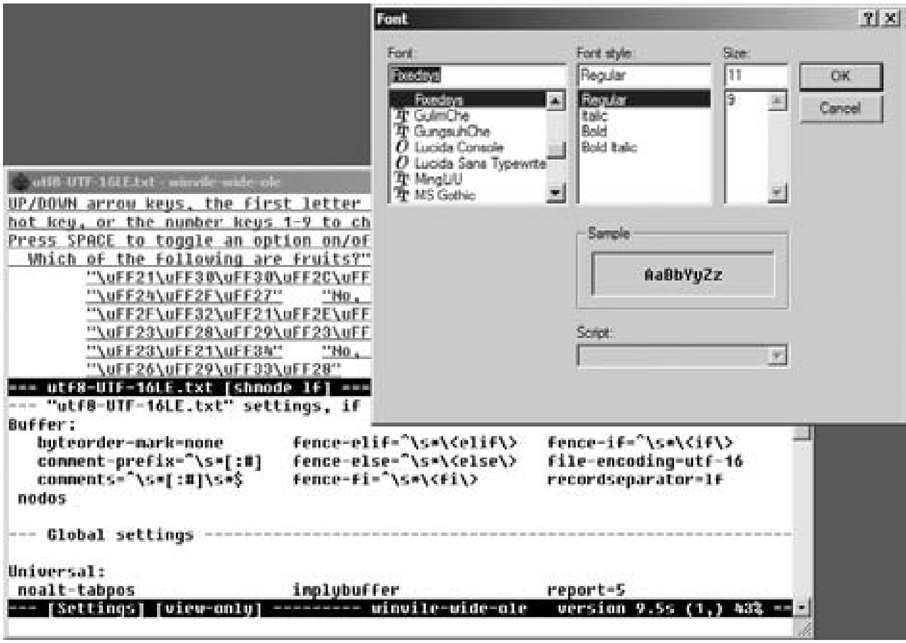



*Рис. 18.4. winvile с не-Unicode шрифтом*

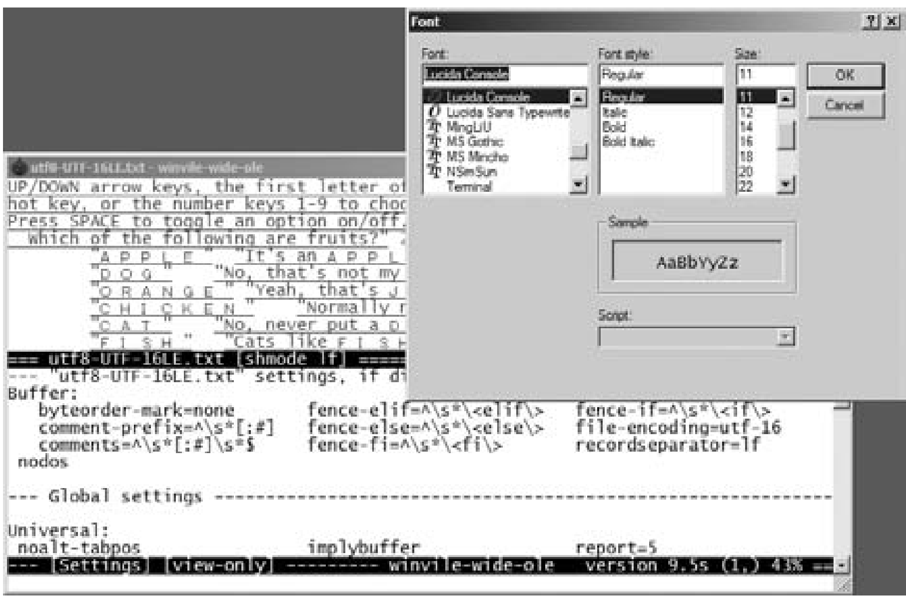


*Рис. 18.5. winvile со шрифтом Unicode*

На рис. 18.4 показан вид `winvile` при редактировании данных Unicode:

- Диалог выбора шрифтов изначально установлен на моноширинный системный шрифт. Как и в `xvile`, этот шрифт можно задавать при старте `winvile` или с помощью скрипта. Также его можно установить посредством OLE-сервера. Наконец, как показано на рисунке, его можно задать обычными средствами Win32.
- Данные имеют формат Unicode UTF-16, без отметки о порядке байтов. Они подчеркнуты, так как палитра подсветки использует подчеркивание и голубой (cyan) цвет для раскраски строк в кавычках.
- Системный шрифт, стоящий по умолчанию, не может отобразить символы в файле. `winvile` видит, что шрифт мал, и отображает данные Unicode в шестнадцатеричном виде.

На рис. 18.5 показан результат выбора более подходящего шрифта. Если снова выбрать системный шрифт, программа снова покажет шестнадцатеричные значения. Если необходимо постоянно видеть символы в шестнадцатеричном виде, в `vile` есть соответствующая опция.

На рис. 18.6 показаны несколько функций меню `winvile`, среди которых:

- `winvile` расширяет системное меню. Его можно увидеть, щелкнув правой кнопкой мыши по заголовку окна. В появившемся меню есть те же пункты, то есть не нужно каждый раз перемещаться наверх. Это поведение включается с помощью пункта *Menu* внизу меню.

- Меню обеспечивают операции открытия, сохранения, печати и работы со шрифтом, типичные для графических приложений. В пункте *CD* можно задать текущий рабочий каталог `winvile`.

  Соответствующие диалоги также доступны в консольной версии для Win32, но без меню.

- `winvile` также позволяет заходить в папку Windows *Favorites* (*Избранное*).
- Недавние файлы (и недавние папки) выбираются из нескольких (количество задается пользователем) недавно использовавшихся файлов (или папок). Программа сохраняет эти имена в реестре, что делает их доступными из всех запущенных экземпляров `winvile`.

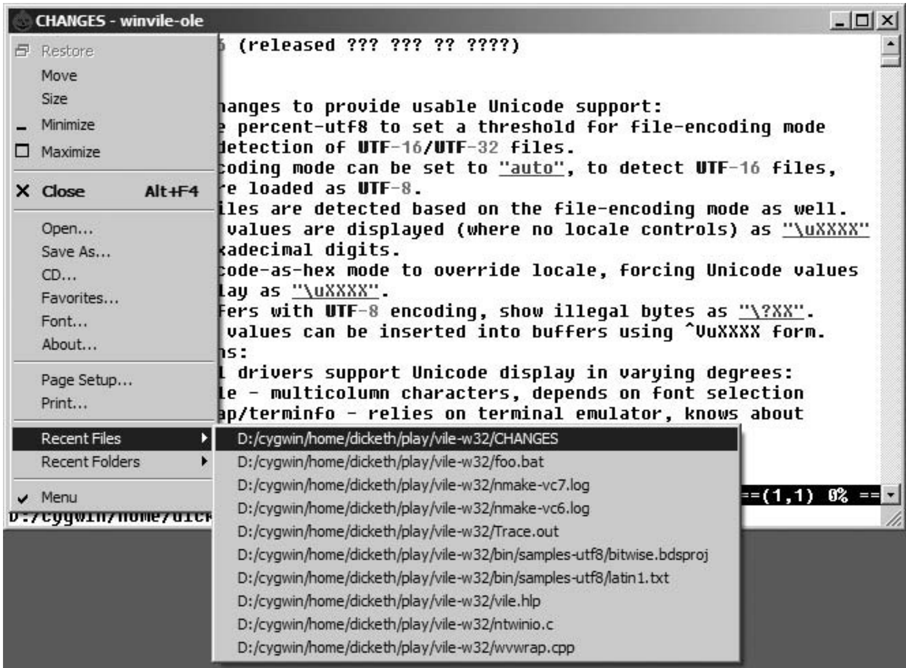



***Рис. 18.6.*** *Меню последних файлов в winvile*

# Расширенные регулярные выражения

Мы уже познакомились с расширенными регулярными выражениями в разделе «Расширенные регулярные выражения» на стр. 152. `vile` предоставляет примерно те же возможности, что и опция `extended` в `nvi`. Они включают групповые выражения POSIX для классов символов, `[[:alnum::]]`, с некоторыми расширениями (дополнительные классы и со-

кращения) и интервальные выражения, такие как {,10}. Синтаксис немного отличается от синтаксиса в nvi – упор делается на дополнительные символы, экранированные обратной косой чертой:

\|

Указывает на альтернативу: house\|home.

\+

Соответствует одному или более предшествующим регулярным выражениям.

\?

Соответствует одному или ни одному предшествующему регулярному выражению.

\(...\)

Предоставляет группировку для *, \+ и \?, а также позволяет указывать подстроки в заменяющей части команды подстановки (\1, \2 и т. д.).

\s, \S

Соответствует пробельным и видимым символам соответственно.

\w, \W

Определяет символы, «из которых состоят слова» (буквенно-цифровые и знак подчеркивания, «_»), и символы, таковыми не являющиеся, соответственно. Например, \w\+ задает ключевые слова и идентификаторы в C/C++[1].

\d, \D

Задает цифры и нецифровые символы соответственно.

\p, \P

Определяет непечатаемые и печатаемые символы соответственно. Пробелы относятся к печатаемым.

vile допускает использование escape-последовательностей \b, \f, \r, \t и \n в заменяющей части команды замены. Они означают Backspace, запрос на подачу страницы, возврат каретки, табуляцию и символ новой строки, соответственно.

> Заметим, что vile подражает perl-обработке \u\L\1\E, а не той, которая есть в vi. Если скомандовать :s/\(abc\)/\u\L\1\E/, то vi заменит на *abc*, тогда как vile и perl заменят на *Abc*. Это полезнее для капитализации слов.

[1] Для самых въедливых добавим, что это также соответствует идентификаторам, начинающимся с цифры; обычно это не приводит к проблемам.

# Улучшенные возможности редактирования

В этом разделе описаны функции vile, ускоряющие и упрощающие обычное редактирование.

## История командной строки и автозавершение

vile запоминает вводимые пользователем команды ex в буфере под названием [History]. Эта возможность управляется опцией history, которая по умолчанию установлена в значение true. Если она выключена, то сохранение команд не ведется, а буфер [History] удаляется. Команда show-history разделяет экран, после чего в новом окне отображается буфер [History].

В реальности командная строка с двоеточием является мини-буфером. Ее можно использовать, чтобы вспомнить строки из буфера [History] и отредактировать их.

Клавиши ↑ и ↓ применяются для прокрутки вверх и вниз по истории команд, а ← и → – для перемещения по строке. Для удаления символов используется текущий символ удаления (как правило, это BACKSPACE). Все остальные вводимые знаки отображаются в текущей позиции курсора.

Мини-буфер можно переключить в режим vi, если ввести символ mini-edit (по умолчанию это ^G). После этого vile подсветит мини-буфер с помощью механизма, определяемого опцией mini-hilite. По умолчанию значение опции – reverse, инвертирующее цвет. В режиме vi можно пользоваться командами vi для перемещения. Также можно задавать команды vile, подходящие для редактирования в одной строке, например i, I, a и A. vile решает, какие команды будут работать, основываясь на своих таблицах команд, что позволяет пользовательским привязкам клавиш также работать в мини-буфере.

Интересной особенностью vile является использование им истории для отображения ранее вводимых данных, соответствующих вводимой команде. Например, после ввода :set и пробела vile высветит подсказку Global value:. При этом можно использовать ↑, чтобы увидеть предыдущие заданные вами глобальные переменные и получить возможность изменить одну из них.

Командная строка ex предоставляет автозавершение различного вида. При вводе имени команды можно в любой момент нажать клавишу TAB, после чего vile по возможности заполнит остаток имени команды. Если нажать TAB во второй раз, программа создаст новое окно, в котором будут показаны все возможные варианты завершения.

Завершение производится у встроенных и пользовательских команд vile, тегов, имен файлов, режимов (про них чуть позже), переменных, значений перечислений (например, имен цветов) и символов терминала

(сюда относятся такие символы, как забой, останов и прочие, определяемые настройками stty).

Заметим, что это приводит к интересному явлению. В редакторах типа vi команды могут иметь очень длинные имена, обычно с уникальными несколькими первыми символами, поэтому принимаются аббревиатуры команд. В редакторах типа Emacs имена команд зачастую не уникальны в первых символах, но поддержка завершения команд позволяет уменьшить объем ввода.

## Стеки тегов

Стеки тегов были описаны ранее в разделе «Стеки тегов» на стр. 157. В vile они доступны и работают непосредственным образом. Здесь они немного отличаются от остальных модификаций vi. Основные отличия – в командах режима vi, используемых для поиска по тегам и их извлечения из стека. В табл. 18.3 показаны команды работы с тегами в vile.

*Таблица 18.3. Команды работы с тегами в vile*

| **Команда** | **Функция** |
|---|---|
| next-tag | Продолжает поиск других соответствий в файле tags. |
| pop[!] | Извлекает положение курсора из стека, восстанавливая курсор в прежней позиции. |
| show-tagstack | Создает новое окно, в котором показан стек тегов. Содержимое окна меняется по мере помещения тегов в стек и их извлечения из него. |
| ta[g][!] [*tagstring*] | Редактирует файл, содержащий *tagstring*, как определено в файле tags. Восклицательный знак ! вынуждает vile переключиться в новый файл, даже если текущий буфер изменен и не сохранен. |

Команды для режима vi описаны в табл. 18.4.

*Таблица 18.4. Команды работы с тегами в командном режиме vile*

| **Команда** | **Функция** |
|---|---|
| ^] | Ищет в файле tags местоположение идентификатора, на котором стоит курсор, и переходит на это место. Текущая позиция заносится в стек тегов. |
| ^T, ^X ^] | Восстанавливает предыдущее положение из стека тегов, то есть извлекает из него один элемент. |
| ^A ^] | То же, что команда :next-tag. |

Как и в других редакторах, способ обращения vile с командами для тегов управляется опциями, как показано в табл. 18.5.

*Таблица 18.5. Опции vile для управления тегами*

| Опция | Функция |
|---|---|
| pin-tagstack | Поиск по тегам и извлечение из стека не меняют текущего окна, то есть оно «закреплено» (pin). По умолчанию эта опция имеет значение false. |
| tagignorecase | Поиск по тегам игнорирует регистр. По умолчанию установлено значение false. |
| taglength | Определяет количество значимых символов в теге, который ищется. Значение по умолчанию, равное нулю, указывает, что значимы все символы. |
| tagrelative | При использовании файла tags из другого каталога имена файлов из этого файла tags рассматриваются относительно того каталога, где расположен файл tags. |
| tags | Этой опции можно присвоить список файлов tags, разделенных пробелами, в которых нужно искать теги. vile загружает все файлы tags в специальные буферы, которые по умолчанию скрыты, но при желании их можно отредактировать. В tags можно помещать переменные окружения и маски оболочки. |
| tagword | Эта опция говорит, что для поиска тега нужно брать все слово, на котором стоит курсор, а не ту его часть, которая начинается с позиции курсора. По умолчанию эта опция выключена, что делает vile совместимым с vi. |

## Бесконечная отмена

В принципе здесь vile похож на другие редакторы, но отличается от них в практике применения. Как в elvis и Vim, в нем можно установить предел для отмен, но, как в nvi, команда . произведет следующую отмену или повторение в зависимости от ситуации. В режиме vi последовательные отмены и возвраты реализованы разными командами.

Для управления количеством запоминаемых изменений vile использует опцию undolimit. По умолчанию она равна 10, то есть вы можете отменить 10 последних изменений. Если установить ее равной нулю, то будет осуществлена поистине «бесконечная отмена», но это может потребовать больших объемов памяти.

Чтобы начать отмену, наберите сначала команды u или ^X u. После этого каждый последующий ввод символа . приведет к новой отмене. Как и в vi, две команды u просто переключают состояние изменения[1], однако ^X u действительно сделает новую отмену.

Команда ^X r выполняет возврат. Если ввести . после ^X r, то он повторится. К командам ^X u и ^X r можно добавить число, после чего vile произведет нужное количество отмен или возвратов.

[1] Отменено или не отменено. – *Прим. науч. ред.*

## Строки произвольной длины и двоичные данные

vile может редактировать файлы со строками произвольной длины и с произвольным числом строк.

Программа автоматически поддерживает двоичные данные. Для этого не требуется никаких специальных команд или опций. Для ввода 8-разрядного символа нажмите ^V, а затем x и две шестнадцатеричные цифры, либо 0 и три восьмеричные цифры, либо три десятичные цифры.

Также можно ввести 16-разрядные данные Unicode, если нажать ^V, затем u, после чего ввести до четырех шестнадцатеричных цифр. Если значение опции file-encoding для текущего буфера – одно из юникодовских (utf-8, utf-16 или utf-32), vile сохраняет его напрямую как UTF-8, отображая его согласно возможностям терминала или дисплея.

Таким образом, мы пришли к теме локализации.

### Поддержка локализации

На протяжении многих лет vile имел лишь зачаточную поддержку локалей. Частично это объяснялось тем, что на различных платформах (за исключением фирменных систем UNIX) эта поддержка была рудиментарной. В vile присутствовали собственные таблицы типов символов (то есть управляющие, числовые, печатаемые, знаки препинания, а также используемые в именах файлов, масках символов и оболочке), что позволяло указать, какие из не-ASCII символов являются печатаемыми.

Времена менялись, и vile менялся соответственно потребностям пользователей. Приведем краткий перечень изменений, упорядоченных логически, а не по хронологии разработки:

- Теперь vile импортирует символьные таблицы с хоста и предоставляет команды для их изменения через скрипты, а не содержит фиксированные таблицы символов.[1]
- Регулярные выражения vile поддерживают классы символов POSIX, а также классы, соответствующие собственным типам символов программы.
- Редактор поддерживает извлечение лексических единиц (токенов, tokens) с экрана, например для tags, скриптов и прочего. Раньше это свойство подразумевало смесь проверок на тип символа со специальным синтаксическим анализатором. Сейчас это только регулярные выражения без необходимости в дополнительной логике анализатора.
- Редактирование файла, содержащего 8-разрядные данные. Например, представление данных в кодировке ISO-8859-7 (Греческая), когда локаль хоста использует UTF-8, может оказаться непростым делом. При старте vile программа проверяет, заканчивается ли локаль

[1] Эта функция полезна даже на фирменных системах UNIX, где не всегда имеются правильные таблицы.

на UTF-8 (или на схожие символы), например el_GR.UTF-8. Если да, то редактор поддерживает редактирование в соответствующей 8-разрядной локали, например el_GR.

- Аналогично при редактировании файлов в среде хоста, поддерживающей UTF-8, будут встречаться файлы в кодировке UTF-8. В более новой версии можно приказать vile сохранять файлы в различных кодировках Unicode и считывать те же кодировки. Поддерживается 8-разрядная модель редактирования, в которой буферы, отмеченные как 8-разрядные, преобразуются в 8-разрядную кодировку, а буферы в Unicode редактируются непосредственно (то есть без преобразования).

Все это – расширения. На каждом этапе все еще остаются старые функции.

Есть и другие аспекты локализации, такие как формат сообщений и порядок сравнения текста, но vile их не рассматривает.

### Форматы файлов

Когда vile считывает файл, он делает несколько предположений о его содержимом, чтобы представить пользователю понятные данные:

- Проводится проверка на наличие у пользователя прав на запись файла.
- Проверяется тип конца строки (CR, LF или CR/LF).
- Делается проверка отметок порядка следования байтов для Unicode.
- Проверяется, используется ли многобайтовая кодировка Unicode.

На основе этих проверок программа устанавливает свойства (они называются «режимами») для только что считанного буфера, которые применяются к этому буферу. Кроме того, vile может преобразовывать данные при чтении следующим образом:

- Удалять символ конца строки из каждой строки, запоминая соответствующий режим recordseparator.
- Если у файла нет завершающего переноса строки, программа устанавливает опцию nonewline.
- Переводить данные UTF-16 и UTF-32 в UTF-8, запоминая соответствующую опцию file-encoding.

Когда vile получает команду сохранить буфер в файл, он использует эти локальные опции для восстановления файла.

## Инкрементный поиск

Как упоминалось ранее в разделе «Инкрементный поиск» на стр. 162, в vile инкрементный поиск осуществляется командами ^X S и ^X R. Чтобы его активировать, не нужно устанавливать какую-либо опцию.

При вводе курсор перемещается по файлу, всегда попадая на первый символ текста, где обнаружено соответствие. ^X S производит инкрементный поиск вперед по файлу, а ^X R — в обратном направлении.

Возможно, вы захотите добавить в файл .vilerc следующие команды (про них будет рассказано позже в разделе «Модель редактирования vile» на стр. 411), чтобы инкрементный поиск производился более знакомыми командами поиска / и ?:

```
bind-key incremental-search /
bind-key reverse-incremental-search ?
```

Также представляет интерес возможность «visual match» (визуализации совпадений), когда подсвечиваются *все* вхождения искомого выражения. Следующая строка в файле .vilerc:

```
set visual-matches reverse
```

предписывает vile использовать для визуализации совпадений инверсию цвета. Поскольку иногда подсветка может усложнять просмотр, команда = выключает любую текущую подсветку, пока вы не введете новый шаблон поиска.

## Прокрутка влево-вправо

Как упоминалось ранее в разделе «Прокрутка влево-вправо» на стр. 163, такая прокрутка в vile включается опцией :set nolinewrap. В отличие от других редакторов, здесь она установлена по умолчанию. Длинные строки отмечаются слева и справа значками < и >. Значение sideways определяет количество символов, на которые vile смещает экран при прокрутке влево или вправо. Если sideways установить равным нулю, каждая прокрутка переместит экран на одну треть. Иначе экран будет смещаться на заданное количество символов.

## Визуальный режим

vile отличается от elvis и Vim в том, как он подсвечивает текст, ожидающий обработку. Редактор использует команду «выделенного перемещения» (quoted motion) — q.

В начале области вы вводите q, затем любые команды перемещения vi, чтобы добраться до конца области, и снова нажимаете q, чтобы закончить выделенное перемещение. vile подсветит выделенный текст.

Аргументы команды q задают тип подсветки. 1q (то же, что и q) выполняет точную подсветку, 2q подсвечивает только строки, а 3q выполняет прямоугольное выделение.

Как правило, выделенное перемещение используется в сочетании с оператором, таким как d или y. Таким образом, d3qjjwq удалит прямоугольник, заданный перемещением. Если использовать выделение без оператора, область остается подсвеченной, и к ней можно обратиться нажатием ^S. Таким образом, d ^S удалит подсвеченную область.

Кроме того, прямоугольные области могут задаваться метками[1]. Как известно, метка может использоваться для отсылки либо к определенному символу (когда к ней обращаются по `), либо к определенной строке (когда к ней обращаются по '). Кроме того, ссылка на метку (скажем, метку, заданную командой mb) с помощью `b вместо 'b меняет тип производимой операции — d'b удалит набор строк, а d`b — части двух строк и строки между ними. Использование ссылки на метку в виде ` даст более «точную» область, чем ссылка на метку вида '.

В vile есть ссылка на метку третьей разновидности. Команда \ может использоваться как новый способ ссылки на метку. Сама по себе она ведет себя как ` и перемещает курсор на тот символ, где была установлена метка. Если же ее сочетать с оператором, то ее поведение меняется. Ссылка на метку становится «прямоугольником», так что действие d\b удалит этот прямоугольник из символов, углы которого определяются положением курсора и символом, на котором стоит метка b.

**Клавиши** — **Результат**

ma

```
The 6th edition of <citetitle>Learning the vi Editor</citetitle>
brings the book into the late 1990&rsquo;s.
In particular, besides the &ldquo;original&rdquo; version of
<command>vi</command> that comes as a standard part of every Unix
system,
there are now a number of freely available &ldquo;clones&rdquo;
or work-alike editors.
```

Установить на *b* в *book* метку a.

3jfr

```
The 6th edition of <citetitle>Learning the vi Editor</citetitle>
brings the book into the late 1990&rsquo;s.
In particular, besides the &ldquo;original&rdquo; version of
<command >vi</command> that comes as a standard part of every Unix
system,
there are now a number of freely available &ldquo;clones&rdquo;
or work-alike editors.
```

Переместить курсор на *r* в *number*, чтобы отметить противоположный угол.

^A ~\a

```
The 6th edition of <citetitle>Learning the vi Editor</citetitle>
brings the BOOK INTO The late 1990&rsquo;s.
In particulAR, BESIDES the &ldquo;original&rdquo; version of
<command>vi</COMMAND> that comes as a standard part of every Unix
system,
there are nOW A NUMBER of freely available &ldquo;clones&rdquo;
or work-alike editors.
```

Поменять регистр прямоугольника, ограниченного меткой a.

[1] Спасибо Полу Фоксу (Paul Fox) за это объяснение.

Команды, задающие произвольные области и производящие над ними операции, обобщены в табл. 18.6.

*Таблица 18.6. Команды vile для блочного режима*

| Команда | Операция |
|---|---|
| q | Начало и конец выделенного перемещения. |
| ^A r | Открывает прямоугольник. |
| > | Смещение текста вправо. То же, что и ^A r, когда область является прямоугольником. |
| < | Смещение текста влево. То же, что и d, когда область является прямоугольником. |
| y | Копирование всей области. vile запоминает, что она была прямоугольной. |
| c | Изменение области. Для непрямоугольной области удаляет весь текст между конечными точками и переходит в режим вставки. Если область прямоугольная, запрашивает ввод текста, который будет заполнять строки. |
| ^A u | Меняет регистр всех букв области на прописной. |
| ^A l | Меняет регистр всех букв области на строчный. |
| ^A ~ | Переключает регистр всех букв области. |
| ^A SPACE | Заполняет область пробелами. |
| p, P | Вставляет текст обратно. vile выполняет прямоугольную вставку, если исходный текст был прямоугольным. |
| ^A p, ^A P | Заставляет ранее скопированный текст вставиться обратно, как если бы он был прямоугольным. В качестве ширины прямоугольника берется длина самой длинной скопированной строки. |

# Помощь программисту

В этом разделе рассматриваются возможности облегчить работу программиста в vile.

## Ускорение цикла редактирование–компиляция

Для управления разработкой программы vile использует две команды режима vi, которые приведены в табл. 18.7.

Программа понимает сообщения *Entering directory XXX* и *Leaving directory XXX*, генерирующие GNU make, что позволяет ей отыскать нужный файл, даже если он находится в другом каталоге.

Анализ сообщений об ошибках осуществляется с помощью регулярных выражений, хранящихся в буфере [Error Expressions]. vile создает его автоматически, а затем использует при нажатии ^X ^X. При необходимо-

сти туда можно добавлять выражения. Тогда программа использует расширенный синтаксис, позволяющий указывать расположение имен файлов, номеров строк, столбцов и прочего в сообщениях об ошибках. В онлайн-справке можно прочитать подробности, хотя, возможно, вам не придется делать никаких изменений, поскольку даже в изначальном виде все работает хорошо.

*Таблица 18.7. Команды vile для разработки программ в режиме vi*

| **Команда** | **Функция** |
|---|---|
| `^X !`*command* ENTER | Запускает команду, а ее вывод сохраняет в буфер под именем `[Output]`. |
| `^X ^X` | Находит следующую ошибку. `vile` делает синтаксический анализ вывода и перемещает на позицию следующей ошибки. |

Поиск ошибок в `vile` также отслеживает изменения файла, учитывая вставки и удаления при переходе от одной ошибки к другой.

Поиск ошибок применяется к самому последнему буферу, созданному путем чтения вывода команды оболочки. Например, команда `^X!command` сгенерирует буфер с именем `[Output]`, а команда `:e !command` – с именем `[!command]`. Поиск ошибок подстроится соответствующим образом.

Используя команду `:error-buffer`, можно выполнить поиск ошибок в любом произвольном буфере (а не только в том, который является выводом команд оболочки). Это позволит использовать поиск ошибок для предыдущего вывода компилятора или команды `egrep`.

## Подсветка синтаксиса

`vile` поддерживает подсветку синтаксиса во всех своих конфигурациях. Для расцветки синтаксиса используются специальные программы – *фильтры синтаксиса*. Они могут быть встроенными в `vile` или запускаться как внешние. Редактор посылает содержимое раскрашиваемого буфера в фильтр синтаксиса, считывает версию с разметкой и применяет эту разметку для раскраски буфера.

Встроенные фильтры работают быстрее внешних программ, и оболочка не оказывает влияния на их отображение в терминале. На некоторых платформах может осуществляться динамическая подгрузка фильтров синтаксиса. Это позволяет уменьшить размер исполняемого модуля программы, хоть и не дает такой же скорости, как при встроенных фильтрах.

На настоящий момент существует 71 программа, а также отдельная программа для man-страниц UNIX. Некоторые программы используются для файлов нескольких типов. Например, C, C++ и Java имеют сходный синтаксис, но разные ключевые слова.

в vile есть макросы, запускающие фильтры синтаксиса либо автоматически при изменении буфера, либо по запросу. Они приведены в табл. 18.8.

*Таблица 18.8. Команды подсветки синтаксиса в vile*

| Команда | Привязка клавиш | Функция |
|---|---|---|
| :HighlightFilter | | Вызывает подсветку синтаксиса для данного буфера. vile выбирает фильтр, основываясь на расширенном свойстве буфера, называемом *основным режимом* (о нем будет рассказано позже в разделе «Основные режимы» на стр. 412).<br>Если фильтры встроенные, инициализация vile устанавливает режим *autocolor*, чтобы вызвать этот макрос через пять секунд после окончания изменения буфера пользователем. |
| :HighlightFilterMsg | ^X-q | Прикрепляет подсветку к текущему буферу с помощью HighlightFilter. После завершения выводит сообщение[a]. |
| :HighlightClear | ^X-Q | Очищает все подсветки для текущего буфера. Не влияет на основной режим буфера. |
| :set-highlighting *majormode* | | Меняет основной режим буфера на *majormode* и запускает подсветку синтаксиса. |
| :show-filtermsgs | | Показывает ошибки фильтра синтаксиса для текущего буфера. Если *фильтр синтаксиса* обнаруживает ошибки, он сообщает о них, и vile отображает их в буфере [Filter Messages], позволяя перемещаться по позициям, где были обнаружены ошибки. |

[a] Когда в середине 1990-х подсветка синтаксиса была впервые реализована в vile, было важно показать, что она завершилась. Времена изменились – машины стали быстрее.

При каждом своем запуске *фильтр синтаксиса* считывает один или более внешних файлов, содержащих ключевые слова, которые нужно выделять цветом, а также соответствующие цвета и атрибуты для них (жирный, подчеркнутый, курсивный и прочие). Он ищет эти файлы (с расширением .keywords), используя имя режима *majormode* буфера. В онлайн-справке приведены правила для этого поиска. Можно использовать макрос :which-keywords, чтобы увидеть, где vile ищет эти файлы, а также где именно он их нашел. См. пример 18.1.

*Пример 18.1. Примерный вывод «:which-keywords cmode»*

```
Show which keyword-files are tested for:
        cmode                ❶
```

```
(* marks found-files)

$cwd                        ❷
  ./.c.keywords
$HOME
  ~/.c.keywords
  ~/.vile/c.keywords
$startup-path               ❸
* /usr/local/share/vile/c.keywords
```

❶ *major mode*, который всегда заканчивается на «mode»

❷ Ваш текущий рабочий каталог

❸ Путь поиска скриптов `vile`

Фильтры синтаксиса `vile` используют общий набор цветов, определенный как классы: `Action`, `Comment`, `Error`, `Ident`, `Ident2`, `Keyword`, `Keyword2`, `Literal`, `Number`, `Preproc` и `Type`, независимо от того, сконфигурирован ли `vile` для X11, терминала (`termcap`, `terminfo`, `curses`) или Windows. Большая часть определений ключевых слов отсылает к классу. Это позволяет менять все цвета, отредактировав лишь один файл, обычно – `$HOME/.vile.keywords`. Онлайн-справка дает подробности настройки цветов синтаксиса.

С одной стороны, по причине осуществления подсветки синтаксиса внешней программой можно написать сколько угодно подсветок для разных языков. С другой, из-за того что эти функции работают на низком уровне, выполнение подобной задачи непрограммистами затруднительно. Онлайн-справка описывает, как работают фильтры подсветки.

Каталог *ftp://invisible-island.net/vile/utilities* содержит фильтры для раскраски файлов `make`, `input`, `perl`, HTML и `troff`, созданные и присланные пользователями. Там же есть макросы, которые раскрашивают строки файлов RCS по их возрасту!

# Интересные особенности

`vile` содержит ряд интересных особенностей по теме этого раздела:

*Модель редактирования* `vile`

Модель редактирования `vile` сильно отличается от модели `vi`. Взяв за основу концепции Emacs, она обеспечивает изменение привязок клавиш и более динамичную командную строку.

*Основные режимы*

`vile` поддерживает «режимы» редактирования. Это группы установок опций, облегчающие редактирование файлов определенного типа.

*Процедурный язык*

Процедурный язык редактора позволяет определять функции и макросы, делающие программу более гибкой и программируемой.

*Разнообразные небольшие функции*

Несколько небольших функций для облегчения повседневного редактирования.

## Модель редактирования vile

В vi и его модификациях функции редактирования «зашиты» в программу, а ассоциации между символами команд и их действиями зафиксированы в коде. Например, клавиша x удаляет символы, а клавиша i запускает режим вставки. Не обращаясь к головокружительным фокусам, вы не сможете поменять функциональность этих двух клавиш (если вообще сможете).

Модель редактирования vile, полученная из Emacs через MicroEMACS, совсем другая. В редакторе есть определенные именованные функции, каждая из которых выполняет отдельную задачу редактирования, например delete-next-character или delete-previous-character. Многие из этих функций привязаны к сочетаниям клавиш, например delete-next-character привязана к клавише x.[1]

vile имеет различные разновидности привязок клавиш для командного режима, режима вставки и режима выделения. Здесь мы описываем привязки для нормального режима редактирования. Сменить привязку очень просто: нужно вызвать команду :bind-key, а в качестве аргументов задать ей имя функции и сочетание клавиш, к которому она будет привязана. Как уже упоминалось ранее, в файл .vilerc можно поместить следующие команды:

```
bind-key incremental-search /
bind-key reverse-incremental-search ?
```

Они меняют команды поиска / и ? так, чтобы те производили инкрементный поиск.

Помимо предопределенных функций vile содержит простой язык программирования, позволяющий писать процедуры. Учитывая это, вы сможете привязать команду, вызывающую процедуру, к определенному сочетанию клавиш. В качестве своего языка GNU Emacs использует чрезвычайно мощный вариант Lisp. В vile есть более простой язык с меньшим охватом.

Как и в Emacs, командная строка vi имеет большие интерактивные возможности. Многие команды в качестве аргумента отображают значение по умолчанию. Вы можете либо отредактировать его подходящим образом, либо принять, нажав на ENTER. При вводе команд редактирования в режиме vi, таких как команды изменения или удаления символов, вы увидите отклик от этой операции в строке состояния.

---

[1] В vile 9.6 имеется 421 функция (включая доступные только для конфигураций X11 или Win32), а предопределенных привязок клавиш – около 260.

«Любопытный» режим ex, о котором Пол (Paul) говорил раньше, лучше всего отражен в действии команды :s (substitute). Она выдает новое приглашение в трех случаях: для ввода шаблона поиска и текста замены, а также во флагах команды.

В качестве примера предположим, что вам захотелось поменять в файле все вхождения *perl* на *awk*. В других редакторах вы бы написали просто :1,$s/perl/awk/gENTER и именно это увидели бы в командной строке. Следующие примеры показывают содержимое командной строки с двоеточием vile *по мере ввода* вышеуказанной команды:

| Клавиши | Результат |
|---|---|
| :1,$s | Первая часть команды замены. |
| / | substitute pattern: █ |
| | vile запрашивает шаблон для поиска. Сюда подставляется любой ранее вводившийся шаблон для повторного использования. |
| perl/ | replacement string: █ |
| | На следующем разделителе / программа попросит ввести текст замены. Будет предлагаться любой ранее использовавшийся текст. |
| awk/ | (g)lobally, ([1-9])th occurrence on line, (c)onfirm, and/or (p)rint result: █ |
| | У последнего разделителя редактор подсказывает необязательные флаги. Введите любой нужный вам и нажмите ENTER. |

vile придерживается этого стиля во всех подходящих командах ex. Например, команда чтения (:r) предлагает имя последнего считанного файла. Чтобы считать его снова, нажмите ENTER.

Наконец, анализатор команд ex в vile слабее, чем в других редакторах. Например, вы не сможете использовать шаблоны поиска для задания диапазона строк (:/now/,/forever/s/perl/awk/g), а команда перемещения (m) не реализована. Тем не менее на практике отсутствие этих функций не является преградой.

## Основные режимы

*Основной режим* (*major mode*)[1] – это совокупность установок опций, которые применяются при редактировании файлов определенного типа. Эти опции, например ширина табуляции, применяются отдельно к каждому буферу.

vile предоставляет опции трех типов:

- *Универсальные* (*Universal*), действуют во всей программе.
- *Буферные* (*Buffer*), применяются к содержимому буфера памяти.

[1] В документации vile это слово пишется слитно.

- *Оконные* (*Window*), применяются к окнам (в нашей терминологии это панели).

Установки опций для *буфера* (*buffer*) и для *окна* (*window*) могут быть локальными и глобальными. Любой буфер (или окно, в зависимости от опции) может иметь собственное приватное (локальное) значение опции. Если такового нет, используется глобальное. Основные режимы вводят новый уровень между локальными и глобальными переменными *буфера*, предоставляя значения опций, которые используются буфером при отсутствии у него их приватных значений.

vile имеет два встроенных основных режима: cmode для правки программ на C и C++ и режим vile mode для скриптов редактора, которые загружены в буферы памяти. В режиме cmode можно использовать % для соответствия командам предпроцессора C (#if, #else и #endif). Программа выполнит автоматическую расстановку отступов при вводе фигурных скобок ({ и }) и осмысленное форматирование комментариев C. Значения опций tabstop и shiftwidth также устанавливаются для каждого основного режима в отдельности.

С помощью основных режимов можно использовать эти функции в программах, написанных на других языках. Следующий пример, любезно предоставленный Томасом Дики (Thomas Dickey), определяет новый основной режим shmode для редактирования скриптов оболочки Bourne. (Для других оболочек стиля Bourne, таких как ksh, bash или zsh, он тоже подходит.)

```
define-mode sh
set shsuf "\.sh$"
set shpre "^#!\\s*\/.*sh\\>$"
define-submode sh comment-prefix "^\\s*/[:#]"
define-submode sh comments       "^\\s*/\\?[:#]\\s+/\\?\\s*$"
define-submode sh fence-if       "^\\s*\\<if\\>"
define-submode sh fence-elif     "^\\s*\\<elif\\>"
define-submode sh fence-else     "^\\s*\\<else\\>"
define-submode sh fence-fi       "^\\s*\\<fi\\>"
```

Переменная shsuf (от shell suffix) описывает приставку к имени файла, указывающую, что файл является скриптом оболочки. Переменная shpre (от shell preamble) описывает первую строку файла, указывающую, что в файле содержится скрипт оболочки. Затем команда define-submode добавляет опции, применяемые только к тем буферам, в которых был установлен соответствующий основной режим. В этом примере устанавливается интеллектуальное форматирование комментариев и интеллектуальное поведение команды % для скриптов оболочки.

Вышеприведенный пример подробнее, чем нужно. При использовании ~with язык скриптов vile распознает более сокращенную запись:

```
define-mode sh
~with define-submode sh
        suf            "\.sh$"
```

```
        pre            "^#!\\s*\/.*sh\\>$"
        comment-prefix "^\\s*/[:#]"
        comments       "^\\s*/\\?[:#]\\s+/\\?\\s*$"
        fence-if       "^\\s*\\<if\\>"
        fence-elif     "^\\s*\\<elif\\>"
        fence-else     "^\\s*\\<else\\>"
        fence-fi       "^\\s*\\<fi\\>"
~endwith
```

В скриптах инициализации vile содержатся более 90 предопределенных основных режимов. Чтобы увидеть определения доступных основных режимов, воспользуйтесь командой :showmajormodes.

Критерии suffix и prefix используются vile для задания основного режима, который нужно применить при считывании файла в буфер.[1] В табл. 18.9 перечислены все эти критерии.

*Таблица 18.9. Критерии для основных режимов*

| Критерий | Описание |
|---|---|
| after | Заставляет выполнять проверку определенного основного режима после данного. Как правило, основные режимы проверяются в том порядке, в каком они определены. |
| before | Заставляет выполнять проверку определенного основного режима перед данным. Как правило, основные режимы проверяются в том порядке, в каком они определены. |
| mode-filename (mf) | Регулярное выражение, определяющее имена файлов, для которых будет установлен соответствующий основной режим. Это выражение применяется только к той части полного имени файла, из которой удалено имя каталога. |
| mode-pathname (mp) | Регулярное выражение, определяющее пути файлов, для которых будет установлен соответствующий основной режим. |
| preamble (pre) | Регулярное выражение, определяющее первую строку файлов, для которых будет установлен соответствующий основной режим. |
| qualifiers | Указывает, как сочетать критерии preamble и suffixes. Используйте all, чтобы использовать оба, и any, чтобы воспользоваться любым из них. |
| suffixes (suf) | Регулярное выражение, определяющее расширения в именах файлов, для которых будет установлен соответствующий основной режим. Это выражение применяется только к той части имени файла, которая идет после первой точки. |

Определенный основной режим можно всегда задать самому, например:

```
:setl cmode
```

[1] Эти критерии являются опциями четвертой категории, считая универсальные, буферные и оконные. Они не перечислены с другими в табл. B.5, поскольку задаются совершенно другим путем.

укажет программе установить «c» mode[1], но это действие не обновит подсветку синтаксиса. Чтобы сделать и то, и другое, используйте макрос:

```
:set-h cmode
```

(set-highlighting; см. табл. 18.8).

## Процедурный язык

Процедурный язык vile практически не отличается от того, который был в MicroEMACS. Комментарии начинаются в точки с запятой или двойной кавычки, имена переменных окружения (опции редактора) – с $, а пользовательские переменные – с %. Для проверки условий существует несколько встроенных функций, имена которых начинаются с &. Команды управления выполнением и некоторые другие начинаются с ~. Знак @, использованный вместе со строкой, запрашивает пользовательский ввод и возвращает набранное. Мы взяли из файла macros.doc следующий довольно странный пример, чтобы вы почувствовали «вкус» этого языка:

```
~if &sequal %curplace "timespace vortex"
        insert-string "First, rematerialize\n"
~endif
~if &sequal %planet "earth" ;If we have landed on earth...
        ~if &sequal %time "late 20th century" ;and we are then
                write-message "Contact U.N.I.T."
        ~else
                insert-string "Investigate the situation....\n"
                insert-string "(SAY 'stay here Sara')\n"
        ~endif
~elseif &sequal %planet "luna" ;If we have landed on our neighbor...
        write-message "Keep the door closed"
~else
        setv %conditions @"Atmosphere conditions outside? "
        ~if &sequal %conditions "safe"
                insert-string &cat "Go outside......" "\n"
                insert-string "lock the door\n"
        ~else
                insert-string "Dematerialize..try somewhen else"
                newline
        ~endif
~endif
```

Подобные процедуры можно хранить в нумерованных макросах либо присваивать им имена, чтобы к ним можно было привязать клавиши. Показанный пример особенно полезен при использовании порта vile на Tardis. ☺

---

[1] Команда setl устанавливает локальные свойства буфера, а команда :set cmode – основной режим по умолчанию, если vile не смог распознать файл.

Более реалистичный пример от Пола Фокса (Paul Fox) запускает grep, ищущий слово под курсором во всех файлах C. Затем он помещает результаты поиска в буфер, названный по данному слову, и выполняет такие установки, чтобы встроенный поиск ошибок (^X ^X) использовал этот вывод в качестве списка строк, на которые нужно перейти. В конце этот макрос привязывается к ^A g. Команда ~force разрешает аварийное завершение команды без вывода сообщения об ошибке:

```
14 store-macro
        set-variable %grepfor $identifier
        edit-file &cat "!egrep -n " &cat %grepfor " *.[ch]"
        ~force rename-buffer %grepfor
        error-buffer $cbufname
~endm
bind-key execute-macro-14 ^A-g
```

Пользовательские процедуры могут иметь параметры, почти аналогичные оболочке Bourne, но параметры могут быть ограничены конкретными типами данных. Это позволяет процедурам работать так, как ожидается в модели редактирования vile (и в механизме истории команд). Процедуры не являются полностью взаимозаменяемыми со встроенными командами, так как пока нет механизма, который позволял бы команде отмены рассматривать весь макрос как единую операцию.

Наконец, переменным read-hook и write-hook можно присвоить имена процедур, которые будут выполняться после чтения файла и перед его записью соответственно. Это позволяет проделывать примерно то же, что делают пред- и постобработка файлов в elvis и автокоманды в Vim.

Этот язык довольно мощный. В него включены операции сравнения и контроля за выполнением, а также переменные, дающие доступ к значительной части внутреннего состояния vile. Файл macros.doc в документации редактора приводит детальное описание языка.

## Разнообразные небольшие функции

Несколько других небольших функций, о которых стоит упомянуть:

*Перенаправление в* vile

Если вы поставите vile последней командой в цепи каналов, он создаст буфер с именем [Standard Input], что позволит его отредактировать. Возможно, это «идеальный вариант программы просмотра».

*Редактирование файлов Windows*

Опция dos, будучи установленной в true, приведет к тому, что vile будет убирать возврат каретки из конца строки при чтении каждого файла, а при записи поставит его снова. Это позволит облегчить редактирование файлов Windows в системах UNIX или GNU/Linux.

*Переформатирование текста*

Команда ^A f переформатирует текст, производя в выделенном фрагменте переносы слов с одной строки на другую. Она понимает ком-

ментарии C и оболочки (строки, начинающиеся на * или #) и цитаты в электронной почте (начинаются с >). Похожа на команду UNIX `fmt`, но работает быстрее.

*Форматирование информационной строки*

Переменная `modeline-format` – это строка, управляющая тем, как `vile` форматирует строку состояния. Эта строка находится внизу каждого окна и описывает состояние буфера, например его имя, текущий основной режим, изменен ли буфер, является ли текущим командный режим или режим вставки и прочее.[1]

Строка состоит из последовательностей, использующих проценты в стиле *printf(3)*. Например, `%b` обозначает имя буфера, `%m` – основной режим, а `%l` – номер строки, если была установлена опция `ruler`. Символы в строке, не являющиеся частью форматирования, понимаются буквально.

В `vile` есть много других возможностей. Совпадение «ощущений пальцев» с `vi` позволяет легко перейти на `vile` из другого редактора. Программируемость дает большую гибкость, а интерактивная природа и разумные установки по умолчанию делают его более дружественным для новичков, чем обычный `vi`.

## Исходный код и поддерживаемые операционные системы

Официальный веб-сайт `vile` – *http://invisible-island.net/vile/vile.html*. По `ftp` программу можно найти на *ftp://invisible-island.net/vile/vile.tar.gz*. Файл `vile.tar.gz` всегда является символической ссылкой на текущую версию.

Редактор написан на ANSI C. Его можно собрать и запустить под UNIX, OpenVMS, MS-DOS, консольной и GUI-версиями Win32, BeOS, QNX и OS/2.

Компиляция `vile` идет обычным образом. Загрузите дистрибутив по `ftp` или с веб-страницы, распакуйте его, запустите программу `configure` и вызовите `make`:

```
$ gzip -d < vile.tar.gz | tar -xvpf -
...
$ cd vile-*; ./configure
...
$ make
...
```

[1] Документация `vile` ссылается на это как на *modeline*. Однако, поскольку в `vile` также реализована функция `modeline` из `vi`, мы называем ее строкой состояния, чтобы избежать путаницы.

`vile` должен настроиться и собраться без проблем. Для установки воспользуйтесь `make install`.

Если вы хотите, чтобы раскраска синтаксиса работала гладко, вам, скорее всего, придется запустить `configure` с опцией `--with-builtin-filters`. Нужно использовать `flex` (версии 2.54a или более поздней), а не `lex`, так как версии этой утилиты для UNIX не всегда корректно работают. Также скрипт `configure` не воспримет версию `flex`, если она очень старая.

Если вам нужно рассказать об ошибке или проблеме в `vile`, напишите по адресу *vile@nongnu.org*. Для сообщения об ошибках предпочтительнее использовать именно его. При необходимости можете связаться с Томасом Дики (Thomas Dickey) по адресу *dickey@invisible-island.net*.

# IV

# Приложения

В части IV собран справочный материал, представляющий интерес для пользователя `vi`. Эта часть содержит следующие приложения:

- Приложение А «Редакторы vi, ex и Vim»
- Приложение В «Установка опций»
- Приложение С «Возможные проблемы»
- Приложение D «vi и Интернет»

# Редакторы vi, ex и Vim

В этом приложении в формате справочника собраны стандартные функции `vi`. Сюда включены команды, вводимые с двоеточием (известные как команды `ex`, так как они восходят к первоначальному изданию этого редактора), а также самые популярные команды Vim.

В этом приложении представлены следующие темы:

- Синтаксис командной строки
- Обзор операций `vi`
- Алфавитный список ключей командной строки
- Команды `vi`
- Конфигурация `vi`
- Основы `ex`
- Алфавитный список команд `ex`

## Синтаксис командной строки

Чаще всего применяются следующие три способа запуска сеанса `vi`:

```
vi [options] file
vi [options] +num file
vi [options] +/pattern file
```

Вы можете открыть файл для редактирования, опционально – на строке *num* или на первой строке, отвечающей шаблону *pattern*. Если файл *file* не указан, `vi` открывает пустой буфер.

## Опции командной строки

Поскольку vi и ex – это одна и та же программа, опции у них одинаковые. Однако некоторые опции подходят только для одной из версий. Опции, присущие только Vim, отмечены соответствующим образом.

+[ *num* ]

Начинает редактирование со строки под номером *num* либо с последней строки файла, если *num* опущено.

+/*pattern*

Начинает редактирование на первой строке, отвечающей шаблону *pattern*. (В ex это не сработает, если в стартовом файле .exrc включена опция nowrapscan, так как ex начинает редактирование на последней строке файла.)

+?*pattern*

Начинает редактирование с последней строки, отвечающей шаблону *pattern*.

-b

Редактирует файл в двоичном режиме. {Vim}

-c *command*

При старте выполняет заданную команду ex. В vi допускается только одна опция -c, а Vim принимает до 10 штук. Старая версия этой опции, +*command*, тоже поддерживается.

--cmd *command*

Эквивалентна -c, но выполнение команды происходит перед чтением ресурсных файлов. {Vim}

-C

В Solaris vi: аналогична -x, но предполагается, что файл уже зашифрован.

В Vim: запускает редактор в режиме совместимости с vi.

-d

Запуск в режиме diff. Работает как vimdiff. {Vim}

-D

Режим отладки для работы со скриптами. {Vim}

-e

Запуск как ex (в строковом, а не полноэкранном режиме).

-h

Вывод информационного сообщения и выход. {Vim}

`-i` *`file`*

Использовать указанный файл *file* вместо применяемого по умолчанию (`~/.viminfo`) для сохранения и восстановления состояния Vim. {Vim}

`-l`

Вход в режим Lisp для запуска программ на Lisp (поддерживается не во всех версиях).

`-L`

Выводит список всех файлов, которые были сохранены из-за сбоя в сеансе редактора или в системе (поддерживается не во всех версиях). В Vim эта опция эквивалентна `-r`.

`-m`

Запуск редактора с отключенной опцией `write`, то есть вы не сможете сохранять файлы. {Vim}

`-M`

Не позволяет изменять текст в файлах. {Vim}

`-n`

Не использует файл подкачки; изменения записываются только в память. {Vim}

`--noplugin`

Не загружать никаких плагинов. {Vim}

`-N`

Запуск Vim в режиме, не совместимом с `vi`. {Vim}

`-o[`*`num`*`]`

Запуск Vim с количеством открытых окон, равным *num*. По умолчанию для каждого файла открывается одно окно. {Vim}

`-O[`*`num`*`]`

Запуск Vim с *num*-количеством открытых окон, упорядоченных на экране горизонтально (вертикальное разделение). {Vim}

`-r [`*`file`*`]`

Режим восстановления. Восстанавливает и возобновляет редактирование файла *file* после прерванного сеанса работы программы или системного сбоя. Без указания *file* выдает список файлов, доступных для восстановления.

`-R`

Редактирует файлы в режиме «только для чтения».

`-s`

«Тихий» режим, то есть приглашения не выводятся. Полезен при запуске скриптов. Это поведение также можно установить более старой опцией `-`. В Vim работает только при использовании с `-e`.

`-s` *`scriptfile`*

Считывает и выполняет команды из заданного файла *scriptfile*, как если бы они вводились с клавиатуры. {Vim}

`-S` *`commandfile`*

После загрузки всех файлов для правки, указанных в командной строке, считывает и выполняет команды из заданного файла *commandfile*. Является сокращенным вариантом команды `vim -c 'source` *`commandfile`*`'`. {Vim}

`-t` *`tag`*

Открывает файл, содержащий тег *tag*, и помещает курсор на его определение.

`-T` *`type`*

Задает опцию типа терминала. Эта опция отменяет значение переменной окружения `$TERM`. {Vim}

`-u` *`file`*

Считывает информацию о конфигурации из указанного файла ресурсов вместо ресурсного файла `.vimrc`, используемого по умолчанию. Если в качестве аргумента *file* стоит `NONE`, Vim не будет загружать никаких ресурсных файлов и плагинов, а запустится в режиме, совместимом с `vi`. Если аргументом выступает `NORC`, то ресурсы не будут загружаться, а плагины – будут.

`-v`

Запускается в полноэкранном режиме (в `vi` установлен по умолчанию).

`--version`

Выдает информацию о версии и выходит из программы. {Vim}

`-V[`*`num`*`]`

Подробный режим. В нем выдаются сообщения о том, какие опции были установлены и какие файлы были прочитаны и сохранены. Можно задавать уровень подробностей, чтобы увеличивать и уменьшать количество получаемых сообщений. По умолчанию значение равно 10 (высокая подробность). {Vim}

`-w` *`rows`*

Устанавливает размер окна таким, чтобы за один раз отображалось *rows* строк. Полезно при редактировании по медленной коммутируемой линии (или при плохом соединении с Интернетом). В старых версиях `vi` пробел между опцией и ее аргументом не допускался. В Vim эта опция не поддерживается.

-W *scriptfile*

Записывает в указанный файл *scriptfile* все команды, набранные в текущем сеансе. Позднее его можно использовать с командой -s. {Vim}

-x

Запрашивает ключ, который будет использоваться при попытках шифрования и дешифрования файла с помощью crypt (поддерживается не во всех версиях).[1]

-y

vi без режима. Опция запускает Vim в режиме вставки без командного режима. Эквивалентна вызову Vim как evim. {Vim}

-Z

Запуск Vim в ограниченном режиме. Вызов команд оболочки и приостановка работы редактора не допускаются. {Vim}

Хотя большинство людей знают команды ex только по их использованию в vi, этот редактор существует и как отдельная программа. Его можно вызвать из оболочки (например, для редактирования файлов, как часть скрипта). Внутри ex можно ввести команду vi или visual для запуска vi. Аналогично в vi можно ввести Q, чтобы выйти из vi и попасть в ex.

Выйти из ex можно несколькими способами:

| | |
|---|---|
| :x | Выход (сохранение изменений и выход). |
| :q! | Выход без сохранения изменений. |
| :vi | Запуск редактора vi. |

# Обзор операций vi

В этом разделе представлен обзор следующих тем:

- Режимы vi
- Синтаксис команд vi
- Команды строки состояния

## Командный режим

После открытия файла вы попадаете в командный режим. Из него можно:

- Выходить в режим вставки
- Давать команды редактирования
- Перемещать курсор по файлу
- Вызывать команды ex

[1] Шифрование командой crypt является слабым. Не пользуйтесь им для хранения важных тайн.

- Выходить в оболочку UNIX
- Сохранять текущую версию файла
- Выходить из vi

## Режим вставки

В режиме вставки можно вводить в файл новый текст. Обычно в этот режим переходят клавишей i. Чтобы выйти оттуда и попасть в командный режим, нажмите ESC. Полный список команд, переводящих в режим вставки, будет представлен позже в разделе «Команды вставки» на стр. 431.

## Синтаксис команд vi

Команды редактирования в vi следуют следующему общему виду:

```
[n] operator [m] motion
```

Основными *операторами* редактирования являются:

| | |
|---|---|
| c | Начать изменение. |
| d | Начать удаление. |
| y | Начать копирование. |

Если в качестве объекта операции выступает текущая строка, то *motion* эквивалентен операторам cc, dd и yy. Если нет, то команды редактирования действуют на объекты, определенные командами перемещения курсора или командами поиска по шаблону. (Например, cf. начнет изменение до следующей точки.) $n$ и $m$ – это количество применений операции и число объектов, на которые распространяется эта операция. Если указаны $n$ и $m$, то всего выполняется $n \times m$ действий.

В качестве объекта, к которому применяется операция, может выступать один из следующих текстовых блоков:

*слово (word)*

Включает символы вплоть до следующего пробельного символа (пробела или табуляции) либо знака препинания. Для объекта, записанного прописными буквами, операция воспринимает в качестве разделителей только пробельные символы.

*Предложение (sentence)*

Текст вплоть до ., ! или ?, за которыми стоят два пробела.

*Абзац (paragraph)*

Текстовый блок до следующей пустой строки или макроса, определенного опцией para=.

*Раздел (Section)*

Текст до заголовка следующего раздела nroff/troff, который определяется опцией sect=.

*Перемещение (Motion)*

Блок до следующего символа или текстового объекта, заданного определителем перемещения, включая поиск по шаблону.

### Примеры

| | |
|---|---|
| `2cw` | Изменение следующих двух слов. |
| `d}` | Удаление до следующего абзаца. |
| `d^` | Удаление назад до начала строки. |
| `5yy` | Копирование следующих пяти строк. |
| `y]]` | Копирование до следующего раздела. |
| `cG` | Изменение до конца буфера редактирования. |

В разделе «Изменение и удаление текста» на стр. 432 этого приложения можно найти еще больше команд и примеров.

### Визуальный режим (только в Vim)

В Vim присутствует дополнительная функция, называемая «визуальным режимом». В нем можно подсвечивать текстовые блоки, которые впоследствии станут объектами действия команд редактирования, таких как удаление и копирование. Графическая версия Vim позволяет аналогичным образом использовать для подсветки текста мышь. Подробности см. выше в разделе «Движение в визуальном режиме» на стр. 197.

| | |
|---|---|
| `v` | При выделении текста в визуальном режиме за один раз выделяется один символ. |
| `V` | При выделении текста в визуальном режиме за один раз выделяется одна строка. |
| CTRL-V | Выделение текста блоками. |

## Команды строки состояния

Большинство команд при вводе не отображаются на экране, однако в строке состояния внизу экрана можно отредактировать эти команды:

| | |
|---|---|
| `/` | Поиск по шаблону вперед. |
| `?` | Поиск по шаблону назад. |
| `:` | Вызов команды `ex`. |
| `!` | Вызов команды UNIX, которая в качестве входа использует объект из буфера и заменяет его выводом команды. Сама команда вводится в строке состояния. |

Команды, вводимые в строке состояния, запускаются клавишей ENTER. Кроме того, в строке состояния отображаются сообщения об ошибках и вывод команды CTRL-G.

# Команды vi

`vi` предоставляет огромное количество одноклавишных команд, доступных в командном режиме, а Vim – дополнительные команды, вызываемые нажатием нескольких клавиш.

## Команды перемещения

В одних версиях `vi` не воспринимаются клавиши расширенной клавиатуры (курсорные, Page Up, Page Down, Home, Insert и Delete), а в других они работают. Однако кнопки, о которых рассказывается в этом разделе, задействованы во всех версиях. Многие пользователи `vi` предпочитают использовать эти клавиши, так как это помогает держать пальцы на среднем ряде клавиатуры. Заметим, что число, стоящее перед командой, означает ее повторение. Команды перемещения также используются после оператора, обрабатывающего текст, по которому перемещается курсор.

### Символы

| | |
|---|---|
| `h`, `j`, `k`, `l` | Налево, вниз, вверх, направо (←, ↓, ↑, →). |
| Пробел (Spacebar) | Направо. |
| BACKSPACE | Налево. |
| CTRL-H | Налево. |

### Текст

| | |
|---|---|
| `w`, `b` | Вперед/назад на «слово» (последовательность, составленная из букв, чисел и знаков подчеркивания). |
| `W`, `B` | Вперед/назад на «СЛОВО» (СЛОВА разделяются пробельными символами). |
| `e` | На конец слова. |
| `E` | На конец СЛОВА. |
| `ge` | На конец предыдущего слова. {Vim} |
| `gE` | На конец предыдущего СЛОВА. {Vim} |
| `)`, `(` | На начало следующего/текущего предложения. |
| `}`, `{` | На начало следующего/текущего абзаца. |
| `]]`, `[[` | На начало следующего/текущего раздела. |
| `][`, `[]` | На конец следующего/текущего раздела. {Vim} |

### Строки

Длинная строка в файле может отображаться в виде нескольких строк (*переноситься* с одной строки экрана на другую). Хотя все команды работают со строками так, как они заданы в файле, несколько команд обрабатывают строки так, как они выглядят на экране. Опция Vim `wrap` позволяет контролировать отображение длинных строк.

| | |
|---|---|
| 0, $ | Первая/последняя позиция текущей строки. |
| ^, _ | Первый непустой символ текущей строки. |
| +, - | Первый непустой символ следующей/предыдущей строки. |
| ENTER | Первый непустой символ следующей строки. |
| *num* \| | Столбец *num* текущей строки. |
| g0, g$ | Первый/последний символ экранной строки. {Vim} |
| g^ | Первый непустой символ экранной строки. {Vim} |
| gm | Середина экранной строки. {Vim} |
| gk, gj | Выше/ниже на одну экранную строку. {Vim} |
| H | Самая верхняя строка экрана. |
| M | Средняя строка экрана. |
| L | Последняя строка экрана. |
| *num* H | *num* строк ниже самой верхней строки. |
| *num* L | *num* строк выше самой нижней строки. |

## Экраны

| | |
|---|---|
| CTRL-F, CTRL-B | Прокрутить вниз/вверх на один экран. |
| CTRL-D, CTRL-U | Прокрутить вниз/вверх на полэкрана. |
| CTRL-E, CTRL-Y | Показать еще одну строку снизу/сверху экрана. |
| z ENTER | Сделать текущую строку самой верхней на экране. |
| z. | Поместить текущую строку в середину экрана. |
| z- | Сделать текущую строку самой нижней на экране. |
| CTRL-L | Перерисовать экран (без прокрутки). |
| CTRL-R | vi: перерисовать экран (без прокрутки).<br>Vim: повторить последнее отмененное действие. |

## Поиск

| | |
|---|---|
| / *pattern* | Искать *pattern* ниже по тексту. Команда заканчивается нажатием ENTER. |
| / *pattern* /+ *num* | Переход на строку *num* после *pattern*. Поиск *pattern* ниже по тексту. |
| / *pattern* /- *num* | Переход на строку *num* раньше *pattern*. Поиск *pattern* вниз по тексту. |
| ? *pattern* | Искать *pattern* вверх по тексту. Команда заканчивается нажатием ENTER. |
| ? *pattern* ?+ *num* | Переход на строку *num* после *pattern*. Поиск *pattern* выше по тексту. |
| ? *pattern* ?- *num* | Переход на строку *num* раньше *pattern*. Поиск *pattern* выше по тексту. |
| :noh | Приостановить подсветку поиска до следующего поиска. {Vim} |

| | |
|---|---|
| `n` | Повторить последний поиск. |
| `N` | Повторить последний поиск в обратном направлении. |
| `/` | Повторить последний поиск вниз по тексту. |
| `?` | Повторить последний поиск вверх по тексту. |
| `*` | Поиск слова под курсором вниз по тексту. Ищется точное соответствие слову. {Vim} |
| `#` | Поиск слова под курсором вверх по тексту. Ищется точное соответствие слову. {Vim} |
| `g*` | Поиск слова под курсором выше по тексту. Ищутся символы этого слова, являющиеся частью более длинного слова. {Vim} |
| `g#` | Поиск слова под курсором вверх по тексту. Ищутся символы этого слова, являющиеся частью более длинного слова. {Vim} |
| `%` | Найти скобку (круглую, квадратную, фигурную), соответствующую данной. |
| `f` *x* | Переместить курсор вперед по строке на символ *x*. |
| `F` *x* | Переместить курсор назад по строке на символ *x*. |
| `t` *x* | Переместить курсор вперед на символ, стоящий в текущей строке перед *x*. |
| `T` *x* | Переместить курсор назад на символ, стоящий в текущей строке после *x*. |
| `,` | Обратить направление поиска последнего `f`, `F`, `t` или `T`. |
| `;` | Повторить последний `f`, `F`, `t` или `T`. |

## Номера строк

| | |
|---|---|
| CTRL-G | Отобразить текущие номера строк. |
| `gg` | Перейти на первую строку в файле. {Vim} |
| *num* `G` | Перейти на строку с номером *num*. |
| `G` | Перейти на последнюю строку в файле. |
| `:` *num* | Перейти на строку с номером *num*. |

## Метки

| | |
|---|---|
| `m` *x* | Поместить метку *x* в текущей позиции. |
| `` ` `` *x* | (Обратная кавычка) Передвинуть курсор на метку *x*. |
| `'` *x* | (Апостроф) Перейти на начало строки, содержащей *x*. |
| ` `` ` | (Обратные кавычки) Перейти на позицию, с которой был сделан самый последний переход. |
| `''` | (Апострофы) То же самое, но возвращает на начало строки. |
| `'"` | (Одинарная и двойная кавычки) Переместить на позицию, где было осуществлено закрытие файла. {Vim} |
| `` `[ ``, `` `] `` | (Обратная кавычка, квадратная скобка) Переместить на начало/конец предыдущей текстовой операции. {Vim} |

| | |
|---|---|
| `'[, ']` | (Апостроф, квадратная скобка) То же самое, но перемещает на начало строки, где производилась операция. {Vim} |
| `` `. `` | (Обратная кавычка, точка) Перейти на последнее изменение файла. {Vim} |
| `'.` | (Апостроф, точка) То же самое, но перемещает на начало строки. {Vim} |
| `'0` | (Апостроф, ноль) Помещает на место последнего выхода из Vim. {Vim} |
| `:marks` | Выводит список активных меток. {Vim} |

## Команды вставки

| | |
|---|---|
| `a` | Приписать текст после курсора. |
| `A` | Приписать текст к концу строки. |
| `c` | Начать операцию изменения. |
| `C` | Изменить текст до конца строки. |
| `gI` | Вставить текст в начало строки[a]. {Vim} |
| `i` | Вставить текст перед курсором. |
| `I` | Вставить текст в начало строки[b]. |
| `o` | Открыть новую строку ниже курсора. |
| `O` | Открыть новую строку выше курсора. |
| `R` | Начать замещение текста. |
| `s` | Заменить символ. |
| `S` | Заменить всю строку. |
| ESC | Выйти из режима вставки. |

[a] В колонке с номером 1. – *Прим. науч. ред.*
[b] Перед первым непробельным символом. – *Прим. науч. ред.*

В режиме вставки работают следующие команды:

| | |
|---|---|
| BACKSPACE | Удаляет предыдущий символ. |
| DELETE | Удаляет текущий символ. |
| TAB | Вставляет табуляцию. |
| CTRL-A | Повторяет последнюю вставку. {Vim} |
| CTRL-D | Сдвигает строку влево к предыдущему `shiftwidth`. {Vim} |
| CTRL-E | Вставляет символ, расположенный ниже курсора. {Vim} |
| CTRL-H | Удаляет предыдущий символ (эквивалентна Backspace). |
| CTRL-I | Вставляет табуляцию. |
| CTRL-K | Начало вставки символа из нескольких нажатий клавиш. |
| CTRL-N | Вставляет следующее завершение шаблона слева от курсора. {Vim} |
| CTRL-P | Вставляет предыдущее завершение шаблона слева от курсора. {Vim} |
| CTRL-T | Сдвигает строку вправо к следующему `shiftwidth`. {Vim} |

| | |
|---|---|
| CTRL-U | Удаляет текущую строку. |
| CTRL-V | Вставляет следующий символ буквально. |
| CTRL-W | Удаляет предыдущее слово. |
| CTRL-Y | Вставляет символ, расположенный выше курсора. {Vim} |
| CTRL-[ | Выход из режима вставки. |

Некоторые из управляющих символов, перечисленных в предыдущей таблице, устанавливаются опцией `stty`. Установки для вашего терминала могут отличаться.

## Команды редактирования

Помните, что базовыми операторами редактирования являются `c`, `d` и `y`.

### Изменение и удаление текста

Следующий список не является исчерпывающим, но он иллюстрирует часто используемые операции:

| | |
|---|---|
| `cw` | Изменить слово. |
| `cc` | Изменить строку. |
| `c$` | Изменить текст от текущей позиции курсора до конца строки. |
| `C` | Команда эквивалентна `c$`. |
| `dd` | Удалить текущую строку. |
| *num* `dd` | Удалить *num* строк. |
| `d$` | Удалить текст от текущей позиции курсора до конца строки. |
| `D` | Действие аналогично `d$`. |
| `dw` | Удалить слово. |
| `d}` | Удалить текст до следующего абзаца. |
| `d^` | Удалить предыдущий текст до начала строки. |
| `d/` *pat* | Удалить текст вплоть до первого вхождения шаблона. |
| `dn` | Удалить текст до следующего вхождения шаблона. |
| `df` *x* | Удалить текст до символа *x* в текущей строке включительно. |
| `dt` *x* | Удалить текст до символа *x* в текущей строке (исключая сам символ). |
| `dL` | Удалить текст до последней строки на экране. |
| `dG` | Удалить текст до конца файла. |
| `gqap` | Переформатировать текущий абзац на `textwidth`. {Vim} |
| `g~w` | Сменить регистр слова. {Vim} |
| `guw` | Сделать буквы слова строчными. {Vim} |
| `gUw` | Сделать буквы слова прописными. {Vim} |
| `p` | Вставить последний удаленный или скопированный текст после курсора. |
| `gp` | Команда аналогична `p`, но помещает курсор в конец вставленного текста. {Vim} |

| | |
|---|---|
| gP | Аналогична P, но помещает курсор в конец вставленного текста. {Vim} |
| ]p | Аналогична p, но следит за текущими установками отступов. {Vim} |
| [p | Аналогична P, но следит за текущими установками отступов. {Vim} |
| P | Вставить последний удаленный или скопированный текст перед курсором. |
| r *x* | Меняет символ на *x*. |
| R *text* | Замещает текст новым, начиная с позиции курсора. ESC выходит из режима замещения. |
| s | Подставляет символ. |
| 4s | Подставляет четыре символа. |
| S | Подставляет всю строку. |
| u | Отменяет последнее изменение. |
| CTRL-R | Повторяет последнее изменение. {Vim} |
| U | Восстанавливает текущую строку. |
| x | Удаляет символ в текущей позиции курсора. |
| X | Удаляет один предшествующий символ. |
| 5X | Удаляет пять предшествующих символов. |
| . | Повторяет последнее изменение. |
| ~ | Меняет регистр и перемеащает курсор вправо. |
| CTRL-A | Увеличивает на 1 число, на котором стоит курсор. {Vim} |
| CTRL-X | Уменьшает на 1 число, на котором стоит курсор. {Vim} |

## Копирование и перемещение

Имена регистров – буквы от a до z. Прописные буквы добавляют текст к соответствующему регистру.

| | |
|---|---|
| Y | Копировать текущую строку. |
| yy | Эквивалентна Y. |
| " *x* yy | Копировать текущую строку в регистр *x*. |
| ye | Копировать текст до конца слова. |
| yw | Как ye, но включает пробельный символ после слова. |
| y$ | Копировать остаток строки. |
| " *x* dd | Удалить текущую строку в регистр *x*. |
| " *x* d | Удалить текст в регистр *x*. |
| " *x* p | Вставить содержимое регистра *x*. |
| y]] | Копировать текст до заголовка следующего раздела. |
| J | Объединить текущую строку со следующей. |
| gJ | Аналогична J, но без вставки пробела. {Vim} |
| :j | Эквивалентна J. |
| :j! | Аналогична gJ. |

## Сохранение и выход

| | |
|---|---|
| ZZ | Выход из vi с записью только в случае, если были сделаны изменения. |
| :x | Эквивалентна ZZ. |
| :wq | Записать файл и выйти. |
| :w | Записать файл. |
| :w *file* | Записать в файл *file*. |
| :*n* , *m* w *file* | Записать в новый файл *file* строки с *n* по *m*. |
| :*n* , *m* w >> *file* | Приписать строки с *n* по *m* к существующему файлу *file*. |
| :w! | Записать файл (невзирая на защиту). |
| :w! *file* | Перезаписать файл *file* текущим текстом. |
| :w %. *new* | Записать текущий буфер с именем *file* как *file.new*. |
| :q | Выйти из vi (не сработает, если были сделаны изменения). |
| :q! | Выйти из vi (потеряв все изменения). |
| Q | Выйти из vi и запустить ex. |
| :vi | Вернуться в vi после команды Q. |
| % | Заменяется в командах редактирования на имя текущего файла. |
| # | Заменяется в командах редактирования на имя альтернативного файла. |

## Доступ к нескольким файлам

| | |
|---|---|
| :e *file* | Редактировать другой файл *file*, при этом текущий файл становится альтернативным. |
| :e! | Вернуться к версии файла последнего сохранения. |
| :e + *file* | Начать редактирование с конца файла *file*. |
| :e +*num file* | Открыть файл *file* на строке *num*. |
| :e # | Открыть на предыдущей позиции альтернативного файла. |
| :ta *tag* | Редактировать файл на позиции *tag*. |
| :n | Редактировать следующий файл в списке файлов. |
| :n! | Вынужденный переход к следующему файлу. |
| :n *files* | Указать новый список файлов *files*. |
| :rewind | Редактировать первый файл в списке файлов. |
| CTRL-G | Показать текущий файл и номер строки. |
| :args | Вывести список редактируемых файлов. |
| :prev | Редактировать предыдущий файл в списке файлов. |

## Команды работы с окнами (Vim)

В следующей таблице представлен список основных команд для управления окнами в Vim. См. также команды split, vsplit и resize в разделе

«Алфавитный перечень команд ex» на стр. 440. Для краткости управляющие символы в списке обозначены знаком ^.

| | |
|---|---|
| `:new` | Открыть новое окно. |
| `:new` *file* | Открыть файл *file* в новом окне. |
| `:sp` [*file*] | Разделить текущее окно. Если указан *file*, этот файл открывается в новом окне для редактирования. |
| `:sv` [*file*] | Команда эквивалентна `:sp`, но новое окно создается только для чтения. |
| `:sn` [*file*] | В новом окне открывается для редактирования файл *file*, следующий в списке файлов. |
| `:vsp` [*file*] | Аналогична `:sp`, но разделение вертикальное, а не горизонтальное. |
| `:clo` | Закрыть текущее окно. |
| `:hid` | Скрыть текущее окно, если только оно не является последним видимым окном. |
| `:on` | Сделать текущее окно единственным видимым. |
| `:res` *num* | Установить размер окна равным *num* строк. |
| `:wa` | Записать все измененные буферы в свои файлы. |
| `:qa` | Закрыть все буферы и выйти. |
| `^W s` | Эквивалентна `:sp`. |
| `^W n` | Аналогична `:new`. |
| `^W ^` | Открыть новое окно с альтернативным (ранее редактировавшимся) файлом. |
| `^W c` | Аналогична `:clo`. |
| `^W o` | Аналогична `:only`. |
| `^W j`, `^W k` | Перенести курсор в окно, расположенное на экране ниже/выше текущего. |
| `^W p` | Перенести курсор в предыдущее окно. |
| `^W h`, `^W l` | Перенести курсор в окно, расположенное на экране слева/справа от текущего. |
| `^W t`, `^W b` | Перенести курсор в окно, расположенное на экране выше/ниже всех. |
| `^W K`, `^W B` | Переместить текущее окно вверх/вниз экрана. |
| `^W H`, `^W L` | Переместить текущее окно в левый/правый край экрана. |
| `^W r`, `^W R` | Провернуть окна вниз/вверх. |
| `^W +`, `^W -` | Увеличить/уменьшить размер текущего окна. |
| `^W =` | Выровнять по размеру все окна. |

## Взаимодействие с системой

| | |
|---|---|
| `:r` *file* | Считать содержимое файла *file* после курсора. |
| `:r !`*command* | Считать вывод от команды *command* после текущей строки. |
| `:` *num* `r !`*command* | Аналогична предыдущей команде, но вывод помещается после строки *num* (для самого начала файла нужно взять 0). |

| | |
|---|---|
| `:!`*command* | Выполнить команду и вернуться в `vi`. |
| `!`*motion command* | Передать текст, определенный через *motion*, в команду UNIX *command*; заменить текст выводом этой команды. |
| `:` *n* `,` *m* `!`*command* | Передать в *command* строки с *n* по *m* и заменить их выводом. |
| *num*`!!`*command* | Передать в *command num* строк и заменить их выводом. |
| `:!!` | Повторить последнюю системную команду. |
| `:sh` | Создать подоболочку; *EOF* вернет в редактор. |
| CTRL-Z | Приостановить редактор (возобновить работу можно с помощью `fg`). |
| `:so` *file* | Считать и выполнить команды `ex` из файла *file*. |

## Макросы

| | |
|---|---|
| `:ab` *in out* | Использовать *in* как аббревиатуру для *out* в режиме вставки. |
| `:unab` *in* | Удалить аббревиатуру для *in*. |
| `:ab` | Вывести список аббревиатур. |
| `:map` *string sequence* | Отобразить строку символов *string* на последовательность команд *sequence*. Функциональные клавиши обозначаются как `#1`, `#2` и т. д. |
| `:unmap` *string* | Убрать отображение для строки *string*. |
| `:map` | Вывести список всех строк, использующихся при отображениях. |
| `:map!` *string sequence* | Отобразить строку символов *string* на последовательность *sequence* в режиме ввода. |
| `:unmap!` *string* | Удалить отображение для режима ввода (возможно, вам потребуется экранировать символы с помощью CTRL-V). |
| `:map!` | Вывести список всех строк, использующихся при отображениях для режима ввода. |
| `q` *x* | Записать вводимые символы в регистр, заданный буквой *x*. Если буква прописная, ввод будет добавляться к содержимому регистра. {Vim} |
| `q` | Остановить запись. {Vim} |
| `@` *x* | Выполнить регистр, определенный символом *x*. Для повторения последней команды `@` используется `@@`. |

`vi` не использует следующие символы в командном режиме, поэтому их можно применять при отображении пользовательских команд:

*Буквы*

`g`, `K`, `q`, `V` и `v`

*Управляющие клавиши*

`^A`, `^K`, `^O`, `^W`, `^X`, `^_` и `^\`

*Символы*

`_`, `*`, `\`, `=` и `#`

Знак = используется в vi, если установлен режим Lisp. Разные версии редактора могут применять некоторые из этих символов, так что перед их использованием лучше их проверить.

Vim не использует ^K, ^_, _ или \.

## Разные команды

| | |
|---|---|
| < | Сдвиг влево на одну единицу shiftwidth текста, определяемого последующей командой перемещения. {Vim} |
| > | Сдвиг вправо на одну единицу shiftwidth текста, определяемого последующей командой перемещения. {Vim} |
| << | Сдвиг строки влево на один shiftwidth (восемь символов по умолчанию). |
| >> | Сдвиг строки вправо на один shiftwidth (восемь символов по умолчанию). |
| >} | Сдвиг вправо до конца абзаца. |
| <% | Сдвиг влево до соответствующей круглой, фигурной или квадратной скобки. (Курсор должен стоять на подходящем символе.) |
| == | Сделать отступ строки в стиле C либо с использованием программы, указанной в опции equalprg. {Vim} |
| g | Начало многих многосимвольных команд в Vim. |
| K | Найти на странице man (или в программе, определенной в keywordprg) слово, на котором стоит курсор. {Vim} |
| ^O | Вернуться к предыдущему переходу. {Vim} |
| ^Q | Эквивалентна ^V. {Vim} (В некоторых терминалах это возобновление потока данных.) |
| ^T | Возврат к предыдущему положению в стеке тегов (Solaris vi, Vim, nvi, elvis и vile). |
| ^] | Произвести поиск текста под курсором по тегу. |
| ^\ | Войти в режим строкового редактора ex. |
| ^^ | (Клавиша каретки с нажатой клавишей Ctrl.) Вернуться к предыдущему редактировавшемуся файлу. |

# Конфигурация vi

В этом разделе описывается следующее:

- Команда :set
- Опции для :set
- Примерный файл .exrc

## Команда :set

Команда :set позволяет задавать опции, меняющие характеристики среды редактирования. Опции можно поместить в файл .exrc или задать в сеансе работы vi.

Если команда находится в .exrc, то двоеточие можно не ставить:

| | |
|---|---|
| :set *x* | Включить булевскую опцию *x*; показать значение других опций. |
| :set no *x* | Выключить булевскую опцию *x*. |
| :set *x* = *value* | Задать опции *x* значение *value*. |
| :set | Показать измененные опции. |
| :set all | Показать все опции. |
| :set *x* ? | Показать значение опции *x*. |

В приложении B даны таблицы опций для Solaris vi, Vim, nvi, elvis и vile. За подробностями обратитесь к этому приложению.

## Пример файла .exrc

В файле скриптов ex символ двойной кавычки начинает комментарий. Следующие строки кода являются примером пользовательского файла .exrc:

```
set nowrapscan                    " Searches don't wrap at end of file
set wrapmargin=7                  " Wrap text at 7 columns from right margin
set sections=SeAhBhChDh nomesg    " Set troff macros, disallow message
map q :w^M:n^M                    " Alias to move to next file
map v dwElp                       " Move a word
ab ORA O'Reilly Media, Inc.       " Input shortcut
```

Псевдоним q в Vim не нужен, так как в нем есть команда :wn. Псевдоним v перекроет команду v из Vim, которая начинает операцию посимвольного выделения в визуальном режиме.

# Основы ex

Строковый редактор ex служит основой для экранного редактора vi. Команды ex действуют на текущую строку или на диапазон строк из файла. Зачастую ex используется из vi. В этом редакторе перед командами ex нужно ставить двоеточие, а их выполнение начинается по нажатию ENTER.

ex можно запустить и сам по себе – из командной строки, аналогично vi. (Таким же образом можно запустить скрипт ex.) В vi можно использовать команду Q, чтобы выйти из программы и попасть в ex.

## Синтаксис команд ex

Чтобы ввести команду ex из vi, наберите:

```
:[address] command [options]
```

Начальное : указывает, что это команда ex. При вводе команда отображается в строке состояния. Чтобы выполнить ее, нажмите ENTER.

*Address* – это номер строки или диапазон строк, которые являются объектом для команды *command*. Опции *options* и адреса *addresses* обсуждаются ниже. Команды ex описываются в разделе «Алфавитный перечень команд ex» на стр. 440.

Выйти их ex можно несколькими способами:

| | |
|---|---|
| :x | Выход с сохранением изменений. |
| :q! | Выход без сохранения изменений. |
| :vi | Переход в редактор vi с текущим файлом. |

## Адреса

Если адрес не указан, объектом команды выступает текущая строка. Если адрес указывает диапазон строк, то он имеет следующий формат:

```
x,y
```

где *x* и *y* – первая и последняя адресуемые строки (в буфере *x* должна находиться перед *y*). Каждая из *x* и *y* может быть как номером строки, так и символом. Если использовать точку с запятой (;) вместо запятой (,), то текущая строка будет установлена на *x* перед интерпретацией *y*. Запись 1,$ обозначает все строки в файле аналогично %.

## Адреса строк

| | |
|---|---|
| 1,$ | Все строки файла. |
| *x* , *y* | Строки с *x* по *y*. |
| *x* ; *y* | Строки с *x* по *y*, но текущая строка устанавливается на *x*. |
| 0 | Начало файла. |
| . | Текущая строка. |
| *num* | Абсолютный номер строки *num*. |
| $ | Последняя строка. |
| % | Все строки. Эквивалентно 1,$. |
| *x* - *n* | *n* строк перед *x*. |
| *x* + *n* | *n* строк после *x*. |
| -[*num*] | Одна или *num* строк выше по тексту. |
| +[*num*] | Одна или *num* строк далее по тексту. |
| ' *x* | (Апостроф) Строка с меткой *x*. |
| '' | (Два апострофа) Предыдущая метка. |
| /*pattern*/ | Вперед до строки, отвечающей шаблону. |
| ?*pattern*? | Назад до строки, отвечающей шаблону. |

Больше информации об использовании шаблонов можно найти в главе 6.

## Опции

**!**

Указывает на вариантную форму команды, что отменяет ее обычное поведение. Знак `!` должен стоять сразу после команды.

*count*

Число повторений команды. В отличие от команд `vi`, *count* не может стоять перед командой, так как номер, стоящий перед командой, рассматривается как адрес строки. Например, `d3` удалит три строки, начиная с текущей, а `3d` удалит строку 3.

*file*

Имя файла, на который действует команда. `%` означает текущий файл, а `#` – предыдущий.

# Алфавитный перечень команд ex

Команды `ex` можно ввести, задав любую уникальную аббревиатуру. В следующем списке-справочнике в качестве заголовка используется полное название, а в строке синтаксиса ниже – самая короткая из возможных аббревиатур команды. В примерах предполагается, что команда вводится из `vi`, то есть приглашение `:` уже содержится.

---

## abbreviate

ab [*string text*]

Определяет строку *string*, которая будет преобразовываться в текст *text* при наборе. Если строка *string* и текст *text* не заданы, выведется список всех аббревиатур.

### Примеры

Обратите внимание: `^M` появляется, если ввести `^V`, а затем ENTER.

```
:ab ora O'Reilly Media, Inc.
:ab id Name:^MRank:^MPhone:
```

---

## append

[*address*] a[!]
*text*
.

Приписывает новый текст *text* к указанному адресу *address* либо к текущему адресу, если *address* не указан. Чтобы переключить установку `autoindent`, используемую при вводе, добавьте знак `!`. Таким образом, если `autoindent` включена, `!` выключает ее. После ввода команды набирайте новый текст. Его ввод заканчивается после ввода строки, содержащей одну точку.

**Пример**

```
:a                       Начать приписывание к текущей строке
Append this line
and this line too.
.                        Закончить ввод приписываемого текста
```

## args

```
ar
args file . . .
```

Выводит пункты списка аргументов (файлов, указанных в командной строке). Текущий файл выводится в квадратных скобках ([]).

Второй синтаксис используется в Vim, где позволяет переопределить список редактируемых файлов.

## bdelete

`[num] bd[!] [num]`

Выгружает буфер *num* и удаляет его из списка буферов. Для удаления несохраненного буфера добавьте `!`. Буфер также можно указывать по имени файла. Если он не указан, выгружается текущий. {Vim}

## buffer

`[num] b[!] [num]`

Начать редактировать буфер *num* из списка буферов. Чтобы переключиться в него из несохраненного буфера, добавьте `!`. Буфер также можно указывать по имени файла. Если он не указан, будет продолжено редактирование текущего буфера.

## buffers

`buffers[!]`

Выводит все элементы списка буферов. Некоторые буферы (например, удаленные) не будут выводиться. Для их отображения используйте `!`. Другим сокращением этой команды является `ls`. {Vim}

## cd

```
cd dir
chdir dir
```

Меняет текущий каталог для редактора на *dir*.

## center

[*address*] ce [*width*]

Центрирует строку по указанной ширине *width*. Если ширина не задана, используется textwidth. {Vim}

## change

[*address*] c[!]
*text*
.

Указанные строки заменяются на текст *text*. Чтобы переключить установку autoindent во время ввода текста, используйте !. Набор нового текста заканчивается после ввода строки, содержащей одну точку.

## close

clo[!]

Закрывает текущее окно, если только оно не последнее. Если буфер окна не открыт в другом окне, он выгружается из памяти. Эта команда не будет выгружать несохраненный буфер, а добавление ! приведет к его скрытию. {Vim}

## copy

[*address*] co *destination*

Копирует строки, заданные *address*, в указанный адрес *destination*. Команда t (сокращение от «to») является синонимом для copy.

### Пример

| | |
|---|---|
| :1,10 co 50 | *Копирует первые 10 строк непосредственно под 50-ю строку* |

## delete

[*address*] d [*register*] [*count*]

Удаляет строки, заданные в *address*. Если задан *register*, то текст сохраняется или добавляется в именованный регистр. Имена регистров — строчные буквы от a до z. Прописные имена регистров приведут к добавлению текста. Если указан *count*, удаление будет выполняться указанное в *count* число раз.

### Примеры

| | |
|---|---|
| :/Part I/,/Part II/-1d | *Удалить текст до строки выше «Part II»* |
| :/main/+d | *Удалить строку ниже «main»* |
| :.,$d x | *Удалить текст от текущей строки до последней в регистр x* |

## edit

`e[!] [+num] [filename]`

Начать редактирование файла *filename*. Если *filename* не задан, начинается редактирование копии текущего файла. Чтобы начать редактирование нового файла, несмотря на то, что в текущем есть несохраненные изменения, используйте `!`. Аргумент +*num* указывает, что правку нужно начать на строке *num*. В качестве *num* может выступать шаблон в форме /*pattern*.

### Примеры

| | |
|---|---|
| `:e file` | *Редактирует файл в текущем буфере программы* |
| `:e +/^Index #` | *Редактирует альтернативный файл на строке, соответствующей шаблону* |
| `:e!` | *Начать редактирование текущего файла заново* |

## file

`f [filename]`

Изменение имени файла текущего буфера на *filename*. При следующем сохранении буфер будет записан в файл *filename*. При смене имени устанавливается флаг состояния буфера «not edited», указывающий, что вы не редактируете существующий файл. Если новое имя *filename* совпадает с именем уже существующего файла, введите `:w!` для перезаписи существующего файла. При указании имени файла можно использовать символ `%`, обозначающий имя текущего файла. Для имени альтернативного файла используется `#`. Если *filename* не указан, то будет выведено имя текущего файла и состояние буфера.

### Пример

```
:f %.new
```

## fold

`address fo`

Сворачивает строки, указанные в *address*. Свертка сжимает несколько строк на экране в одну, которую позже можно будет развернуть. Текст файла при этом не меняется. {Vim}

## foldclose

`[address] foldc[!]`

Закрывает свертки на указанном *address* или, если он не указан, по текущему адресу. Для закрытия нескольких уровней сверток используйте `!`. {Vim}

## foldopen

`[address] foldo[!]`

Открывает свертки на указанном *address* либо, если адрес не прописан, по текущему адресу. Для открытия нескольких уровней сверток используйте `!`. {Vim}

## global

`[address] g[!]/pattern/[commands]`

Выполняет команды *commands* на всех строках, содержащих *pattern*, или, если указан *address*, на всех строках, содержащих *pattern*, в заданном диапазоне. Если *commands* не задано, выводит на печать все эти строки. Чтобы выполнить *commands* в строках, не содержащих *pattern*, используйте `!`. См также `v` ниже в этом списке.

### Примеры

| | |
|---|---|
| `:g/Unix/p` | *Выводит все строки, содержащие «Unix»* |
| `:g/Name:/s/tom/Tom/` | *Меняет «tom» на «Tom» во всех строках, содержащих «Name:»* |

## hide

`hid`

Закрывает текущее окно, если только оно не последнее, но не удаляет буфер из памяти. Эту команду можно без опасений применять к несохраненному буферу. {Vim}

## insert

```
[address] i[!]
text
.
```

Вставляет *text* в строку, расположенную перед указанным *address*, или по текущему адресу, если *address* не указан. Чтобы переключить установку `autoindent` во время набора текста, используйте `!`. Ввод нового текста заканчивается после ввода строки, содержащей одну точку.

## join

`[address] j[!] [count]`

Помещает текст из указанного диапазона в одну строку. При этом пробелы расставляются так, чтобы после точки (.) стояло два пробела, ни одного пробела перед `)` и один пробел в остальных случаях. Чтобы не менять пробелы, используйте `!`.

#### Пример

`:1,5j!` *Объединяет первые пять строк, сохраняя пробелы*

### jumps

`ju`

Выводит список переходов для команд CTRL-I и CTRL-O. Он содержит запись большинства команд перемещения, которые переносят более чем на одну строку. Позиция курсора перед каждым переходом записывается.

### k

`[`*`address`*`] k `*`char`*

Команда эквивалентна `mark`; см. `mark` ниже в этом списке.

### left

`[`*`address`*`] le [`*`count`*`]`

Строки, определенные в *address*, или текущая строка, если *address* не задан, выравниваются по левому краю. Ставится отступ из *count* пробелов. {Vim}

### list

`[`*`address`*`] l [`*`count`*`]`

Выводит указанные строки так, чтобы табуляции отображались в виде `^I`, а концы строк – как `$`. `l` работает как временная версия `:set list`.

### map

`map[!] [`*`string commands`*`]`

Определяет клавиатурный макрос под именем *string* в качестве указанной последовательности команд. Обычно *string* – это один символ или строка *#num*, причем последняя соответствует функциональной клавише клавиатуры. Чтобы создать макрос для режима ввода, используется `!`. Если не задать аргументы, выведется список уже определенных макросов.

#### Примеры

| | |
|---|---|
| `:map K dwwP` | *Перестановка двух слов* |
| `:map q :w^M:n^M` | *Запись текущего файла и переход к следующему* |
| `:map! + ^[bi(^[ea)` | *Заключить предыдущее слово в круглые скобки* |

В Vim есть команды `K` и `q`, которые будут заменены на псевдонимы из примеров.

## mark

[*address*] ma *char*

Отметить указанную строку *char* одной строчной буквой. Действие эквивалентно k. Возврат к этой строке — 'x (апостроф и *x*, где действие *x* аналогично *char*). В Vim также используются прописные и цифровые символы для меток. Строчные буквы работают так же, как в vi, а прописные связываются с файлами и используются при работе с несколькими файлами. Цифровые метки, однако, поддерживаются в специальном файле viminfo, и их нельзя задать этой командой.

## marks

marks [*chars*]

Выводит список меток, определенных *chars*, либо все текущие метки, если *chars* не указаны. {Vim}

### Пример

| | |
|---|---|
| :marks abc | *Выводит метки a, b и c* |

## mkexrc

mk[!] *file*

Создает файл .exrc, содержащий команды set для измененных опций ex и отображений клавиш. Текущие установки опций будут сохранены, поэтому вы сможете восстановить их позже. {Vim}

## move

[*address*] m *destination*

Перемещает указанные в *address* строки по адресу *destination*.

### Пример

| | |
|---|---|
| :.,/Note/m /END/ | *Перемещает текстовый блок на место после строки, содержащей «END»* |

## new

[*count*] new

Создает новое окно высотой *count* строк с пустым буфером. {Vim}

## next

n[!] [[+*num*] *filelist*]

Редактирует следующий файл из списка аргументов командной строки. Для получения этого списка используется *args*. Если задан *filelist*,

текущий список аргументов будет заменен на *filelist*, а редактирование начнется в первом файле из нового списка. При использовании аргумента +*num* редактирование начнется на строке *num*. *num* может быть шаблоном, представленным как /*pattern*.

#### Пример

`:n chap*` *Начать редактирование всех файлов «chapter»*

### nohlsearch

`noh`

Временно прекращает подсветку всех соответствий шаблону поиска при использовании опции `hlsearch`. Подсветка восстанавливается при следующем поиске. {Vim}

### number

[*address*] nu [*count*]

Каждая строка, определенная в *address*, выводится с номером строки в буфере. В качестве альтернативного сокращения для `number` используется `#`. *count* указывает количество показываемых строк, начиная с *address*.

### only

`on [!]`

Делает текущее окно единственным на экране. Открытые окна с несохраненными буферами не удаляются с экрана (скрываются), если только вы не используете символ `!`. {Vim}

### open

[*address*] o [/*pattern*/]

Вход в открытый режим (`vi`) на строках, определенных по *address*, или строках, отвечающих шаблону *pattern*. Выход из открытого режима происходит по клавише `Q`. Этот режим позволяет использовать обычные команды `vi`, но только для одной строки за раз. Может оказаться полезным на медленных коммутируемых линиях (или при использовании сильно удаленных соединений по `ssh`).

### preserve

`pre`

Сохраняет текущий буфер редактора, как если бы система была близка к сбою.

## previous

`prev[!]`

Редактирует предыдущий файл из списка аргументов командной строки. {Vim}

## print

[*address*] p [*count*]

Выводит строки, определенные по *address*. *count* указывает количество выводимых строк, начиная с *address*. Другое сокращение – P.

### Пример

| | |
|---|---|
| :100;+5p | *Показать 100-ю строку и следующие за ней 5 строк* |

## put

[*address*] pu [*char*]

Вставляет ранее удаленные или скопированные строки из именованного регистра, определенного по *char*, в строку, определенную по *address*. Если *char* не указан, восстанавливается последняя удаленная или скопированная строка.

## qall

`qa[!]`

Закрывает все окна и завершает сеанс редактирования. Для отказа от всех изменений с момента последнего сохранения используйте !. {Vim}

## quit

`q[!]`

Завершает текущий сеанс редактирования. Чтобы отказаться от всех изменений с момента последнего сохранения, нажмите !. Если сеанс содержит дополнительные файлы в списке аргументов, к которым ни разу не обращались, выход осуществляется с помощью q! или двух q. Vim закрывает окно редактирования, только если на экране есть другие открытые окна.

## read

[*address*] r *filename*

Копирует текст из файла *filename* после строки, определенной в *address*. Если *filename* не указан, используется имя текущего файла.

**Пример**

`:0r $HOME/data` *Считать файл в начало текущего файла*

---

## read

`[address] r !command`

Считывает вывод команды оболочки *command* в текст после строки, определенной в *address*.

**Пример**

`:$r !spell %` *Поместить в конец файла результат проверки орфографии*

---

## recover

`rec [file]`

Восстанавливает файл *file* из области системного сохранения.

---

## redo

`red`

Восстанавливает последнее отмененное изменение. Команда аналогична CTRL-R. {Vim}

---

## resize

`res [[±]num]`

Меняет высоту текущего окна, устанавливая ее равной *num* строк. Если указаны `+` или `-`, высота увеличивается или уменьшается на *num* строк. {Vim}

---

## rewind

`rew[!]`

Возвращается к первому файлу из списка аргументов и начинает его редактирование. Если текущий файл не сохранялся с момента последнего изменения, то для возврата к первому файлу нужно использовать `!`.

---

## right

`[address] ri [width]`

Строки, определенные в *address* (или текущая строка, если *address* не задан), выравниваются по правому краю ширины *width*. Если *width* не задана, используется опция `textwidth`. {Vim}

## sbnext

[*count*] sbn [*count*]

Разделяет текущее окно и начинает редактировать буфер под номером *count* в списке буферов. Если *count* не указан, редактируется следующий буфер из списка. {Vim}

## sbuffer

[*num*] sb [*num*]

Разделяет текущее окно и в новом окне начинает редактировать буфер под номером *num* в списке буферов. Редактируемый буфер можно определить также по имени файла. Если буфер не указан, в новом окне открывается текущий буфер. {Vim}

## set

se *parameter1 parameter2* . . .

Устанавливает значение опции для каждого *parameter* или, если ни один *parameter* не указан, выводит все опции, значения которых были изменены. Для булевских опций каждый *parameter* можно трактовать как *option* или *nooption*. Опциям других типов можно задать значения синтаксисом *option*=*value*. Чтобы вывести текущие установки, необходимо указать all. Форма вида set *option*? отображает значение опции. Таблицы, где перечислены списки опций, можно найти в приложении B.

### Примеры

```
:set nows wm=10
:set all
```

## shell

sh

Создает новую оболочку. После выхода из нее редактирование возобновляется.

## snext

[*count*] sn [[+*num*] *filelist*]

Разделяет текущее окно и начинает редактирование файла, следующего в списке аргументов командной строки. Если указан *count*, редактируется *count*-й следующий файл в списке. Если задан *filelist*, то текущий список аргументов заменяется на *filelist*, после чего начинается редактирование первого файла оттуда. При наличии аргумента +*num* работа начинается со строки *num*. Кроме того, *num* может быть шаблоном вида /*pattern*. {Vim}

## source

`so` *file*

Считывает и выполняет команды `ex` из файла *file*.

### Пример

```
:so $HOME/.exrc
```

## split

[*count*] `sp` [+*num*] [*filename*]

Разделяет текущее окно и загружает в новое окно файл *filename* или текущий буфер, если *filename* не задан. Высота нового окна задается равной *count* или, если *count* не указан, окно разделяется на равные части. При наличии аргумента +*num* редактирование начинается со строки *num*. Кроме того, *num* может быть шаблоном вида /*pattern*. {Vim}

## sprevious

[*count*] `spr` [+*num*]

Разделяет текущее окно и начинает редактирование предыдущего файла в списке аргументов командной строки. Если указан *count*, редактируется *count*-й предыдущий файл в списке. При наличии аргумента +*num* редактирование начинается со строки *num*. Кроме того, *num* может быть шаблоном вида /*pattern*. {Vim}

## stop

`st`

Приостанавливает текущий сеанс редактирования. Действие аналогично CTRL-Z. Для восстановления сеанса введите команду оболочки `fg`.

## substitute

[*address*] `s` [/*pattern*/*replacement*/] [*options*] [*count*]

Заменяет первое вхождение *pattern* в каждой из указанных строк на *replacement*. Если *pattern* и *replacement* не указаны, повторяет последнюю замену. *count* определяет количество строк, в которых нужно производить замену, начиная с *address*. (Несокращенная форма команды не работает в Solaris `vi`.)

### Опции

- `c` При каждой замене запрашивает подтверждение.
- `g` Заменяет все вхождения *pattern* в каждой строке (глобальная замена).
- `p` Выводит последнюю строку, в которой была сделана замена.

### Примеры

| | |
|---|---|
| `:1,10s/yes/no/g` | *Заменить текст в первых 10 строках* |
| `:%s/[Hh]ello/Hi/gc` | *Глобальная замена с подтверждением* |
| `:s/Fortran/\U&/ 3` | *Сделать в следующих трех строках прописные буквы у «Fortran»* |
| `:g/^[0-9][0-9]*/s//Line &:/` | *Каждой строке, начинающейся с одной или нескольких цифр, приписать «Line» и двоеточие* |

## suspend

`su`

Приостанавливает текущий сеанс редактирования. Команда аналогична CTRL-Z. Для восстановления сеанса введите команду оболочки `fg`.

## sview

[*count*] sv [+*num*] [*filename*]

Эквивалентна команде `split`, но для нового буфера устанавливается опция `readonly`. {Vim}

## t

[*address*] t *destination*

Копирует строки, содержащиеся в *address*, в (to) указанный адрес *destination*. `t` эквивалентна `copy`.

### Пример

| | |
|---|---|
| `:%t$` | *Копирует файл и добавляет его в конец* |

## tag

[*address*] ta *tag*

В файле `tags` находит файл и строку, соответствующую тегу *tag*, после чего начинает редактирование оттуда.

### Пример

Запустите `ctags`, затем переключитесь в файл, содержащий *myfunction*:

```
:!ctags *.c
:tag myfunction
```

## tags

`tags`

Выводит список тегов из стека тегов. {Vim}

## unabbreviate

`una` *word*

Убрать слово *word* из списка аббревиатур.

## undo

`u`

Отменить изменения, проделанные последней командой редактирования. В `vi` команда отмены отменяет саму себя, таким образом производя восстановление. Vim поддерживает несколько уровней отмены. Для повторного применения отмененного действия в Vim используйте `redo`.

## unhide

[*count*] `unh`

Разделяет экран так, чтобы в каждом окне был показан активный буфер из списка буферов. Если указан *count*, то он ограничивает количество окон. {Vim}

## unmap

`unm[!]` *string*

Удалить строку *string* из списка макросов клавиатуры. Для удаления макроса режима ввода используется `!`.

## v

[*address*] `v/`*pattern*`/`[*command*]

Выполняет команду *command* для всех строк, не содержащих *pattern*. Если *command* не указана, эти строки выводятся. `v` эквивалентна `g!`. См `global` ранее в этом списке.

### Пример

| | |
|---|---|
| `:v/#include/d` | *Удалить все строки, за исключением строк с «#include»* |

## version

`ve`

Выводит текущую версию редактора и дату последнего изменения.

## view

`vie[[+`*num*`]` *filename*`]`

Эквивалентна команде `edit`, но для файла устанавливается опция `readonly`. Если выполнить в режиме `ex`, возвращает в нормальный или визуальный режим. {Vim}

### visual

`[address] vi [type] [count]`

Запускает визуальный режим (vi) на строке, указанной в *address*. Вернуться в режим ex можно, нажав Q. *type* заменяется на -, ^ или . (см. команду z ниже в этом разделе).

### visual

`vi [+num] file`

Начать редактирование файла в визуальном режиме (vi), опционально – на строке *num*. Кроме того, *num* может быть шаблоном вида */pattern*. {Vim}

### vsplit

`[count] vs [+num] [filename]`

Аналогична команде split, но экран разделяется вертикально. Аргумент *count* может использоваться для указания ширины нового окна. {Vim}

### wall

`wa[!]`

Записывает все измененные буферы в файлы. Для записи буферов, помеченных как readonly, используйте !. {Vim}

### wnext

`[count] wn[!] [[+num] filename]`

Сохраняет текущий буфер и открывает следующий файл из списка аргументов либо *count*-й следующий файл при заданном *count*. Если приведено имя файла *filename*, то этот файл редактируется следующим. При наличии аргумента *+num* редактирование начинается на строке *num*. *num* также может быть шаблоном вида */pattern*. {Vim}

### wq

`wq[!]`

Сохранение файла и выход одним действием. Запись файла происходит всегда. Флаг ! говорит редактору произвести запись поверх текущего содержимого файла.

### wqall

`wqa[!]`

Записывает все несохраненные буферы и выходит из редактора. При указании ! запись происходит даже у буферов, помеченных как readonly. xall – другой псевдоним для этой команды. {Vim}

## write

[*address*] w[!] [[>>] *file*]

Записывает строки, указанные в *address* (либо все содержимое буфера, если *address* не указан), в файл *file*. При отсутствии *file* содержимое буфера записывается в текущий файл. Если используется >> *file*, к этому файлу дописываются строки. Чтобы редактор записал поверх текущего содержимого файла, поставьте !.

### Примеры

| | |
|---|---|
| :1,10w name_list | *Копировать первые 10 строк в файл* name_list |
| :50w >> name_list | *А теперь приписать в него строку 50* |

## write

[*address*] w !*command*

Передать строки, определенные в *address*, в команду *command*.

### Пример

| | |
|---|---|
| :1,66w !pr -h myfile \| lp | *Печатать первую страницу файла* |

## X

X

Запросить ключ шифрования. Это может оказаться предпочтительней, чем :set key, поскольку в консоли ввод ключа не отображается. Чтобы удалить ключ шифрования, просто установите опции key пустое значение. {Vim}

## xit

x

Сохранить файл, если он менялся со времени последнего сохранения, а затем выйти.

## yank

[*address*] y [*char*] [*count*]

Поместить строки, определенные в *address*, в именованный регистр *char*. Имена регистров – строчные буквы от a до z. При использовании прописных букв текст будет добавлен к соответствующему регистру.

Если *char* не задан, строки помещаются в общий регистр. *count* определяет количество копируемых строк, начиная с *address*.

### Пример

`:101,200 ya a` *Копировать строки 100–200 в регистр «а»*

---

### z

`[`*address*`] z [`*type*`] [`*count*`]`

Выводит текстовое окно, в котором заданная в *address* строка помещается на самый верх. *count* определяет количество выводимых строк.

### Тип

- `+` Помещает указанную строку на самый верх окна (по умолчанию).
- `-` Помещает указанную строку в низ окна.
- `.` Помещает указанную строку в центр окна.
- `^` Выводит предыдущий вид.
- `=` Помещает указанную строку в центр окна и делает ее текущей.

---

### &

`[`*address*`] & [`*options*`] [`*count*`]`

Повторяет предыдущую команду замены (`s`). *count* определяет число строк, в которых нужно произвести замену, начиная с *address*. *options* те же, что в команде замены.

### Примеры

`:s/Overdue/Paid/` *Однократная замена в текущей строке*
`:g/Status/&` *Повтор замены во всех строках с «Status»*

---

`[`*address*`] @ [`*char*`]`

Выполняет содержимое регистра, на который указывает *char*. Если задан *address*, то сначала курсор перемещается по этому адресу. Если в *char* указан символ `@`, то повторяется последняя команда `@`.

---

### =

`[`*address*`] =`

Выводит номер строки, которая задается в *address*. По умолчанию это номер последней строки.

### !

[*address*] !*command*

Выполняет в оболочке команду UNIX *command*. Если указан *address*, то в качестве стандартного ввода для команды *command* используются строки, содержащиеся по этому адресу, которые потом заменяются на вывод команды и ошибок. (Это называется фильтрацией текста через команду.)

#### Примеры

| | |
|---|---|
| `:!ls` | *Вывести список файлов в текущем каталоге* |
| `:11,20!sort -f` | *Отсортировать строки 11–20 в текущем файле* |

### < >

[*address*] < [*count*]

или

[*address*] > [*count*]

Смещает строки, определенные в *address*, налево (<) либо направо (>). При смещении добавляются/удаляются только первые пробелы и табуляции. *count* определяет количество смещаемых строк, начиная с *address*. Опция `shiftwidth` управляет числом столбцов, на которое идет смещение. Повторение < или > увеличивает величину сдвига. Например, `:>>>` сместит в три раза дальше, чем `:>`.

### ~

[*address*] ~ [*count*]

Заменяет последнее использовавшееся регулярное выражение (даже в тексте поиска, но не из команды `s`) на текст замены из последней команды `s`. Это описание довольно абстрактно; за подробностями обратитесь к главе 6.

### address

*address*

Выводит строки, указанные в *address*.

### ENTER

Выводит следующую строку файла. (Только в `ex`, в приглашении `:vi` не работает.)

# B

# Установка опций

В этом приложении описываются важные опции команды set для Solaris vi, nvi 1.79, elvis 2.2, Vim 7.1 и vile 9.6.

## Опции Solaris vi

Таблица B.1 содержит краткие описания важных опций команды set. В первом столбце перечислены опции в алфавитном порядке. Если опция допускает сокращение, то аббревиатура приводится в круглых скобках. Во втором столбце показано значение по умолчанию, используемое vi до вызова команды set (вручную или посредством файла .exrc). В последнем столбце описывается действие опции при ее включении.

*Таблица B.1. Опции set для Solaris vi*

| Опция | Значение | Описание |
|---|---|---|
| autoindent (ai) | noai | В режиме вставки каждая строка получает такой же отступ, как в строке выше или ниже. Используется с опцией shiftwidth. |
| autoprint (ap) | ap | Отображает изменения после каждой команды редактора. (При глобальной замене отображается последняя замена.) |
| autowrite (aw) | noaw | Автоматически записывает (сохраняет) измененный файл при открытии другого с помощью :n или при вызове команды UNIX :!. |
| beautify (bf) | nobf | Игнорировать все управляющие символы во время ввода (кроме tab, новой строки или перевода страницы). |
| directory (dir) | /tmp | Задает каталог, где ex/vi хранит файлы буферов. (Вы должны иметь права записи в этот каталог.) |

| Опция | Значение | Описание |
|---|---|---|
| edcompatible | noedcompatible | Запоминает флаги, использовавшиеся в последней команде замены (глобальная, подтверждение), и использует их в следующей команде замены. Несмотря на название, ни в одной из версий ed так не происходит. |
| errorbells (eb) | errorbells | Включает звуковой сигнал при ошибке. |
| exrc (ex) | noexrc | Разрешает выполнение файла .exrc, расположенного за пределами домашнего каталога пользователя. |
| flash (fp) | nofp | Включает мигание экрана вместо системного сигнала. |
| hardtabs (ht) | 8 | Определяет границы для аппаратных табуляций терминала. |
| ignorecase (ic) | noic | Не обращать внимание на регистр во время поиска. |
| lisp | nolisp | Вставляет отступы в соответствующем Lisp формате. ( ), { }, [[ и ]] меняются так, чтобы иметь смысл для Lisp. |
| list | nolist | Табуляции печатаются как ^I, а концы строк отмечаются с помощью $. (Чтобы указать, что завершающий символ является пробелом или табуляцией, используйте listp.) |
| magic | magic | Символы маски . (точка), * (звездочка) и [] (квадратные скобки) получают специальное значение в шаблонах. |
| mesg | mesg | При редактировании в vi разрешает отображение системных сообщений в терминале. |
| novice | nonovice | Требует использовать длинные имена команд ex, например copy или read. |
| number (nu) | nonu | Отображает номера строк в левой части экрана во время сеанса редактирования. |
| open | open | Разрешает вход в открытый или визуальный режим из ex. Хотя в Solaris vi этого нет, такая опция традиционно включается в vi и может присутствовать в версии vi на вашем UNIX. |
| optimize (opt) | noopt | Отменяет возврат каретки в конце строки при печати нескольких строк. Это ускоряет работу на медленных терминалах при печати строк с пробельными символами (пробелами или табуляциями), стоящими в начале. |
| paragraphs (para) | IPLPPPQP LIpplpipbp | Определяет разделители абзаца для перемещения по { или }. Пары символов, стоящие в значении этой опции, являются именами макросов troff, которые определяют начало абзаца. |

*Таблица B.1 (продолжение)*

| Опция | Значение | Описание |
|---|---|---|
| prompt | prompt | Отображает приглашение ex (:) при вводе команды vi Q. |
| readonly (ro) | noro | Все записи файла будут выдавать ошибку, если только не использовать ! после write (работает с w, ZZ или autowrite). |
| redraw (re) | | Перерисовывает экран при каждой правке (другими словами, режим вставки помещает символы к существующим, а удаленные строки сразу же исчезают). Установка по умолчанию зависит от скорости линии и типа терминала. noredraw полезна при малых скоростях и медленных терминалах: удаленные строки отображаются как @, а вставленные символы появляются поверх имеющихся, пока вы не нажмете ESC. |
| remap | remap | Разрешает вложенные последовательности отображений. |
| report | 5 | Отображает сообщение в строке состояния при каждой правке, затрагивающей как минимум указанное количество строк. Например, 6dd выдаст «6 lines deleted». |
| scroll | [½ *window*] | Количество строк, на которое прокручивается экран командами ^D и ^U. |
| sections (sect) | SHNHH HU | Определяет разделители раздела для перемещения по [[ или ]]. Пары символов, стоящие в значении этой опции, являются именами макросов troff, определяющими начало раздела. |
| shell (sh) | /bin/sh | Имя пути к оболочке, используемое для выхода в оболочку (:!) и в команде оболочки (:sh). Значение по умолчанию определяется из окружения оболочки. На разных системах это значение будет отличаться. |
| shiftwidth (sw) | 8 | Определяет количество пробелов в обратных (backward) (^D) табуляциях при использовании опции autoindent, а также в командах << и >>. |
| showmatch (sm) | nosm | В vi при вводе ) или } курсор «на секундочку» перемещается на соответствующую ( или {. (Если соответствующей скобки нет, выдается системный сигнал.) Опция очень полезна в программировании. |
| showmode | noshowmode | В режиме вставки в строке приглашения отображает сообщение, указывающее на тип вставки, например «OPEN MODE» или «APPEND MODE». |

| Опция | Значение | Описание |
|---|---|---|
| slowopen (slow) | | Задерживает отображение во время вставки. Значение по умолчанию зависит от скорости линии и типа терминала. |
| tabstop (ts) | 8 | Определяет количество пробелов, которые вставляются при нажатии табуляции во время редактирования. (Принтер все равно использует системную табуляцию, равную 8.) |
| taglength (tl) | 0 | Определяет количество символов, которые являются значимыми для тегов. Значение по умолчанию (ноль) означает, что значимы все символы. |
| tags | tags /usr/lib/tags | Определяет путь к файлу, содержащему теги (см. команду ctags для UNIX). По умолчанию vi ищет файл tags в текущем каталоге и в /usr/lib/tags. |
| tagstack | tagstack | Включает стек положений тегов. |
| term | | Устанавливает тип терминала. |
| terse | noterse | Выводит более короткие сообщения об ошибках. |
| timeout (to) | timeout | При использовании отображения клавиш прекратить ожидание следующего символа после 1-й секунды[a]. |
| ttytype | | Задает тип терминала. Эквивалентна term. |
| warn | warn | Отображает предупреждение «No write since last change». |
| window (w) | | Показывает определенное количество строк файла на экране. Значение по умолчанию зависит от скорости линии и типа терминала. |
| wrapmargin (wm) | 0 | Правая граница текста. Если значение больше нуля, по его достижении автоматически вставляется возврат каретки, чтобы разбить строку. |
| wrapscan (ws) | ws | При достижении конца файла поиск продолжается с его начала. |
| writeany (wa) | nowa | Разрешает запись в произвольный файл. |

[a] Если вы используете отображение из нескольких клавиш (например, :map zzz 3dw), то лучше включить notimeout, иначе придется набрать zzz в течение одной секунды. При наличии отображения для курсорной клавиши в режиме вставки (например, :map! ^[OB ^[ja) целесообразно установить timeout. В противном случае vi не будет реагировать на ESC, пока вы не нажмете еще какую-нибудь клавишу.

# Опции nvi 1.79

Всего в nvi 1.79 есть 78 опций, определяющих его поведение. В табл. В.2 приведены только самые важные из них. Большая часть опций из табл. В.1 здесь не повторяется.

*Таблица В.2. Опции set для nvi 1.79*

| Опция | Значение | Описание |
|---|---|---|
| backup | | Строка, описывающая используемый резервный файл. Текущее содержимое файла будет сохраняться в резервный файл перед записью новых данных. Первый символ N заставляет nvi включать номер версии в конец файла; номера версий всегда возрастают. Разумным примером является "N%.bak". |
| cdpath | Переменная окружения CDPATH или текущий каталог | Путь поиска для команды :cd. |
| cedit | | Когда первый символ этой строки вводится в командную строку с двоеточием, nvi открывает новое окно истории команд, которые можно редактировать. Нажатие ENTER на любой строке приводит к ее выполнению. Хорошим примером значения такой опции служит ESC (для его ввода нажмите ^V ^[). |
| comment | nocomment | Если первая непустая строка начинается с /*, // или #, nvi пропускает текст комментария перед отображением файла. Это избавляет от показа длинных скучных уведомлений. |
| directory (dir) | Переменная окружения TMPDIR или /tmp | Каталог, где nvi размещает свои временные файлы. |
| extended | noextended | При поиске используются расширенные регулярные выражения в стиле egrep. |
| filec | | Когда первый символ этой строки вводится в командную строку с двоеточием, nvi рассматривает отделенное пробелом слово перед курсором, как если бы к нему был приписан знак *, и выполняет подстановку в стиле оболочки. ESC также подходит для этой опции в качестве примера (чтобы ввести его, нажмите ^V ^[.) Если этот символ такой же, как у опции cedit, редактирование истории начинается, только если первый символ cedit одновременно является первым и в командной строке с двоеточием. |

| Опция | Значение | Описание |
|---|---|---|
| iclower | noiclower | Делает все поиски с регулярными выражениями нечувствительными к регистру, если шаблон поиска не содержит прописных букв. |
| leftright | noleftright | Длинные строки прокручиваются на экране слева направо, а не переносятся. |
| lock | lock | nvi пытается получить эксклюзивный доступ к файлу. Если это невозможно, для него открывается сеанс в режиме «только для чтения». |
| octal | nooctal | Неопознанные символы отображаются в восьмеричном, а не шестнадцатеричном виде. |
| path | | Список каталогов, разделенных запятыми, где nvi будет искать файл для редактирования. |
| recdir | /var/tmp/ vi.recover | Каталог, в котором хранятся файлы для восстановления. |
| ruler | noruler | Отображает строку и столбец курсора. |
| searchincr | nosearchincr | Выполняется инкрементный поиск. |
| secure | nosecure | Выключается доступ к внешним программам для фильтрации текста, отключаются команды ! и ^Z режима vi, а также команды !, shell, stop и suspend режима ex. После установки этой опции ее уже нельзя отключить. |
| shellmeta | ~{[*?$`'"\ | Если в аргументе, представляющем имя файла в команде ex, встречается один из этих символов, аргумент подставляется программой, заданной в опции shell. |
| showmode (smd) | noshowmode | В строке состояния отображается текущий режим. Если файл изменен, то показывается *. |
| sidescroll | 16 | Количество столбцов, на которые смещается экран влево или вправо при заданной опции leftright. |
| taglength (tl) | 0 | Определяет количество символов, которые являются значимыми для тегов. Значение по умолчанию (ноль) говорит, что значимы все символы. |
| tags (tag) | tags /var/db /libc.tags/sys /kern/tags | Список возможных файлов тегов. |
| tildeop | notildeop | Команда ~ «понимает» идущее следом за ней перемещение, а не только предшествующее количество повторений. |
| wraplen (wl) | 0 | Опция аналогична wrapmargin, за исключением того, что она указывает количество символов, считая от левого поля, по достижении которого строка будет разбита. Значение wrapmargin отменяет wraplen. |

# Опции elvis 2.2

В elvis 2.2 имеется 225 опций, определяющих поведение редактора. В табл. В.3 представлены только самые важные из них. Большая часть опций описана в табл. В.1 и здесь не повторяется.

*Таблица В.3. Опции set для elvis*

| Опция | Значение | Описание |
|---|---|---|
| autoiconify (aic) | noautoiconify | Старое окно сворачивается при разворачивании нового. Только в X11. |
| backup (bk) | nobackup | Создавать резервный файл (xxx.bak) перед записью нового файла на диск. |
| binary (bin) | | Данные буфера не являются текстовыми. Эта опция устанавливается автоматически. |
| boldfont (xfb) | | Имя полужирного шрифта. Только в X11. |
| bufdisplay (bd) | normal | Режим отображения по умолчанию для буфера (hex, html, man, normal, syntax или tex). |
| ccprg (cp) | cc ($1?$1:$2) | Команда оболочки для :cc. |
| directory (dir) | | Указывает место хранения временных файлов. Значение по умолчанию зависит от системы. |
| display (mode) | normal | Имя текущего режима отображения, задаваемое командой :display. |
| elvispath (epath) | | Список каталогов, в которых нужно искать конфигурационные файлы. Значение по умолчанию зависит от системы. |
| focusnew (fn) | focusnew | Фокус клавиатуры переносится на новое окно. Только в X11. |
| font (fnt) | | Название обычного шрифта (для интерфейсов Windows и X11). |
| gdefault (gd) | nogdefault | Команда замены меняет все вхождения. |
| home (home) | $HOME | Домашний каталог для подстановки знака ~ в именах файлов. |
| italicfont (xfi) | | Имя курсивного шрифта. Только в X11. |
| locked (lock) | nolocked | Буфер переводится в режим «только для чтения», после чего все меняющие его команды не смогут выполняться. Обычно устанавливается автоматически для HTML-файлов с правами «только для чтения». |
| lpcolor (lpcl) | nolpcl | Использовать цвет при печати; для :lpr. |
| lpcolumns (lpcols) | 80 | Ширина страницы принтера; для :lpr. |
| lpcrlf (lpc) | nolpcrlf | Принтер требует CR/LF для новой строки в файле; для :lpr. |

| Опция | Значение | Описание |
|---|---|---|
| lpformfeed (lpff) | nolpformfeed | После последней страницы посылается запрос на подачу страницы; для :lpr. |
| lpheader (lph) | nolph | Вверху страницы печатается заголовок; для :lpr. |
| lplines (lprows) | 60 | Длина страницы принтера; для :lpr. |
| lpout (lpo) | | Файл или фильтр для принтера, для :lpr. Обычно значение равно !lpr. Значение по умолчанию зависит от системы. |
| lptype (lpt) | dumb | Тип принтера; для :lpr. Принимает одно из следующих значений: ps, ps2, epson, pana, ibm, hp, cr, bs, dumb, html или ansi. |
| lpwrap (lpw) | lpwrap | Симуляция переноса строки; для :lpr. |
| makeprg (mp) | make $1 | Команда оболочки для :make. |
| prefersyntax (psyn) | never | Контролирует использование синтаксического режима. Полезно, когда в HTML и страницах man нужно показывать содержимое вместо отформатированного текста. Если значение равно never, синтаксический режим никогда не используется; если writable, то он устанавливается для записываемых файлов; если local — устанавливается для файлов текущего каталога; при значении always синтаксический режим используется всегда. |
| ruler (ru) | noruler | Отображает строку и столбец курсора. |
| security (sec) | normal | Одно из значений normal (стандартное поведение vi), safer (пытается предотвратить написание вредоносных скриптов) или restricted (пытается сделать elvis безопасным редактором с ограничениями). Вообще говоря, значение опции устанавливается командой :safely, так что не меняйте его вручную. |
| showmarkups (smu) | noshowmarkups | Для режимов man и html разметка отображается на позиции курсора и нигде больше. |
| sidescroll (ss) | 0 | Величина прокрутки по горизонтали. Нулевое значение – подражание vi (строки переносятся). |
| smartargs (sa) | nosmartargs | После ввода имени функции и символа function (обычно это открывающая круглая скобка) на экран выводятся аргументы функции, полученные из файла тегов. |
| spell (sp) | nospell | Неправильно написанные слова подсвечиваются. Это также работает с программами на основе данных из файла tags. |

*Таблица В.3 (продолжение)*

| Опция | Значение | Описание |
|---|---|---|
| taglength (tl) | 0 | Определяет количество символов, которые важны для тегов. Значение по умолчанию (ноль) указывает, что важны все символы. |
| tags (tagpath) | tags | Список возможных файлов tags. |
| tagstack (tsk) | tagstack | Запоминает исходные позиции, откуда происходил поиск тега в стеке. |
| undolevels (ul) | 0 | Определяет количество отменяемых команд. Нулевое значение воспроизведет поведение vi. Имеет смысл установить значение побольше. |
| warpback (wb) | nowarpback | При выходе перемещает указатель мыши обратно в xterm, откуда был запущен elvis. Только в X11. |
| warpto (wt) | don't | Задает способ перемещения указателя мыши при помощи ^W. don't указывает на запрет движения, scrollbar переносит указатель к полосе прокрутки, origin перемещает его в левый верхний угол, а corners — в самый дальний и самый ближний углы (по отношению к текущей позиции курсора). При этом дисплей X панорамируется, гарантируя, что все окно попадет на экран. |

# Опции Vim 7.1

В Vim 7.1 есть 295(!) опций, влияющих на его поведение. В табл. В.4 обобщены самые важные. Большая часть опций описана в табл. В.1 и здесь не повторяется.

Описание опций в этой таблице пришлось делать кратким. Более подробную информацию можно получить в онлайн-справке Vim.

*Таблица В.4. Опции set для Vim 7.1*

| Опция | Значение | Описание |
|---|---|---|
| autoread (ar) | noautoread | Определяет, был ли файл, открытый в Vim, изменен внешней программой, и автоматически обновляет буфер редактора новой версией файла. |
| background (bg) | dark или light | Программа пытается использовать наиболее подходящие для данного терминала цвета шрифта и фона. Значения по умолчанию зависят от текущего терминала и оконной системы. |

| Опция | Значение | Описание |
|---|---|---|
| backspace (bs) | 0 | Управляет возможностью пройти через новую строку или точку начала вставки, нажимая Backspace. Значения таковы: 0 – совместимость с vi, 1 – переход через символы новой строки и отступы, 2 – переход через начало вставки, символы новой строки и отступы. |
| backup (bk) | nobackup | Делает резервную копию перед перезаписью файла и оставляет ее, если файл был успешно сохранен. Чтобы иметь резервный файл только во время сохранения, используйте опцию writebackup. |
| backupdir (bdir) | ., ~/tmp/, ~/ | Список каталогов для резервных файлов, разделенных запятыми. По возможности резервный файл создается в первом каталоге этого списка. Если список пуст, вы не сможете создать резервный файл. Имя . (точка) дает команду использовать каталог с редактируемым файлом. |
| backupext (bex) | ~ | Строка, которая приписывается к имени файла для получения имени файла резервной копии. |
| binary (bin) | nobinary | Меняет несколько других опций, чтобы облегчить редактирование двоичных файлов. Прошлые значения этих опций запоминаются и восстанавливаются, когда bin вновь отключается. В каждом буфере хранится свой набор сохраненных значений опций. Эту опцию следует установить при редактировании двоичных файлов. Также можно использовать ключ -b командной строки. |
| cindent (cin) | nocindent | Включает автоматическую «умную» расстановку отступов для программы на C. |
| cinkeys (cink) | 0{,0},:,0#,!^F,o,O,e | Список клавиш, при нажатии которых в режиме вставки в текущей строке будет изменен отступ. Работает, если задана опция cindent. |
| cinoptions (cino) | | Управляет расстановкой отступов в программах C при помощи cindent. За подробностями обратитесь к онлайн-справке. |
| cinwords (cinw) | if, else, while, do, for, switch | Эти ключевые слова начинают новый отступ в следующей строке, когда включены smartindent или cindent. Для cindent это проделывается только в определенных местах (внутри {...}). |

*Таблица В.4 (продолжение)*

| Опция | Значение | Описание |
|---|---|---|
| comments (com) | | Разделенный запятыми список строк, начинающих комментарий. За подробностями обратитесь к онлайн-справке. |
| compatible (cp) | cp; nocp, если найден файл .vimrc | Vim начинает вести себя как vi во многих ситуациях, которые невозможно здесь описать. По умолчанию опция включена, чтобы не вызывать удивления. Наличие файла .vimrc отключает совместимость. Обычно это желаемый побочный эффект. |
| completeopt (cot) | menu,preview | Список опций для автозавершения в режиме вставки, разделенных запятыми. |
| cpoptions (cpo) | aABceFs | Список односимвольных флагов, каждый из которых задает конкретную область, в которой Vim будет или не будет вести себя как vi. Если список пуст, используются установки Vim по умолчанию. За подробностями обратитесь к онлайн-справке. |
| cursorcolumn (cuc) | nocursorcolumn | Подсвечивает столбец экрана, на котором стоит курсор, подсветкой CursorColumn. Опция полезна при упорядочивании текста по вертикали. Может замедлить отображение текста. |
| cursorline (cul) | nocursorline | Подсвечивает строку экрана, на которой стоит курсор, подсветкой CursorRow. Облегчает поиск текущей строки во время редактирования. Вместе с cursorcolumn дает эффект перекрестия. Может замедлить отображение текста. |
| define (def) | ^#\s*define | Шаблон поиска, описывающий определения макросов. Значение по умолчанию берется для программ на C. Для C++ используйте ^\(#\s*define\ \|[a-z]*\s*const\s*[a-z]*\). При использовании с командой :set обратные косые черты нужно удваивать. |
| directory (dir) | ., ~/tmp, /tmp | Список разделенных запятыми имен каталогов для файла подкачки. Он будет создан в первом каталоге, где это возможно. Если список пуст, файл не будет создан, что сделает восстановление невозможным! Значение . (точка) означает, что файл подкачки будет помещен в каталог с редактируемым файлом. Рекомендуется поставить точку (.) первой в списке, чтобы повторное редактирование файлов привело к предупреждающему сообщению. |
| equalprg (ep) | | Внешняя программа, используемая командой =. Если опция пуста, применяются внутренние функции форматирования. |

| Опция | Значение | Описание |
|---|---|---|
| errorfile (ef) | errors.err | Имя файла ошибок для режима quickfix. При использовании ключа командной строки -q значение errorfile устанавливается по его аргументу. |
| errorformat (efm) | **(Слишком длинное)** | Описание формата строк в формате scanf в файле ошибок. |
| expandtab (et) | noexpandtab | При вставке табуляции она заменяется подходящим числом пробелов. |
| fileformat (ff) | unix | Описывает соглашение, касающееся завершения строки в текущем буфере. Возможные значения: dos **(CR/LF)**, unix **(LF)** и mac **(CR)**. Обычно Vim устанавливает эту опцию автоматически. |
| fileformats (ffs) | dos,unix | Список соглашений по завершению строки, которые Vim пытается применить при чтении файла. Указание нескольких пунктов включает автоматическое распознавание конца строки при чтении файла. |
| formatoptions (fo) | Vim по умолчанию: tcq; vi по умолчанию: vt | Последовательность букв, определяющая, как проделывается автоматическое форматирование. За подробностями обратитесь к онлайн-справке. |
| gdefault (gd) | nogdefault | Команда замены меняет все вхождения. |
| guifont (gfn) | | Список шрифтов, разделенных запятыми, которые Vim пытается использовать при старте графической версии. |
| hidden (hid) | nohidden | При выгрузке буфера из окна он не удаляется, а скрывается. |
| history (hi) | Vim по умолчанию: 20; vi по умолчанию: 0 | Определяет, сколько команд, строк поиска и выражений хранится в истории команд. |
| hlsearch (hls) | nohlsearch | Подсвечивает все вхождения самого последнего шаблона поиска. |
| icon | noicon | Vim пытается сменить имя значка, связанного с окном, в котором он запущен. Действие этой опции отменяется опцией iconstring. |
| iconstring | | Строка, использующаяся в качестве имени значка для окна. |
| include (inc) | ^#\s*include | Определяет шаблон для поиска команд include. Значение по умолчанию подходит для программ на C. |
| incsearch (is) | noincsearch | Включает инкрементный поиск. |

*Таблица В.4 (продолжение)*

| Опция | Значение | Описание |
|---|---|---|
| isfname (isf) | @,48-57,/,.,-,_,+,,,$,:,~ | Список символов, которые можно включать в имя файла и путь к файлу. В не-UNIX-системах набор этих символов различается. Знак @ отвечает любому алфавитно-цифровому символу. Он также используется в других опциях isXXX, описанных ниже. |
| isident (isi) | @,48-57,_,192-255 | Список символов, которые можно использовать как идентификаторы. В не-UNIX-системах набор этих символов различается. |
| iskeyword (isk) | @,48-57,_,192-255 | Список символов, которые можно включать в ключевые слова. В не-UNIX-системах набор этих символов различается. Ключевые слова используются при поиске многими командами, такими как w и [i. |
| isprint (isp) | @,161-255 | Список символов, непосредственно отображаемых на экране. В не-UNIX-системах набор этих символов различается. |
| makeef (mef) | /tmp/vim##.err | Имя файла ошибок для команды :make. В не-UNIX-системах набор этих символов различается. Знаки ## заменяются на число, чтобы имя было уникальным. |
| makeprg (mp) | make | Программа, используемая командой :make. Символы % и # в значении параметра будут подставляться. |
| modifiable (ma) | modifiable | Если выключить эту опцию, то в буфере нельзя будет делать изменения. |
| mouse | | Включает мышь в неграфических версиях Vim. Работает в MS-DOS, Win32, QNX pterm и xterm. За подробностями обратитесь к онлайн-справке. |
| mousehide (mh) | nomousehide | Прячет указатель мыши при наборе текста и вновь показывает его при движении мыши. |
| paste | nopaste | Меняет большое количество опций так, чтобы вставка текста в окно Vim при помощи мыши не искажала вставляемый текст. Выключение восстанавливает прошлые значения этих опций. За подробностями обратитесь к онлайн-справке. |
| ruler (ru) | noruler | Показывает строку и столбец позиции курсора. |
| secure | nosecure | Выключает команды определенного типа в стартовом файле. Включается автоматически, если вы не являетесь владельцем файлов .vimrc и .exrc. |

| Опция | Значение | Описание |
|---|---|---|
| shellpipe (sp) | | Строка оболочки, используемая для захвата вывода из :make в файл. Значение по умолчанию зависит от оболочки. |
| shellredir (srr) | | Строка оболочки, используемая для захвата вывода из фильтра во временный файл. Значение по умолчанию зависит от оболочки. |
| showmode (smd) | Vim по умолчанию: smd; vi по умолчанию: nosmd | Помещает сообщение в строку состояния для режимов вставки и замещения, а также визуального режима. |
| sidescroll (ss) | 0 | Определяет, сколько столбцов прокручивается в горизонтальном направлении. Нулевое значение помещает курсор в середину экрана. |
| smartcase (scs) | nosmartcase | Переопределяет действие опции ignorecase, если строка поиска содержит прописные буквы. |
| spell | nospell | Включает проверку орфографии. |
| suffixes | *.bak,~,.o,.h,.info,.swp | Если при завершении имени файла шаблону удовлетворяют несколько файлов, значение этой переменной устанавливает для них приоритет, чтобы выбрать файл, который будет использоваться Vim. |
| taglength (tl) | 0 | Определяет количество символов, значимых для тегов. Значение по умолчанию (ноль) указывает, что значимы все символы. |
| tagrelative (tr) | Vim по умолчанию: tr; vi по умолчанию: notr | Имена файлов в файле tags рассматриваются относительно каталога с этим файлом. |
| tags (tag) | ./tags,tags | Имена файлов для команды :tag, разделенные пробелами или запятыми. Символы ./, стоящие в начале, заменяются на полный путь текущего файла. |
| tildeop (top) | notildeop | Команда ~ работает как оператор. |
| undolevels (ul) | 1000 | Максимальное количество изменений, которые можно отменить. Значение 0 устанавливает совместимость с vi, при котором u отменяет саму себя на первом уровне отмены. Значения по умолчанию в не-UNIX-системах могут быть разными. |

*Таблица В.4 (продолжение)*

| Опция | Значение | Описание |
|---|---|---|
| viminfo (vi) | | При старте считывает файл viminfo, а при выходе записывает в него. Опция имеет сложное значение: она контролирует информацию разного рода, которую Vim хранит в этом файле. За подробностями обратитесь к онлайн-справке. |
| writebackup (wb) | writebackup | Создавать резервную копию перед перезаписью файла. Копия удаляется после успешного сохранения файла, если только не включена опция backup. |

# Опции vile 9.6

vile 9.6 содержит 167 опций (в этом редакторе они называются «режимами»), которые подразделяются на универсальные (universal), буферные (buffer) и оконные (window) в зависимости от их использования. Также есть 101 переменная окружения, которые более пригодны для применения в скриптах, нежели для непосредственного изменения пользователем[1]. Не все опции доступны на всех платформах – некоторые применяются только в X11 или Win32.

В табл. В.5 показаны самые важные опции vile, а также значения по умолчанию, задаваемые на этапе компиляции. Скрипты инициализации, такие как vileinit.rc, переопределяют некоторые из этих значений. Большая часть опций описана в табл. В.1 и здесь не повторяется.

*Таблица В.5. Опции set для vile 9.6*

| Опция | Значение | Описание |
|---|---|---|
| alt-tabpos (atp) | noatp | Определяет расположение курсора на пробельном символе, представляющем табуляцию (на правом или левом его конце). |
| animated | animated | Автоматически обновляет содержимое рабочих буферов при их изменении. |
| autobuffer (ab) | autobuffer | Использует буферы «по времени обращения». Буферы отсортированы в порядке их использования либо редактирования. |

[1] Сюда входят переменные, которые устанавливаются или используются в качестве побочного эффекта от действия других команд. Поскольку они ориентированы на скрипты, их описание очень громоздкое и тоже не подходит для этой таблицы. За подробностями обратитесь к онлайн-справке.

| Опция | Значение | Описание |
|---|---|---|
| autocolor (ac) | 0 | Автоматическая подсветка синтаксиса. Если стоит значение ноль, она отключена. Иначе этой опции нужно присвоить небольшое положительное число, определяющее количество миллисекунд ожидания в «интервале тишины» перед запуском autocolor-hook. |
| autosave (as) | noautosave | Автоматическое сохранение файла. Запись производится после ввода очередного количества символов текста, указанного в autosavecnt. |
| autosavecnt (ascnt) | 256 | Устанавливает, сколько символов нужно ввести для выполнения автоматического сохранения. |
| backspacelimit (bl) | backspacelimit | Если опция отключена, вы можете пройти через точку начала ввода, нажимая Backspace. |
| backup-style | off | Определяет, каким образом создаются резервные копии при сохранении файла. Возможные значения: off, .bak для резервных копий в стиле DOS и tilde для резервных копий в UNIX в стиле Emacs hello.c~. |
| bcolor | default | Устанавливает цвет фона в тех системах, где это поддерживается. |
| byteorder-mark (bom) | auto | Управляет проверкой префикса, используемого для распознавания различных типов кодировки UTF. Значение по умолчанию auto предписывает vile проверить файл; конкретное значение предписывает программе использовать это значение. |
| check-modtime | nocheck-modtime | Выдает предупреждение «file newer than buffer», если файл был изменен с момента последнего чтения или записи, и просит ввести подтверждение. |
| cindent | nocindent | Включает отступы в стиле C, что поможет при вводе расставлять отступы автоматически, как с autoindent. |
| cindent-chars | :#{}( )[] | Список символов, интерпретируемых в режиме cindent. Сюда входят # для отступа до столбца 1 и : для более далекого отступа, как после метки. Пара символов, расположенных в fence-pairs, приводит к тому, что заключенный в эту пару текст получает дополнительный отступ. |

*Таблица В.5 (продолжение)*

<table>
<tr><th>Опция</th><th>Значение</th><th>Описание</th></tr>
<tr><td>cmode</td><td>off</td><td>Встроенный основной режим для кода на С.</td></tr>
<tr><td>color-scheme (cs)</td><td>default</td><td>Задает именованный набор значений fcolor, bcolor, video-attrs и $palette, определенный командой define-color-scheme.</td></tr>
<tr><td>comment-prefix</td><td>^\s*\(\(\s*[#*>]\)\|\(///*\)\)\+</td><td>Определяет начальные символы строк, которые останутся на месте при переформатировании комментариев. Значения по умолчанию хорошо подходят для комментариев в файлах Makefile, оболочке и в программах на С, а также в файлах электронной почты.</td></tr>
<tr><td>comments</td><td>^\s*/\?\(\s*[#*>/]\)\+/\?\s*$</td><td>Регулярное выражение, определяющее разграничители закомментированных абзацев. Целью является сохранение абзацев внутри комментариев при переформатировании.</td></tr>
<tr><td>cursor-tokens</td><td>regex</td><td>Задает способ выполнения редактором vile анализа символов экрана для различных команд: с помощью регулярных выражений или по классам символов. Использует перечисление: both, cclass и regex.</td></tr>
<tr><td>dirc</td><td>nodirc</td><td>При сканировании каталогов для завершения имени файла vile проверяет каждое встреченное имя. Эта опция позволяет различать в приглашении имена файлов и каталогов.</td></tr>
<tr><td>dos</td><td>nodos</td><td>При считывании файлов удаляет CR из пары CR/LF, а при записи помещает их обратно. Новые буферы для несуществующих пока файлов наследуют тип конца строки из операционной системы независимо от значения dos.</td></tr>
<tr><td>fcolor</td><td>default</td><td>Устанавливает цвет шрифта в системах, которые его поддерживают.</td></tr>
<tr><td>fence-begin<br>fence-end</td><td>/*<br>*/</td><td>Регулярные выражения для начала и конца простых невложенных заграждающих меток (fences), например комментариев в С.</td></tr>
<tr><td>fence-if<br>fence-elif<br>fence-else<br>fence-fi</td><td>^\s*#\s*if<br>^\s*#\s*elif\><br>^\s*#\s*else\><br>^\s*#\s*endif\></td><td>Регулярные выражения, отмечающие начало, «else if», «else» и конец строчных вложенных заграждающих меток (fences), таких как команды препроцессора С.</td></tr>
<tr><td>fence-pairs</td><td>{}( )[]</td><td>Каждая пара символов обозначает набор заграждающих меток, соответствующих друг другу при вводе %.</td></tr>
<tr><td>file-encoding</td><td>auto</td><td>Определяет кодировку символов буфера и принимает одно из следующих значений: 8bit, ascii, auto, utf-8, utf-16 или utf-32.</td></tr>
</table>

| Опция | Значение | Описание |
|---|---|---|
| filtername (fn) | | Задает фильтр для подсветки синтаксиса для данного основного режима. |
| for-buffers (fb) | mixed | Определяет, что именно используется в командах for-buffers и kill-buffer для выбора имени буфера: маски (globbing) или регулярные выражения. |
| glob | !echo %s | Задает, как обращаться с символами маски (например, * и ?) в приглашении имени файла. Значение off выключает подстановку, а on включает внутренний механизм, поддерживающий обычные маски оболочки и ~. Значение по умолчанию в UNIX гарантирует совместимость с вашей оболочкой. |
| highlight (hl) | highlight | Включает или выключает подсветку синтаксиса в соответствующих буферах. |
| history (hi) | history | Сохраняет команды, введенные в командной строке с двоеточием (мини-буфере) в буфере [History]. |
| horizscroll (hs) | horizscroll | Перемещает к концу длинной строки и смещает весь экран в сторону. Если опция не установлена, смещается только текущая строка. |
| ignoresuffix (is) | \(\.orig\|~\)$ | Удаляет данный шаблон из имени файла перед поиском соответствия для суффиксов основного режима. |
| linewrap (lw) | nolinewrap | Переносит длинные строки на несколько экранных строк. |
| maplonger | nomaplonger | Функция отображения выбирает самую длинную отображенную последовательность, а не самую короткую. |
| meta-insert-bindings (mib) | mib | Управляет поведением 8-разрядных символов во время вставки. Как правило, привязки клавиш работают только в командном режиме. В режиме вставки все символы вставляют сами себя. Если этот режим включен и вводится метасимвол (то есть символ с установленным восьмым битом), привязанный к функции, то предпочтение будет отдано этой функции, и она будет вызвана из режима вставки. Все несвязанные метасимволы будут по-прежнему вставлять сами себя. |
| mini-hilite (mh) | reverse | Определяет атрибут подсветки, который используется при переключении пользователем режима редактирования в мини-буфере. |

*Таблица В.5 (продолжение)*

| Опция | Значение | Описание |
|---|---|---|
| modeline | nomodeline | Управляет поддержкой строки с информацией о режиме (modeline) в стиле vi. |
| modelines | 5 | Задает количество строк, которые будут сканироваться в начале и конце буфера в поисках строки modeline в стиле vi. |
| overlap-matches | overlap-matches | Меняет подсветку, используемую visual-matches для управления показом перекрывающихся совпадений. |
| percent-crlf | 50 | Процентное соотношение для строк типа CR/LF, чтобы vile автоматически преобразовал буфер recordseparator в crlf. |
| percent-utf8 | 90 | Процентное соотношение для символов, содержащих встроенные нули, что делает их схожими с символами в кодировке UTF-16 или UTF-32. Если опция file-encoding установлена в auto, а соотношение больше указанного, vile загрузит данные буфера как UTF-8. |
| popup-choices (pc) | delayed | Управляет использованием всплывающего окна, помогающего при автозавершении. Значением может быть off, чтобы окно не появлялось, immediate для немедленного появления либо delayed, чтобы ожидалось повторное нажатие на Tab. |
| popup-msgs (pm) | nopopup-msgs | Если эта опция включена, vile выводит буфер [Messages], где показан текст, записанный в строке сообщений. |
| recordseparator (rs) | lf[a] | Определяет формат файлов, которые vile считывает и записывает. Возможные форматы: lf (для UNIX), crlf (для DOS), cr (для Macintosh) и default (lf или crlf, в зависимости от платформы). |
| resolve-links | noresolve-links | Если опция установлена, то vile раскрывает имена файлов, компоненты путей которых являются символьными ссылками. Это помогает избежать многократных правок одного и того же файла из-за разных путей к нему. |
| ruler | noruler | Отображает номер текущей строки и столбца в строке состояния, а также определяет, какое процентное соотношение строк буфера находится выше курсора. |

[a] Зависит от платформы, на которой был скомпилирован vile.

| Опция | Значение | Описание |
|---|---|---|
| showchar (sc) | noshowchar | Показывает значение текущего символа в строке состояния. |
| showformat (sf) | foreign | Задает способ и время отображения информации о recordseparator в строке состояния. Значения опции: always, differs (показывать, когда локальный режим отличается от глобального), local (показывать всегда, когда задан локальный режим), foreign (показывать, когда recordseparator отличается от родного) и never. |
| showmode (smd) | showmode | Отображает в строке состояния индикатор, обозначающий режимы вставки и замены. |
| sideways | 0 | Определяет, на сколько символов перемещается экран налево или направо. При значении 0 экран будет перемещаться на одну треть. |
| tabinsert (ti) | tabinsert | Разрешает физическую вставку табуляции в буфер. Если опцию отключить (notabinsert), vile никогда не будет это делать; вместо табуляции будет вставляться подходящее число пробелов. |
| tagignorecase (tc) | notagignorecase | Делает поиск по тегам нечувствительным к регистру. |
| taglength (tl) | 0 | Определяет количество символов, значимых для тегов. Установка по умолчанию, равная нулю, показывает, что все символы являются значимыми. Это не распространяется на теги, стоящие под курсором. Для них всегда выполняется точное соответствие. (Такое поведение отличается от других редакторов.) |
| tagrelative (tr) | notagrelative | При использовании файла tags из другого каталога имена файлов из этого файла считаются относительно каталога, где располагается файл tags. |
| tags | tags | Список файлов, разделенных запятыми, в которых нужно искать теги. |
| tagword (tw) | notagword | Использовать все слова под курсором в качестве поиска тега, а не только ту его часть, которая начинается с позиции курсора. |

*Таблица В.5 (продолжение)*

| Опция | Значение | Описание |
|---|---|---|
| undolimit (ul) | 10 | Ограничивает количество возможных отмен. Значение ноль показывает отсутствие предела. |
| unicode-as-hex (uh) | nounicode-as-hex | При отображении буфера, кодировка файла которого является одной из разновидностей Unicode (например, utf-8, utf-16 или utf-32), не-ASCII символы показываются в формате \uXXXX, даже если дисплей может показать их как обычные символы. |
| unprintable-asoctal (uo) | nounprintableas-octal | Отображает непечатаемые символы с установленным восьмым битом в восьмеричном виде. Если опция не установлена, они выводятся в шестнадцатеричном формате. Непечатаемые символы с неустановленным восьмым битом показываются в нотации управляющих символов. |
| visual-matches | none | Управляет подсветкой всех соответствий шаблона поиска. Возможные значения: при none подсветки нет, а underline, bold и reverse используются для соответствующих типов подсветки (подчеркивание, полужирный и инверсия цвета). Также можно использовать цвета, если система это поддерживает. |
| xterm-fkeys | noxterm-fkeys | Поддержка модифицированных функциональных клавиш xterm путем генерации системных привязок к Shift-, Ctrl- и Alt- для каждой функциональной клавиши, представленной в описании терминала. |
| xterm-mouse | noxterm-mouse | Разрешает использование мыши из xterm. За подробностями обратитесь к онлайн-справке. |
| xterm-title | noxterm-title | Разрешает обновление заголовка, если вы работаете из xterm. Всякий раз при переключении в другой буфер vile может обновлять заголовок. Использует проверку переменной TERM, эквивалентную режиму xtermmouse. |

# C

# Возможные проблемы

В этом приложении собраны возможные проблемы, описывавшиеся в части I. Здесь же они сведены в одном месте в виде справочника.

## Проблемы при открытии файлов

- *При запуске* `vi` *появляется сообщение* `[open mode]`

  Возможно, неправильно распознается тип вашего терминала. Выйдите из сеанса редактирования с помощью команды `:q` и проверьте переменную окружения `TERM`. Ей нужно задать имя вашего терминала. Также вы можете спросить правильное значение типа терминала у своего системного администратора.

- *Появляется одно из следующих сообщений:*

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

  Либо тип вашего терминала не опознан, либо что-то не так с `terminfo` или `termcap`. Введите `:q` для выхода и проверьте переменную окружения `$TERM` либо попросите системного администратора выбрать тип терминала для вашего окружения.

- *Появляется сообщение* `[new file]`*, когда вы уверены, что файл существует.*

  Проверьте правильность использования регистра символов в имени файла (имена файлов в UNIX чувствительны к регистру). Если все правильно, то, возможно, вы находитесь в другом каталоге. Введите

:q для выхода, после чего проверьте, находитесь ли вы в одном каталоге с файлом (наберите pwd в командной строке UNIX). Если да, то выведите список содержащихся в нем файлов (с помощью ls) и проверьте, может файл существует под немного отличающимся именем.

- *Вы запустили vi, но попали в приглашение с двоеточием (это говорит о том, что вы находитесь режиме строкового редактора ex).*

  Возможно, было введено прерывание перед тем, как vi успел нарисовать экран. Войдите в vi, введя vi в приглашении ex (:).

- *Возникает одно из следующих сообщений:*

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

  «Read only» означает, что разрешено только просматривать файл, а сделанные изменения не сохраняются. Возможно, вы запустили vi в режиме просмотра (либо через view, либо как vi -R) или же у вас отсутствуют права на запись этого файла. Обратитесь к разделу «Проблемы при сохранении файлов» ниже.

- *Возникает одно из следующих сообщений:*

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

  Файл, который вы хотите отредактировать, не является текстовым. Введите :q! для выхода и проверьте его, например, командой file.

- *При вводе :q по одной из вышеназванных причин появляется сообщение:*

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

  Вы ненароком внесли изменение в файл. Для выхода из vi введите :q!. В этом случае изменения, сделанные во время сеанса, не будут сохранены.

## Проблемы при сохранении файлов

- *Вы пытаетесь записать файл, но получаете одно из следующих сообщений:*

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

Введите `:w! file`, чтобы перезаписать существующий файл, или `:w newfile` для сохранения текущей редакции в новом файле.

- *Вы хотите записать файл, но не имеете права на запись для него. Выдается сообщение «Permission denied».*

  Используйте `:w newfile` для записи буфера в новый файл. Если у вас есть права на запись для каталога, то можно заменить первоначальную версию новым файлом с помощью `mv`. Если же их нет, введите `:w pathname/file`, чтобы сохранить буфер в том каталоге, где у вас есть разрешение на запись (например, в домашнем или `/tmp`).

- *Вы пытаетесь записать файл, но получаете сообщение, в котором говорится, что файловая система переполнена.*

  Введите `:!rm junkfile` для удаления (большого) ненужного файла, тем самым освободив место (если команду `ex` начать с восклицательного знака, то вы получите доступ в UNIX).

  Можно ввести `:!df` и посмотреть, есть ли свободное место в другой файловой системе. Если есть, выберите каталог в ней и запишите свой файл туда, воспользовавшись `:w pathname`. (`df` – это команда UNIX, проверяющая свободное место на дисках.)

- *Система переводит вас в open mode и сообщает, что файловая система переполнена.*

  Диск переполнен временными файлами `vi`. Наберите `:!ls /tmp`, чтобы посмотреть наличие файлов для удаления, дабы получить немного места на диске[1]. Если таковые имеются, создайте временную оболочку UNIX, из которой можно удалить эти файлы или воспользоваться другими командами UNIX. Оболочку можно создать командой `:sh`. Для выхода из нее и возврата в `vi` нажмите CTRL-D или введите команду `exit`. (В современных системах UNIX при использовании оболочки с управлением заданиями можно просто нажать CTRL-Z для приостановки `vi` и возврата в командную строку UNIX. Для возврата в `vi` введите `fg`.) После высвобождения места на диске сохраните файл командой `:w!`.

- *Вы пытаетесь записать файл, но получаете сообщение о том, что достигнуты дисковые квоты.*

  Попробуйте заставить систему записать ваш буфер с помощью команды `:pre` (сокращение от `:preserve`). Если это не сработало, поищите файлы для удаления. Воспользуйтесь `:sh` (или CTRL-Z, если система поддерживает управление заданиями) для выхода из `vi` и удаления файлов. Для возврата в `vi` нажмите CTRL-D (или `fg`), затем запишите файл командой `:w!`.

---

[1] `vi` может хранить временные файлы в `/usr/tmp`, `/var/tmp` или вашем домашнем каталоге. Возможно, вам придется немного покопаться, чтобы узнать, что именно занимает столько места.

## Проблемы с попаданием в визуальный режим

- *Во время редактирования в* `vi` *вы неожиданно попали в редактор* `ex`*:*

  `Q` в командном режиме `vi` вызывает `ex`. Попав в `ex`, можно всегда вернуться в `vi` с помощью команды `vi`.

## Проблемы с командами vi

- *При вводе команды текст скачет по экрану, и ничего не работает нужным образом.*

  Убедитесь, что вы вводите именно `j`, а не `J`.

  Возможно, вы случайно нажали клавишу CAPS LOCK. `vi` чувствителен к регистру, то есть прописные команды (`I`, `A`, `J`, и т. д.) отличаются от строчных (`i`, `a`, `j`). Если CAPS LOCK нажата, все ваши команды рассматриваются не как строчные, а как прописные. Нажмите эту клавишу еще раз для возврата в строчный режим, а затем ESC, чтобы попасть в командный. После этого введите либо `U` для восстановления последней измененной строки, либо `u` для отмены последней команды. Возможно, вам придется сделать дополнительные правки, чтобы восстановить искаженную часть файла.

## Проблемы при удалении

- *Вы удалили не тот текст и хотите вернуть его обратно.*

  Есть несколько способов восстановить удаленный текст. Если вы удалили что-нибудь и сразу заметили ошибку, просто нажмите `u` для отмены действия последней команды (например, `dd`). Это сработает, если вы не вводили никаких других команд, поскольку `u` отменяет только последнюю команду. В противовес ей, `U` восстановит строку в первоначальном состоянии – до внесения в нее каких-либо изменений.

  Однако у вас есть возможность восстановить недавнее удаление и с помощью команды `p`, так как `vi` сохраняет последние девять удалений в девяти пронумерованных буферах удаления. Например, если вы знаете, что для восстановления нужно отменить третье удаление, введите:

  ```
  "3p
  ```

  чтобы «вставить» содержимое буфера номер 3 в строку под курсором. Это сработает, только если удалялись строки. Слова и фрагменты строк в буфере не сохраняются. Если необходимо восстановить удаленное слово или часть строки, а `u` не работает, используйте команду `p`. Это восстановит все, что вы удаляли в последний раз.

# vi и Интернет

*Разумеется, vi дружественен пользователю.*
*Только он очень разборчив в выборе друзей.*

Став с 1980 года «стандартным» экранным редактором UNIX, `vi` прочно закрепился а UNIX-культуре.

`vi` помог построить UNIX, который, в свою очередь, выстроил основу современного Интернета. Таким образом, очевидно, что должен существовать хотя бы один сайт, посвященный этому редактору. В данном приложении описаны некоторые из ресурсов `vi`, доступные для знатоков программы.

## Откуда начать

Ничто не устаревает так быстро, как публикация списка сайтов Всемирной сети в печатном издании. Мы попытались дать URL, которые, надеемся, проживут долго.

Кроме того, в разделе «Tips» в документации `elvis` перечислены интересные веб-сайты, связанные с `vi` (откуда мы и начали). Кроме того, стоит взглянуть в новостную группу Usenet `comp.editors`.

## Веб-сайты vi

Существуют два основных веб-сайта, связанных с `vi`: *vi Lover's Home Page* Томера М. Джила (Thomer M. Gil) и *Vi Pages* от Свена Гукеса (Sven Guckes). Каждый из них содержит множество ссылок на другие ресурсы, связанные с `vi`.

## vi Lover's Home Page

Страница *vi Lover's Home Page* расположена по адресу *http://www.thomer.com/vi/vi.html*. Сайт содержит следующие разделы:

- Таблицу известных модификаций `vi` со ссылками на исходный код и дистрибутивы в двоичном формате.
- Ссылки на другие сайты по `vi`, включая *Vi Pages* Свена Гукеса (Sven Guckes).
- Огромное число ссылок на документацию, руководства, справку и учебники для самых разных уровней пользователей.
- Макросы `vi` для написания HTML-документов и решения задачи «Ханойские башни», FTP-сайты и другие наборы макросов.
- Разнообразные ссылки по теме `vi`: стихи, рассказ об «истинной истории» программы, ее сравнение с Emacs и кофейные кружки `vi` (см. раздел «vi для любителей Java» на стр. 485).

Там есть много чего еще. Это хорошая отправная точка.

## Vi Pages

Сайт *Vi Pages* расположен по адресу *http://www.vi-editor.org*. На нем есть следующие разделы:

- Подробное сравнение опций и функций у разных модификаций `vi`.
- Снимки экрана различных версий редактора.
- Таблица, в которой перечислены многие модификации `vi` и контакты их создателей (имя, адрес, URL).
- Несколько указателей на файлы FAQ.
- Некоторые интересные цитаты о `vi`. Например, одна из них вынесена в эпиграф этой главы.
- Другие ссылки, в том числе ссылка на кофейные кружки `vi`.

Сайт *vi Lover's Home Page* ссылается на этот ресурс как на «единственный сайт про `vi` на этой планете, который лучше, чем тот, что вы читаете». Его тоже стоит посетить.

## vi Powered!

Мы нашли логотип *vi Powered* (рис. D.1). Это небольшой файл GIF, который можно добавлять на свою веб-страницу, показывая, что при ее создании вы использовали `vi`.

***Рис. D.1.*** *vi Powered!*

Первоначальной домашней страницей логотипа vi Powered, созданного Антонио Валле (Antonio Valle), была *http://www.abast.es/~avelle/vi.html*. Она написана на испанском языке и сейчас не доступна. Английская находится на *http://www.darryl.com/vi.shtml*, а инструкции по размещению логотипа можно прочитать на *http://www.darryl.com/addlogo.html*. Чтобы поместить логотип на свой сайт, необходимо проделать несколько простых шагов:

1. Скачайте его. Для этого зайдите на *http://www.darryl.com/vipower.gif* и сохраните картинку. Можно использовать утилиту скачивания из командной строки, такую как wget.
2. Добавьте следующие строки в код вашей веб-страницы в подходящем месте:

```
<A HREF="http://www.darryl.com/vi.html">
<IMG SRC="vipower.gif">
</A>
```

   После этого на странице появится логотип с гипертекстовой ссылкой на домашнюю страницу *vi Powered!*. Можно добавить к тегу <IMG> атрибут ALT="This Web Page is vi Powered" для пользователей неграфических браузеров.
3. В раздел <HEAD> веб-страницы добавьте следующий код:

```
<META name="editor" content="/usr/bin/vi">
```

Как Настоящие Программисты избегают текстовых процессоров WYSIWYG в пользу troff, так и Настоящие Веб-мастера отказываются от различных новомодных сред для верстки HTML в пользу vi. Можете с гордостью использовать логотип *vi Powered!* для демонстрации этого факта. ☺

Логотип Vim с незначительными изменениями можно найти на *http://www.vim.org/logos.php*. Также несколько логотипов *Vim Powered!* для веб-сайтов доступны на *http://www.vim.org/buttons.php*.

## vi для любителей Java

Несмотря на название, этот подраздел касается Java, который вы пьете, а не на котором программируете[1].

Теоретически «Настоящий Программист» помимо использования vi для программирования на C++, составления документации на troff и написания своей веб-страницы, несомненно, захочет чашечку кофе. Его можно пить из кружки с напечатанным на ней справочником команд vi!

Когда мы в первый раз нашли ссылку на кружки vi, они были доступны в наборе из четырех штук на специальном веб-сайте. Этого сайта боль-

---

[1] Тем не менее интересно совпадение: язык Java выпущен компанией Sun Microsystems, основателем и бывшим вице-президентом которой является Билл Джой (Bill Joy) – автор первоначального vi.

ше нет, но кружки, футболки, свитеры и коврики для мыши с символикой vi теперь есть на *http://www.cafepress.com/geekcheat/366808*.

## Online-учебник по vi

На двух упомянутых нами веб-страницах имеется огромное число ссылок на документацию vi. Пожалуй, стоит отметить онлайн-учебник из девяти частей из журнала *Unix World* за авторством Вальтера Зинтца (Walter Zintz). Стартовая страница учебника находится по адресу *http://www.networkcomputing.com/unixworld/tutorial/009/009.html* (страница перемещалась. Возможно, ссылка с домашних страниц vi не будет работать, но этот URL был действительным, когда мы проверяли его в начале 2011 года). В учебнике освещаются следующие темы:

- Основы редактирования
- Адреса в строковом режиме
- Команда g (global)
- Команда подстановки
- Среда редактора (команды set, tags и EXINIT и .exrc)
- Адреса и столбцы
- Команды замены r и R
- Автоматические отступы
- Макросы

Также в этом учебнике есть онлайн-тест, который можно пройти, чтобы узнать, насколько хорошо вы усвоили материал учебника. Можете сразу приступить к тесту, чтобы понять, насколько хорошо вы усвоили нашу книгу!

# Другой клон vi

На рис. с D.2 по D.9 рассказана история vigor, *другой* модификации vi.

Исходный код vigor доступен на *http://vigor.sourceforge.net*.

# Развлеките друзей!

Возможно, в долгосрочной перспективе самая ценная информация о vi будет в FTP-архивах на *alf.uib.no*. Первоначально они располагались на *ftp://afl.uib.no/pub/vi*. Этот сайт исчез, но зеркало архивов можно найти по адресу *ftp://ftp.uu.net/pub/text-processing/vi*[1]. Файл INDEX в каталоге описывает содержимое архивов и перечисляет дополнительные зеркала, которые могут быть ближе к вам географически.

[1] Может быть, вам повезет больше, если вы запросите доступ к сайту из клиента FTP командной строки, а не из веб-браузера.

К сожалению, последний раз архивы обновлялись в мае 1995 года. Тем не менее основная функциональность vi с той поры не поменялась, информация и макросы оттуда все еще полезны. В архиве содержатся четыре подкаталога:

docs

Документация vi и несколько постов в comp.editors.

macros

Макросы vi.

comp.editors

Различные материалы с comp.editors.

programs

Исходный код модификаций vi для различных платформ (и другие программы). Пользуйтесь ими с осторожностью, так как многие программы уже устарели.

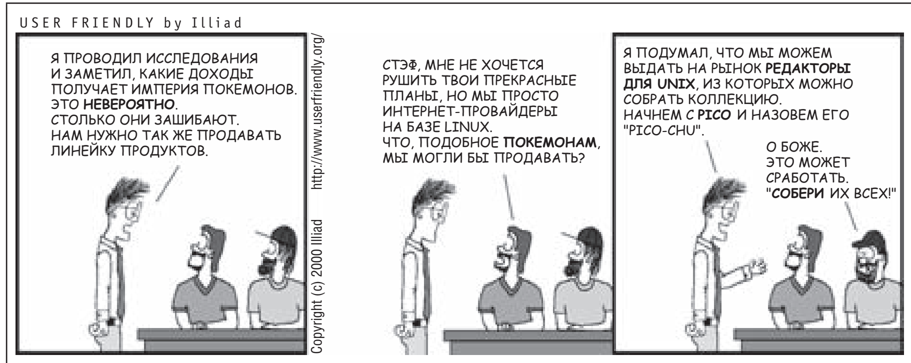



***Рис. D.2.*** *История vigor – часть I*

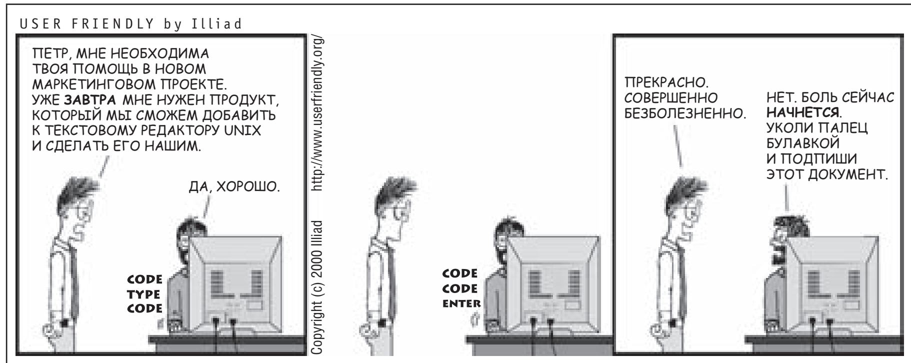



***Рис. D.3.*** *История vigor – часть II*

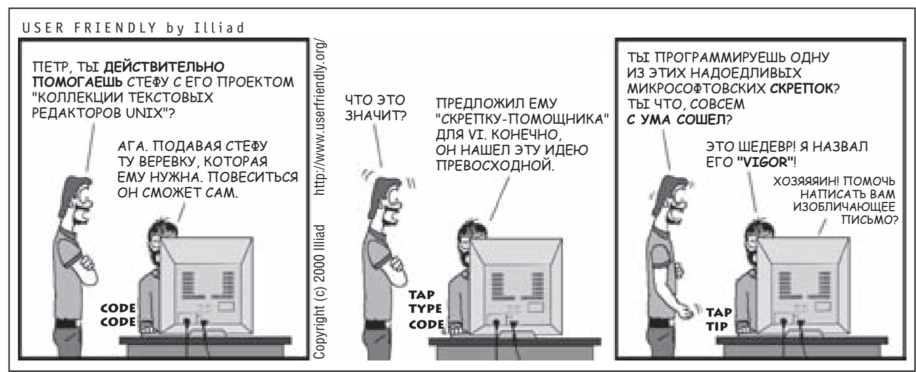



***Рис. D.4.*** *История vigor – часть III*

***Рис. D.5.*** *История vigor – часть IV*

Самыми интересными являются `docs` и `macros`. Каталог `docs` содержит огромное количество статей и материалов, включая руководства для новичков, объяснения ошибок и быстрые ссылки, а также множество коротких заметок по типу «how to» (например, как сделать заглавной первую букву в предложении). Там даже есть песня про `vi`!

Каталог `macros` содержит более 50 файлов, производящих самые разные действия. Здесь мы упомянем только три из них. (Файлы, чье имя заканчивается на `.Z`, сжаты программой UNIX `compress`. Для разархивирования можно пользоваться как `uncompress`, так и `gunzip`.)

`evi.tar.Z`

: «Эмулятор» Emacs. Идея состоит в том, чтобы превратить `vi` в безрежимный редактор (у которого есть только режим ввода, а команды подаются управляющими клавишами). Это проделывается с помощью скрипта оболочки, заменяющего переменную окружения `EXINIT`.

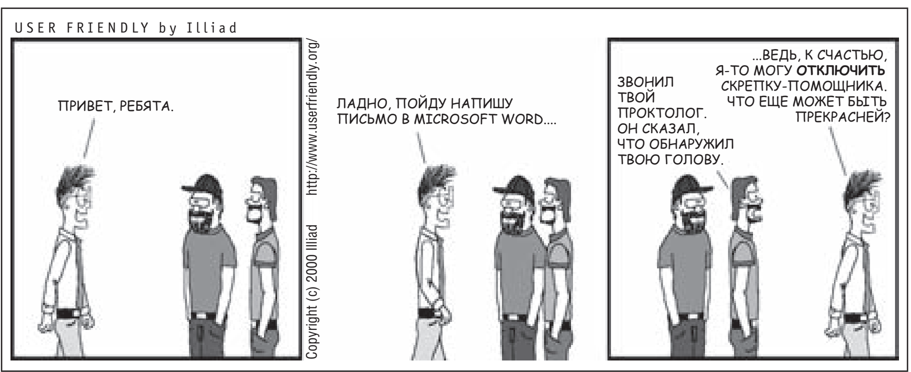



***Рис. D.6.*** *История vigor – часть V*

***Рис. D.7.*** *История vigor – часть VI*

hanoi.Z

Возможно, это самое популярное из необычных применений vi: набор макросов, решающих задачу о ханойских башнях. Программа просто показывает ходы; она не перемещает диски в действительности. Для развлечения мы перепечатали ее на врезке ниже в этом разделе.

turing.tar.Z

Эта программа использует vi для реализации настоящей машины Тьюринга! Очень познавательно смотреть, как она выполняет программы.

Есть еще очень, очень много интересных макросов, включающих режимы perl и RCS.

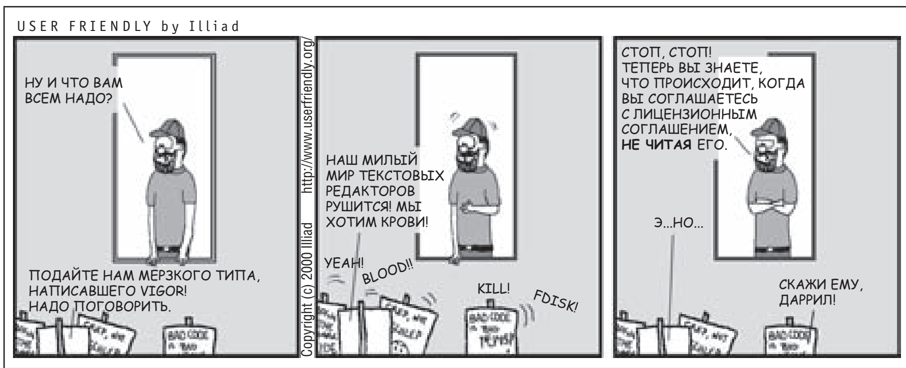



***Рис. D.8.*** *История vigor – часть VII*

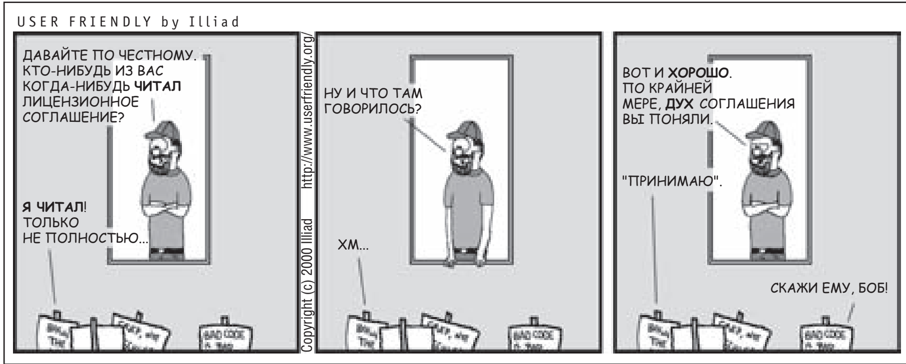



***Рис. D.9.*** *История vigor – часть VIII*

## Ханойские башни, версия vi

```
" From: gregm@otc.otca.oz.au (Greg McFarlane)
" Newsgroups: comp.sources.d,alt.sources,comp.editors
" Subject: VI SOLVES HANOI
" Date: 19 Feb 91 01:32:14 GMT
"
" Submitted-by: gregm@otc.otca.oz.au
" Archive-name: hanoi.vi.macros/part01
"
" Everyone seems to be writing stupid Tower of Hanoi programs.
" Well, here is the stupidest of them all: the hanoi solving
" vi macros.
"
" Save this article, unshar it, and run uudecode on
```

```
" hanoi.vi.macros.uu. This will give you the macro file
" hanoi.vi.macros.
" Then run vi (with no file: just type "vi") and type:
"     :so hanoi.vi.macros
"     g
" and watch it go.
"
" The default height of the tower is 7 but can be easily changed
" by editing the macro file.
"
" The disks aren't actually shown in this version, only numbers
" representing each disk, but I believe it is possible to write
" some macros to show the disks moving about as well. Any takers?
"
" (For maze solving macros, see alt.sources or comp.editors)
"
" Greg
"
" ------------ REAL FILE STARTS HERE ---------------
set remap
set noterse
set wrapscan
" to set the height of the tower, change the digit in the following
" two lines to the height you want (select from 1 to 9)
map t 7
map! t 7
map L 1G/t^MX/^0^M$P1GJ$An$BGC0e$X0E0F$X/T^M@f^M@h^M$A1GJ@f0l$Xn$PU
map g IL
map I KMYNOQNOSkRTV
map J /^0[^t]*$^M
map X x
map P p
map U L
map A "fyl
map B "hyl
map C "fp
map e "fy2l
map E "hp
map F "hy2l
map K 1Go^[
map M dG
map N yy
map O p
map q tllD
map Y o0123456789Z^[0q
map Q 0iT^[
map R $rn
map S $r$
map T ko0^M0^M^M^[
map V Go/^[
```

# Наслаждение чистым вкусом[1]

```
vi is [[13~^[[15~^[[15~^[[19~^[[18~^ a
muk[^[[29~^[[34~^[[26~^[[32~^ch better editor than this emacs. I know
I^[[14~'ll get flamed for this but the truth has to be
said. ^[[D^[[D^[[D^[[D ^[[D^[^[[D^[[D^[[B^
exit ^X^C quit :x :wq dang it :w:w:w :x ^C^C^Z^D
```

Джеспер Лоридсен (Jesper Lauridsen) с *alt.religion.emac*s

Мы не можем обсуждать vi как часть культуры UNIX, не признав, что, возможно, самым долгим дебатам в сообществе UNIX подвергалась тема сравнения vi и Emacs[2].

Например, дискуссии об этом неожиданно возникали на comp.editors (и других группах новостей) в течение нескольких лет (рис. D.10 это хорошо иллюстрирует). Итоги некоторых обсуждений можно почитать на многих веб-сайтах, упомянутых выше. Там есть ссылки и на более поздние версии страниц.

Вот некоторые аргументы в пользу vi:

- Он доступен на всех системах UNIX. Если вы устанавливаете систему либо переходите с одной на другую, то все равно сможете его использовать.
- Позволяет располагать пальцы на среднем ряду клавиатуры. Это огромный плюс для «слепой печати».
- Команды вводятся одиночными обычными символами (иногда их два). Их намного удобнее набирать по сравнению с управляющими символами и метасимволами, требуемыми Emacs.
- Как правило, vi меньше и менее требователен к ресурсам, чем Emacs. Время запуска также намного меньше, вплоть до 10 раз.
- Теперь, когда в модификациях редактора появились такие функции, как инкрементный поиск, поддержка нескольких окон и буферов, графический интерфейс, подсветка синтаксиса и умные отступы, а также программируемость через расширения, функциональный разрыв между двумя программами существенно сузился, если не пропал вовсе.

---

[1] Оригинальное название раздела «Tastes great, less filling» – знаменитый слоган рекламы пива Miller Lite 1974 года (который также можно перевести как «Больше вкуса, меньше градуса»). – *Прим. пер.*

[2] Хорошо, это действительно религиозная война, но мы пытаемся вести себя хорошо. (Другая религиозная война, BSD против System V, была прекращена POSIX. Победила System V, хотя BSD получила достаточные уступки. ☺)

***Рис. D.10****. Это не религиозная война. Нет!*

В заключение нужно упомянуть еще о двух вещах. Во-первых, на самом деле популярными являются две версии Emacs: первоначальный GNU Emacs и Xemacs, выведенный из ранней версии GNU Emacs. У обоих есть преимущества и недостатки, равно как и свой круг фанатов[1].

Во-вторых, хотя в GNU Emacs всегда были пакеты эмуляции `vi`, они, как правило, были неидеальны. Однако «режим viper» сейчас рассматривается как прекрасная эмуляция `vi`. Она может служить мостиком к изучению Emacs для всех желающих.

Наконец, не забывайте, что окончательным судьей для полезности программы выступает пользователь. Вы всегда можете применять утилиты, дающие больше производительности. `vi` и его модификации отлично справляются со многими задачами.

# Цитаты vi

Наконец, у нас есть несколько цитат о `vi` (спасибо Браму Моленару (Bram Moolenaar), автору Vim):

> ТЕОРЕМА: `vi` совершенен
>
> ДОКАЗАТЕЛЬСТВО: `VI` – это «6» римскими цифрами. Делителями числа шесть являются 1, 2 и 3. 1+2+3=6. То есть 6 является совершенным числом. Следовательно, `vi` совершенен.

Артур Татейши (Arthur Tateishi)

---

[1] И те, и другие испытывают личную неприязнь к `vi`! ☺

Реакция от Натана Т. Элгера (Nathan T. Oelger):

> Что это дает для Vim? Если смотреть на VIM как на римскую запись числа, то это может быть: (1000–(5+1))=994, что, в свою очередь, равно 2*496+2. 496 делится на 1, 2, 4, 8, 16, 31, 62, 124 и 248, а 1+2+4+8+16+31+62+124+248=496. То есть 496 – это совершенное число. Следовательно, Vim совершеннее, чем `vi`, в два раза *плюс* парочка дополнительных достоинств.
>
> Таким образом, Vim еще лучше, чем просто *совершенный*.

Следующая цитата, кажется, суммирует все необходимое для истинного ценителя `vi`:

> Для меня `vi` – это дзен. Я использую `vi` для практики дзена. Каждая команда – это коан. Проникновенный для пользователя, непостижимый для непосвященных. При каждом использовании вы постигаете истину.

Сатиш Редди (Satish Reddy)

# Алфавитный указатель

## Символы


$ (знак доллара)
команда перемещения курсора, 36, 58
метасимвол, 95
обозначение последней строки файла (ex), 80
\1, \2, ... метасимволы, 99, 200
& (амперсанд)
для повторения последней команды, 101
метасимвол, 99
' (апостроф)
'' (переход на метку), команда, 65, 73
команда перехода на метку, 73
метка строки (vile), 406
| (вертикальная черта)
альтернатива, метасимвол, 348
дизъюнкция, метасимвол, 152
для сочетания команд ex, 83
\|, метасимвол, 198
ручное сворачивание, 279
! (восклицательный знак)
cinkeys, синтаксические правила, 288
буферы, взаимодействие с, 217
для команд UNIX, 121, 122, 123
игнорирование предупреждений о сохранении, 85
отображения клавиш для режима вставки, 132
: (двоеточие)
:!, команды и, 121
команды ex и, 24, 77
режим строкового редактирования, 27
- (дефис)
для предыдущих строк файла (ex), 81
команда перемещения курсора, 34, 58
, (запятая)
для диапазонов строк (ex), 77, 79
команда повторного поиска, 63
* (звездочка)
cinkeys, синтаксические правила, 288
метасимвол, 95
? (знак вопроса)
\?, метасимвол, 366
команда поиска, 61
метасимвол, 153
– (знак минус)
буферы, описание, 217
ручное сворачивание, 279
+ (знак плюс), 422
\+, метасимвол, 198, 366
буферы, описание, 217
для следующих строк файла (ex), 81
запуск команды при старте vi, 68
команда перемещения курсора, 34, 58
метасимвол, 153
% (знак процента)
буферы, описание, 217
любой символ строки (ex), 92
обозначение текущего файла, 89
представление произвольной строки (ex), 80
совпадение скобок, 146
= (знак равно)
\=, метасимвол, 198, 366
буферы, описание, 217
команда инициализации номера строки, 80
# (знак решетки)
буферы, описание, 217
команда отображения номеров строк, 80
метаинформация, извлечение, 176
обозначение альтернативного файла, 89
^ (каретка)
метасимвол, 95
[ ] (квадратные скобки)
метасимвол, 95
[: :], метасимволы, 98
[. .], метасимволы, 97
[= =], метасимволы, 98
[[, ]] (перемещение курсора), команды, 59
/ (косая черта)
команда поиска, 60
открытие файла в указанном месте, 68
ссылка на метку (vile), 406
( ) (круглые скобки)
группировка, метасимволы, 153, 348
\(...\), метасимволы, 96, 200
найти и удалить, 131
( и ) (перемещение курсора), команды, 59
совпадение, 146
+--, маркер, метка позиции свертки, 278
` (обратная кавычка)
`` (переход на метку), команда, 64, 73
команда перехода на метку, 73
метка символа (vile), 406
\ (обратная косая черта), метасимвол, 95, 99

\1, \2, ... метасимволы, 99, 200
\d, \D, метасимволы, 399
\e, \E, метасимволы, 100
\k, \K, метасимволы, 199
\n, метасимвол, 99
\p, \P, метасимволы, 200, 399
\s, \S, метасимволы, 399
\u и \l, метасимволы, 100
\U и \L, метасимволы, 100
\w, \W, метасимволы, 399
\?, метасимвол, 399
\+, метасимвол, 198, 366, 399
\=, метасимвол, 198
\|, метасимвол, 366, 399
\(...\), метасимволы, 399
-?, опция (elvis), 356
-?, опция (vile), 384
+?, опция, 422
+/, опция, 422
_ (подчеркивание), использование в именах файлов, 26
~~ (свертки), смена регистра, 280
@ (символ at)
\@, метасимвол, 366
@, опция (vile), 384, 386
@-функции, 136
~ (тильда)
:~ (подстановка с использованием шаблона последнего поиска), команда (ex), 101
команда смены регистра, 42
метасимвол, 99, 200
свертки, 281
. (точка)
echo, команда и, 230
undo/redo (nvi), 351
метасимвол, 95
повторение команды, 48, 93
символ текущей строки (ex), 80
; (точка с запятой)
для диапазона строк (ex), 83
команда повторного поиска, 63
< > (угловые скобки)
<< (перенаправить/документ heredoc), оператор, 140
>> (перенаправить/добавить), оператор, 86
совпадение, 146
{ } (фигурные скобки)
\{...}, метасимвол, 199, 366
{ и } (перемещение курсора), команда, 59
cinkeys, опция, 287
метасимволы, 153, 348
поиск и совпадение, 146
свертка и, 273

## Числа

0 (перемещение курсора), команда, 36, 58

## A

a (добавить), команда, 37, 39, 431
ex, 440
A (добавить), команда, 50, 431
-a, опция (elvis), 355, 357
a, флаг состояния, 217
a:, переменная Vim, 228
:ab (сокращение), команда (ex), 124, 440
Acme, редактор, 21
«Address search hit BOTTOM without matching pattern», сообщение, 62
:alias, команда, 375
:amenu, команда, 260
:apropos, команда, 385
:ar, команда, 88, 441
:args, команда, 88, 441
ASCII-символы, 322
autocmd, команда, 238, 243, 246, 248
autoiconify, опция (elvis), 364
autoindent, метод, 284
autosave, опция, 71
autowrite, опция, 71, 120
awk, язык управления данными, 143

## B

:b (buffer), команда, 441
b (перемещение по словам), команда, 36
B (перемещение по словам), команда, 36
-b, опция, 321, 422
\b, метасимвол, 200
b:, переменная Vim, 228
background, опция color, 308
backupcopy, опция, 328
backupdir, опция, 328
backupnext, опция, 328
backupskip, опция, 328
«Bad file number», сообщение, 28
«Bad termcap entry», сообщение, 27
:badd, команда, 220
:ball, команда, 220
:bd (bdelete), команда, 441
:bdelete, команда, 220
:behave, команда (gvim), 255
:bfirst, команда, 220
:bg (hide window), команда (nvi), 347
binary, опция (elvis), 370
:bind-key, команда, 411
blank, параметр (опция sessionoptions), 335
:blast, команда, 220
blinktime, опция (elvis), 364
«Block device required», сообщение, 27
«Block special file», сообщение, 28
:bmod, команда, 220
:bnext, команда, 220
:bNext, команда, 220
:bprevious, команда, 220
:browse, команда, 369
bs, значения (опция lptype), 379
bufdisplay, опция (elvis), 378
bufdo, команда, 220
BufEnter, автокоманда, 208
:buffers, команда, 217, 220, 441
buffers, параметр (опция sessionoptions), 335
BufLeave, автокоманда, 208
BufNewFile, команда, 237

BufRead, команда, 237
BufReadPost, команда, 237
BufReadPre, команда, 237
BufWrite, команда, 237
BufWritePre, команда, 237
:bunload, команда, 220

## C

c (изменить), команда, 37, 431, 442
  cc, команда, 40, 41
  cw, команда, 39, 40
  примеры использования, 53, 67
C (изменить), команда, 41, 431
-c, опция, 68, 422
  elvis, редактор, 355
  nvi, редактор, 344
  vile, редактор, 384
-C, опция, 422
c$, команда, 432
C/C++, языки программирования
  cmode, режим (vile), 413
  комментарии, добавить (пример), 131
:calc, команда (elvis), 375
cc, команда, 371, 432
ccprg, опция (elvis), 371
cd, команда, 441
cedit, опция (nvi), 349
center, команда, 442
«Character special file», сообщение, 28
cindent, метод, 284
cinkeys, опция cindent, 286
cinoptions, опция cindent, 286, 290
cinwords, опция cindent, 286, 289
-client, опция (elvis), 361
clo (close), команда, 442
:close, команда (elvis), 359
:close[!], команда, 223
cmd, команда, 208, 422
  windo и bufdo, команды, 219, 220
cmdheight, опция, 216
:cnewer, команда, 318
:cnext, команда, 318
:co (копирование), команда (ex), 78, 442
:colder, команда, 318
:color, команда, 308
colorscheme, команда, 225, 227, 307, 308, 309
  глобальные переменные, использование в скриптах Vim, 234
comment, режим отображения (elvis), 372
compatible, опция, 176
:configure, команда (vile), 390
:copy, команда (ex), 78
:copy-to-clipboard, команда (xvile), 395
COSE, стандарты, 356
countzF, команда свертки, 275
:cprevious, команда, 318
cr, значения (опция lptype), 379
cscope, программа, 351
ctags, команда (UNIX), 147
  Exuberant ctags, программа, 154, 351
  стеки тегов, 157
    elvis, редактор, 367
    nvi, редактор, 351
    Solaris vi, 148, 157
    vile, редактор, 401
CTRL-, команды
  CTRL-@, 49
  CTRL-^, команда, 89
  CTRL-] (найти тег), 157, 350, 368, 401
  CTRL-A CTRL-] (следующий тег; vile), 401
  CTRL-B, CTRL-F (прокрутка), 56
  CTRL-D, CTRL-U (прокрутка), 56
  CTRL-E, CTRL-Y (прокрутка), 56
  CTRL-G (отображение номеров строк), 64, 80
  CTRL-T (найти тег), 350, 368
  CTRL-T CTRL-X CTRL-] (следующий тег; vile), 401
  CTRL-V, 127
  CTRL-V, команда (режим блоков elvis), 370
  CTRL-W, команды
    nvi, циклическое перемещение между окнами, 348
    оконные команды elvis для режима vi, 360
  CTRL-X CTRL-R, CTRL-X CTRL-L (прокрутка, vile), 164
  CTRL-X CTRL-S, CTRL-X CTRL-R (поиск, vile), 162
  завершение слов и, 294
  изменение размера окна, 214
  курсор, перемещение между окнами и, 210
curdir, параметр (опция sessionoptions), 335
CursorMoved, команда, 237, 238
CursorMovedI, команда, 237, 238
cw (change word), команда, 432
Cygwin, 297

## D

d (удалить), команда, 37, 43
  db, d$, d0, команды, 44
  dd, команда, 43, 44
  de и dE, команды, 44
  df, команда, 64
  dw, команда, 43
  в именной буфер, 48, 72
  в нумерованный буфер, 45
  нумерованные буферы, 71
  примеры использования, 53, 67
D (удалить), команда, 44
:d (удалить), команда (ex), 78
-d, опция, 422
-D, опция, 422
d$, команда, 432
date, команда (UNIX), 121
dav, 325
dd (удалить строку), команда, 281, 432
:delete, команда (ex), 78
:delete-other-windows, команда (vile), 388
:delete-window, команда (vile), 388
:describe-function, команда (vile), 385

:describe-key, команда (vile), 385
df, команда, 30, 64, 432
dG, команда, 432
di (display), команда
  elvis, редактор, 377
  nvi, редактор, 347, 350
dictionary, опция, 297
diff, команда, 203, 330
diff, метод, создание сверток, 274
directory, буфер, 218
«Directory», сообщение, 28
«Disk quota has been reached», сообщение, 31
:display mode, команда, 376
:display syntax, команда (elvis), 372
dL, команда, 432
dn, команда, 432
dt, команда, 432
dumb, значения (опция lptype), 379
dw, команда, 432
d^, команда, 432
d}, команда, 432

## E

e (перемещение курсора), команда, 59
:e, команда, 434
:e (редактировать файл), команда (ex), 89, 443
  :e!, команда, 89
\e, метасимвол, 100, 200
\E, метасимвол, 100
-e, опция, 422
:e! ENTER, команда, 29
eadirection, опция, 207, 215
«easy gvim» (MS Windows), 251
echo, команда, 227
Eclipse, 272
edcompatible, опция, 101
ed, строковый редактор, 21
ed, текстовый редактор, 22
:edit, команда, 217
Edit, команда (nvi),
  nvi, редактор, 347
else, блок, 226
elseif, блок, 226
elvis (клон vi), 344, 354
  set, команда, список опций, 464
  бесконечная отмена, 162
  будущее, 380
  важные аргументы командной строки, 355
  длина строк, 161
  инициализация, 356
  интересные особенности, 374
  получить исходный код, 380
  прокрутка в стороны, 163
  расширенные регулярные выражения, 366
  сокращения слов, 126
  суммарный список функций, 167
  улучшения, 366
  управление печатью, 378
elvis.arf, файл, 378, 379
elvis.awf, файл, 380
elvis.brf, файл, 370, 379
elvis.bwf, файл, 379
elvis.ini, скрипт, 357
elvis.msg, файл, 357, 374
elvisexhistory, буфер, 366
elvispath, опция (elvis), 357
ELVISPATH, переменная окружения (elvis), 357
Emacs, текстовый редактор, 21, 251
  vile, модель редактирования, 410
  vi, редактор против, 492
END, клавиша, отображение, 134
epson, значения (опция lptype), 379
equalalways, опция, 207, 215
:er, errlist команды (elvis), 371
errorformat, опция, 318
ESC для командного режима, 33
/etc/vi.exrc, файл (nvi), 346
:eval, команда (elvis), 375
ex, команды
  вызов команд UNIX, 121
  открытие файлов и, 479
  сохранение файлов и, 480
  стеки тегов и, 302
ex, скрипты, 137
ex, строковый редактор, 21
ex, текстовый редактор, 22, 75
  вызов с несколькими файлами, 138
  запуск буфера из, 137
  использование команд ex в vi, 24
  команды, 440
  основы, 438
  фильтрация текста, 122
«Executable», сообщение, 28
execute, команда, 230
EXINIT, переменная окружения, 117
  elvis, редактор, 357
  nvi, редактор, 346
«[Existing file]», сообщение, 30
exists(), функция, 241
expr, метод, создание сверток, 274
exrc, опция, 119, 346, 357
.exrc, файлы, 117, 119, 346, 438
  безопасность (elvis), 375
Exuberant ctags, программа, 154, 351
:exusage, команда (nvi), 345

## F

f (поиск строки), команда, 63
F (поиск строки), команда, 63
-f, опция (elvis), 355
-F, опция, 344
:f (file), команда, 443
«File exists», сообщение, 30
«File is read only», сообщение, 27, 30
«File system is full», сообщение, 30
«File to load», сообщение, 364
filec, опция (nvi), 349
:files, команда, 217, 220
FileType, команда, 237
:find-file, команда (vile), 388
«First address exceeds second», сообщение, 82

firstx, firsty, опции (elvis), 364
fold, команда, 443
foldc, команда, 443
foldcolumn, поле, 281
foldenable, установить, 283
foldlevel. команда, 282
foldo, команда, 444
folds, параметр (опция sessionoptions), 335
for, цикл, 138
Fred Fish disk, 172
FreeBSD, 297
FTP, 325
function, режим отображения (elvis), 372
function...endfunction, конструкция, 231

## G

:g (глобальная замена), команда (ex), 93, 444
    метасимволы, в строках замены, 98
    метасимволы, поиск по шаблону, 95
    повторение команд, 113
    примеры использования шаблонов, 102
    сбор строк, 114
    трюки при заменах, 101
:g (глобальный поиск), команда (ex), 83
G (переход), команда, 64
-g, опция, 270
    gvim, 252
g, опция (команда :s), 92
-G, опция (elvis), 355
g:, Vim, переменная, 228
gg, опция, 430
gI, команда, 431
gJ, команда, 433
globals, параметр (опция sessionoptions), 335
GNU Emacs, текстовый редактор, 21
gp, команда, 432
gP, команда, 433
gqap, команда, 432
:gui, команда, 270
    elvis, 363
GUI-интерфейсы
    elvis, редактор, 360, 372
    gvim, 251
    vile, редактор, 389
guicursor, опция, 270
guifont, опция, 270
guifontset, опция, 270
guifontwide, опция, 270
guiheadroom, опция, 270
guioptions, опция, 257, 270
guipty, опция, 270
guitablabel, опция, 270
guitabtooltip, опция, 270
guw, команда, 432
gUw, команда, 432
gvim, 209, 251
    запуск, 252
    изменение размера окна и, 213
    меню, 255
    редактирование с вкладками, 222
$GVIMINIT, переменная окружения, 253
.gvimrc, файл инициализации, 252
    colorscheme, команда и, 225
    массивы и, 236
    функции, определение, 232
gzip, утилита, 168
g~w, команда, 432

## H

H (начало), команда, 57
-h, опция, 422
    vile, редактор, 384
h (перемещение курсора), команда, 34, 58
h, флаг состояния, 217
help, 384
:help, команда, 217, 218
help, буфер, 218
help, параметр (опция sessionoptions), 335
--help, опция, 206
hex, режим отображения (elvis), 370, 376
hid (hide), команда, 444
highlight, команда, 309
:historical-buffer, команда (vile), 389
[History], буфер (vile), 400
HOME, клавиша, отображение, 134
$HOME/.nexrc, файл (nvi), 346
horizscroll, опция, 163
hp, значения (опция lptype), 379
HTML, 329
html, режим отображения (elvis), 368, 376

## I

i (вставить), команда, 33, 431, 444
I (вставить), команда, 50, 431
-i, опция, 423
    elvis, редактор, 355
i, флаг, gvim mouse, опция, 253
\i, \I, метасимволы, 199
ibm, значения (опция lptype), 379
ic, опция, 100, 117
IDEs (интегрированные среды разработки), 174
if...then...else, конструкция, 226, 235
ignorecase, опция, 120
include, файл (C), 299
:incremental-search, команда (vile), 405
incsearch, опция
    elvis, редактор, 162
    Vim, редактор, 162
indent, метод, создание сверток, 274
indentexpr, метод, 285
inputtab, опция (elvis), 367
insert (i), команда, 444
isfname, опция (Vim), 199, 200
isident, опция (Vim), 199, 200
iskeyword, опция (Vim), 297
isprint, опция (Vim), 200

## J

J (объединение), команда, 52, 433
j (перемещение курсора), команда, 34, 58
ju (jump), команда, 444

## K

k (перемещение курсора), команда, 34, 58
keyword, режим отображения (elvis), 372

## L

L (конец), команда, 57
l (перемещение курсора), команда, 34, 58
\l, метасимвол, 100
\L, метасимвол, 100
-l, опция, 423
-L, опция, 423
l:, Vim, переменная, 228
:last, команда (elvis, Vim), команда, 88
LaTeX, издательская система, 23
leftright, опция (nvi), 163, 352
:let, команда, 241
linewrap, опция (vile), 163
Linux, Vim для, 178
:loadview, команда, 274
localoptions, параметр (опция sessionoptions), 335
lpcolor, опция (elvis), 378
lpcolumns, опция (elvis), 378
lpcontrast, опция (elvis), 378
lpconvert, опция (elvis), 378
lpcrlf, lpc, опции (elvis), 378
lpformfeed, lpff, опции (elvis), 378
lplines, опция (elvis), 378
lpoptions, lpopt, опции (elvis), 378
lpout, lpo, опции (elvis), 378
:lpr, команда (elvis), 378
lprows, опция (elvis), 378
lptype, опция (elvis), 378
lpwrap, lpw, опции (elvis), 378
:ls, команда, 217, 220

## M

m (пометка места), команда, 73
M (середина), команда, 57
-m, опция, 423
-M, опция, 423
:m (перемещение), команда (ex), 78
Mac OS X, установка Vim, 177, 183
magic, опция, 120
Make, кнопка (elvis), 363
:make, команда (elvis), 363, 371, 372
make, программа, 315
makeprg, опция, 318
    elvis, редактор, 371
man, режим отображения (elvis), 376
manual, метод, создание сверток, 274
:map, команда (ex), 126, 445
    команды в .exrc-файлах, 119
    примеры использования, 129
marker, метод, создание сверток, 275
:menu, команда, 260, 266, 270
mini-hilite, опция (vile), 400
mksession, команда, 334
:mkview, команда, 274
modeline, опция, 321
:modeline-format, команда (vile), 417
Mortice Kern Systems, 143
:move, команда (ex), 78
:move-next-window-down, команда (vile), 388
:move-next-window-up, команда (vile), 388
:move-window-left, команда (vile), 388
:move-window-right, команда (vile), 388
MS Windows, использование gvim, 251, 269
$MYGVIMRC, переменная, 253

## N

:n (следующий файл), команда (ex), 88
n (повторный поиск), команда, 61, 93
N (повторный поиск), команда, 61
n, флаг, (mouse, опция), 254
\n, метасимвол, 200
-n, опция, 423
-N, опция, 423
    vile, редактор, 384
:new, команда, 208, 358, 435, 446
«[new file]», сообщение, 27
NEXINIT, переменная окружения, 346
.nexrc, файл (nvi), 346
:Next, команда (nvi), 347
:next-tag, команда (vile), 401
:next-window, команда (vile), 388
noexpandtab, опция, 321
noh, команда, 447
noignorecase, опция, 120
nolinewrap, опция (vile), 405
nomagic, опция, 120
:no (:normal), команда (elvis), 376, 377
«Non-ascii file», сообщение, 28
nonu (nonumber), опция, 80
--noplugin, опция, 423
normal, режим отображения (elvis), 376
«Not a typewriter», сообщение, 27
«No Toolkit», сообщение, 389
«No write since last change», сообщение, 28, 84
notagstack, опция (elvis), 368
nowrap, опция, 336
    elvis, редактор, 370
nowrapscan, опция, 62, 120
nroff, пакет форматирования, 23
nu, опция, 36, 447
num, команда, 432
nvi (клон vi), редактор, 343
    set, команда, список опций, 462
    длина строк, 161
    документация и онлайн-справка, 345
    инициализация, 346
    интересные функции, 352
    многооконное редактирование, 346
    получить исходный код, 353
    прокрутка в стороны, 163
    сокращения слов, 126
    стеки тегов, 350

## O

o (открыть строку), команда, 50, 431
O (открыть строку), команда, 50, 431
-o, опция, 423
    elvis, редактор, 355

-O, опция, 423
:only[!], команда, 224
options, параметр (опция sessionoptions), 335
other, режим отображения (elvis), 373

## P

:p (вставка), команда (ex), 90
p (вставка), команда, 37, 45, 46, 47
  из именных буферов, 48, 72, 89
P (вставка), команда, 46
  из именных буферов, 48, 72, 89
:p (print), команда (ex), 76, 448
PAGE UP, PAGE DOWN, клавиши, отображение, 134
pana, значения (опция lptype), 379
:paste-from-clipboard, команда (xvile), 395
PATH, переменная окружения, установка Vim, 177
«Pattern not found», сообщение, 60
«Permission denied», сообщение, 27, 30
pin-tagstack, опция (vile), 402
:pop (:po), команда, 367, 401
:position-window, команда (vile), 388
POSIX, стандарт, 173
:pre, команда, 447
  ex, 31, 70
prep, режим отображения (elvis), 373
prev, команда, 448
:Previous, команда (nvi), 347
:previous-window, команда (vile), 388
ps, ps2, значения (опция lptype), 379
:pu (put), команда, 448

## Q

:q (выход), команда (ex), 29, 84
  :q!, команда, 85
Q, команда, 78
:q (выделенное перемещение), команда (vile), 405
:q!, команда, 425
qa, команда, 448
:qall, команда (elvis), 359
quickfix, буфер, 218
Quickfix List, окно, 315
Quit, кнопка (elvis), 363
:quit, команда, 223

## R

:r (чтение), команда (ex), 86, 448
r (замена символа), команда, 41, 42, 51
R (замена символа), команда, 42, 51, 431
\r, метасимвол, 200
-r, опция, 70
  elvis, редактор, 355
  nvi, редактор, 344
-R, опция, 70, 423
  vile, редактор, 384
rcp (удаленное копирование), 325
:read, команда (ex), 86, 121
read-hook, опция (vile), 416
«[Read only]», сообщение, 27
rec, команда, 449
red, команда, 449
res, команда, 449
:resize, команда, 214, 215, 347
resize, параметр (опция sessionoptions), 335
:resize-window, команда (vile), 388
:restore-window, команда (vile), 388
:reverse-incremental-search, команда (vile), 405
rew, команда, 449
:rew, :rewind, команды (ex), 88
rm, команда UNIX, 30
ruler, опция, 164

## S

s (подстановка), команда, 42, 51, 52, 431
S (подстановка), команда, 42, 51, 52, 431
s (подстановка), команда (ex), 77, 91, 412
  контекстно-зависимая замена, 93
  метасимволы в строках замены, 98
  метасимволы для поиска по шаблону, 95
  примеры использования шаблонов, 102
  трюки при заменах, 101
-s, опция, 424
  elvis, редактор, 355
  nvi, редактор, 344
  vile, редактор, 384
-S, опция, 424
-SS, опция (elvis), 356
\s, \S, метасимволы, 200
s:, Vim, переменная, 228
:safely, команда (elvis), 375
:sall (:sa), команда (elvis), 359
sam, редактор, 21
:save-window, команда (vile), 388
sb, команда, 450
:sball, команда, 220
:sbfirst, команда, 220
:sblast, команда, 220
:sbmod, команда, 220
sbn, команда, 450
:sbnext, команда, 220
:sbNext, команда, 220
:sbprevious, команда, 220
:sbuffer, команда, 220
scp (безопасное удаленное копирование по SSH), 325
scratch, буфер, 218
:scroll-next-window-down, команда (vile), 388
:scroll-next-window-up, команда (vile), 389
se, команда, 450
searchincr, опция (nvi), 162, 352
sed, потоковый редактор, 143
sesdir, параметр (опция sessionoptions), 335
sessionoptions, опция, 334
:set, команда, 117, 118, 119, 176, 254, 437, 458
:set-window, команда (vile), 389
:sfind, команда, 208
sftp (безопасный FTP), 325
:sh, команда (ex), 30, 31, 450
:sh (создать оболочку), команда (ex), 121

shiftwidth, использование режимов отступа, 282
shmode, режим (пример; vile), 413
:show-commands, команда (vile), 385
:show-history, команда (vile), 400
showmode, опция, 164
:show-tagstack, команда (vile), 401
:shrink-window, команда (vile), 389
sidescroll, значение, 163, 370
sidescroll, опция (nvi), 352
sidescrolloff, опция, 337
:skeyword, команда, 199, 200
slash, параметр (опция sessionoptions), 335
:slast (:sl), команда (elvis), 359
smartindent, метод, 284
sn, команда, 450
:snew (:sne), команда (elvis), 358
:sNext (:sN), команда (elvis), 359
:so, команда (ex), 120
Solaris vi
  set, команда, список опций, 458
  сокращения слов, 126
  стеки тегов, 148
sort, команда (UNIX), 121
sp, команда, 451
Split, кнопка (elvis), 363
:split, команда, 205, 207, 217, 358, 387
:split-current-window, команда (vile), 387, 389
spr, команда, 451
:srewind (:sre), команда (elvis), 359
st, команда, 451
:stack (:stac), команда (elvis), 367
:stag (:sta), команда (elvis), 359
:stag[!], тег, 221
statusline, опция, 232
stevie, редактор, 172, 354
stopshell, опция (elvis), 364
strftime( ), функция, 227
string, режим отображения (elvis), 373
sts, команда, 303
stty, команда, 24, 25
su, команда, 452
substitute (:s), команда (ex)
  vile, редактор, 412
:sunhide, команда, 220
sv, команда, 452
:sview, команда, 208
:syntax, команда, 305
syntax, метод, создание сверток, 274
syntax, режим отображения (elvis), 376

## T

:t (копирование), команда (ex), 78
t (поиск строки), команда, 63
T (поиск строки), команда, 63
^T, команда, 304
\t, метасимвол, 200
-t, опция, 424
  elvis, редактор, 356
  nvi, редактор, 345
  vile, редактор, 384
-T, опция, 424
t:, Vim, переменная, 228
ta, tag, команды (nvi), 350
:Ta, Tag, команды (nvi), 350
:ta, tag команды (Solaris vi), 157
<TAB>, использование в пунктах меню, 262
:tabclose, команда, 222
:tabnew, команда, 222
:tabonly, команда, 222
tabpages, параметр (опция sessionoptions), 336
tag, (:ta), команда
  elvis, редактор, 367
  vile, редактор, 401
:tag, команда, 147, 157, 368
tagignorecase, опция (vile), 402
taglength, опция, 351
  elvis, редактор, 368
  Solaris vi, 158
  vile, редактор, 402
:tagp, tagpop команды (nvi), 350
tagpath, опция
  elvis, редактор, 368
tagprg, опция (elvis), 369
tagrelative, опция (vile), 402
tags, опция
  elvis, редактор, 368
  nvi, редактор, 350
  vile, редактор, 402
tags, формат файла, 155, 351
tagstack, опция
  elvis, редактор, 368
:tagt, tagtop команды (nvi), 350
tagword, опция (vile), 402
TERM, переменная окружения, 25, 27
  открытие файлов и, 479
termcap, библиотека, 24
termcap, данные, 25
«Termcap entry too long», сообщение, 27
terminfo, библиотека, 24
terminfo, данные, 25
TeX, издательская система, 23
tex, режим отображения (elvis), 376
textwidth, опция, 321
thesaurus, опция, 297
tl, (taglength), опция
  Solaris vi, 158
:tmenu, команда, 268
/tmp (специальное имя файла, nvi), 353
:toggle-buffer-list, команда (vile), 389
TOhtml, команда, 329
toolbar, опция, 270
:topleft, команда, 208
troff
  пакет форматирования, 23
  поместить вокруг слова код жирного шрифта, 130
  преобразование глоссария к формату (пример), 128
  сортировка глоссария в алфавитном порядке (пример), 141

:tselect, команда, 221, 303, 304

## U

u (отмена), команда, 45, 49
восстановление буфера, 72
U (отмена), команда, 45
-U gvimrc, опция, 270
\u, метасимвол, 100
\U, метасимвол, 100
-u, опция, 424
u, флаг состояния, 217
undolevels, опция, 332
elvis, редактор, 369
undolimit, опция (vile), 402
:unhide, команда, 220
UNIX
Vim, установка, 178
команды, 121
unix, параметр (опция sessionoptions), 336
unm, команда, 453
/usr/tmp, каталог, 30

## V

-v, опция, 424
vile, редактор, 384
-V, опция, 424
-V, опция (elvis), 356
v, V, команды (режим блоков elvis), 370
v:, Vim, переменная, 229
v:fname_in, переменная, 331
v:fname_new, переменная, 331
v:fname_out, переменная, 331
/var/tmp, каталог, 30
variable, режим отображения (elvis), 373
:version, команда, 177
--version, опция, 424
:vertical, команда, 216
:vi, команда, 78, 425
vi, команда (UNIX)
опции командной строки, 68
vi, текстовый редактор
настройка среды редактирования, 117
фильтрация текста, 122
vi.exrc, файл (nvi), 346
view, команда (UNIX), 70
:view-file, команда (vile), 389
vile (клон vi), 71, 382
set, команда, список опций, 472
длина строк, 161
документация и онлайн-справка, 384
интересные особенности, 410
прокрутка в стороны, 163
расширенные регулярные выражения, 398
сокращения слов, 126
VILEINIT, переменная окружения (vile), 386
.vilemenu, файл, 386, 395
VILE_HELP_FILE, переменная окружения (vile), 386
VILE_STARTUP_FILE, переменная окружения (vile), 386
VILE_STARTUP_PATH, переменная окружения (vile), 385
Vim, 169, 229
set, команда, список опций, 466
длина строк, 161
многооконность в, 202
прокрутка в стороны, 163
расширенные регулярные выражения, 198
сокращения слов, 126
vimdiff, команда, 284, 330
viminfo, опция, 333
.vimrc, файл инициализации, 252
strftime(), функция и, 227
VimResized, команда, 237
visual, 405
«Visual needs addressable cursor or upline capability», сообщение, 27
Visual Studio, 272
:viusage, команда (nvi), 345
:vnew, команда, 208
:vsplit, команда, 206, 208

## W

w (перемещение по словам), команда, 36
W (перемещение по словам), команда, 36
:w (запись), команда, 70, 88
:w (запись), команда (ex), 30, 84
:w!, команда, 85
переименование буфера, 85
сохранение фрагмента файла, 85
^W, команда
курсор, перемещение между окнами и, 210
^W, сочетания клавиш с, 205
-w, опция, 424
nvi, редактор, 345
-W, опция, 425
w:, Vim, переменная, 228
w!, команда перезаписи файла, 30
^Wc, команда, 223
^Wf, команда, 221
^WF, команда, 221
^Wg], команда, 221
^Wg^], команда, 221
^WH, команда, 212
^WJ, команда, 212
^WK, команда, 212
^WL, команда, 212
^Wo, команда, 224
^W^O, команда, 224
^Wq, команда, 223
^W^Q, команда, 223
^Wr, команда, 213
^W^R, команда, 213
^WR, команда, 213
^Ws, команда, 207
^W^S, команда, 207
^WS, команда, 207
^WT, команда, 212
^Wx, команда, 213
^W^X, команда, 213
^W^_, команда, 216

^W^], команда, 221
^W_, команда, 216
^W-, команда, 216
^W], команда, 221
^W+, команда, 216
^W<, команда, 216
^W=, команда, 216
^W>, команда, 216
^W|, команда, 216
windo, команда, 219
window, опция, 118
:window (:wi), команда (elvis), 359
Windows, файлы, редактирование в vile, 416
WinEnter, команда, 237
winheight, опция, 205, 215
WinLeave, команда, 237
winminheight, опция, 216
winminwidth, опция, 216
winpos, параметр (опция sessionoptions), 336
winsize, параметр (опция sessionoptions), 336
winvile, редактор, 396
winwidth, опция, 205, 215
wm (wrapmargin), опция, 35, 120
  отключить при большой вставке, 131
  повторение больших вставок, 49
:wq, команда, 29
:wquit, команда (elvis), 359
wrap, опция, 336
  elvis, редактор, 163
wrapmargin (wm), опция, 35, 120, 321
  отключить при большой вставке, 131
  повторение больших вставок, 49
wrapscan, опция, 62, 69, 120
writebackup, опция, 328
write-hook, опция (vile), 416

## X

x (удаление символа), команда, 45, 433
X (удаление символа), команда, 45, 433
:x (записать и выйти), команда (ex), 84, 425
-x, опция, 425
X, ресурсы для elvis, 365
X Window System, 21
  использование gvim, 252, 269
X11, интерфейс
  elvis, 357, 360, 364
  vile, 389
XEmacs, текстовый редактор, 21
xscrollbar, опция (elvis), 365
xvile, редактор, 390
XVILE_MENU, переменная окружения (vile), 386

## Y

Y (копировать строку), команда, 47, 433
y (копировать), команда, 37, 47
  yy, команда, 47
  в именованный буфер, 48, 72
  в нумерованный буфер, 47
  примеры использования, 53, 67
y (копировать), команда (ex), 90
-y, опция, 425
y$, команда, 433
ye, команда, 433
yw, команда, 433
yy, команда, 433

## Z

z, команда, 56
-Z, опция, 425
za, команда свертки, 276
zA, команда свертки, 275
zc, команда свертки, 276, 280, 281
zC, команда свертки, 275
zd, команда свертки, 276
zD, команда свертки, 275
zE, команда свертки, 275, 283
zf, команда свертки, 275
zi, команда свертки, 276
zj, команда свертки, 276
zk, команда свертки, 276
zm, команда свертки, 276, 282
zM, команда свертки, 275
zn, команда свертки, 276
zN, команда свертки, 276
zo, команда, 281
zo, команда свертки, 276
zO, команда свертки, 276
zr, команда свертки, 276, 282
ZZ, команда, 434
ZZ, команда выхода из vi, 28

## А

абзацы
  перемещение по, 59
  разделители, 59
абсолютные адреса строк, 79
абсолютный путь, 26
автозавершение ввода, 293
автозавершение ключевых слов, 176
автозавершение команд, 160, 161
автокоманды, 237
альтернатива, 198, 348, 366
альтернативные .exrc-файлы, 119
архивы vi (FTP), 486

## Б

база данных
  перестановка записей (пример), 111
безопасность, elvis, 375
бесконечная отмена, 162
  nvi, редактор, 351
блоков (визуальный) режим
  elvis, редактор, 370
  vile, редактор, 405
большие вставки, 49, 131, 161
буфер обмена, Windows, 269
буферы, 25, 71
  autosave и autowrite, опции, 71
  взаимодействие с окнами, 217
  восстановление после системного сбоя, 70
  временный буфер (метасимволы), 200

запуск содержимого, 136
именованные буферы, 48, 71, 89
произвольные имена (nvi), 353
команды, 219, 366
обзор, 74
копирование в буфер содержимого файла, 86
многооконное редактирование и, 151, 203
нумерованные буферы для удаления/копирования, 45, 71
переименование (ex), 85
сохранение вручную, 70
специальные, 218

## В

веб-сайты vi, 483
ветвление отмен, 332
визуальный режим (блоков)
elvis, редактор, 370
vile, редактор, 405
визуальный режим, 254
проблемы попадания в, 482
вкладки, редактирование, 222
вложенные свертки, 274
внутренние функции, 248
восстановление удалений, 45, 71
временный буфер, 96
вставка текста, 37, 50
a (добавить), команда, 37, 39
в режиме вставки, 33
из именованных буферов, 48, 72, 89
повторение вставки с CTRL-@, 49
поддержка больших вставок, 49, 131, 161
с копированием, копировать-и-вставить, 47
с удалением, вырезать-и-вставить, 46
встроенный калькулятор (elvis), 375
вывод (UNIX), считывание в файлы, 121
выделение текста в xvile, 394
вызов vi, с несколькими файлами, 87
выражения, 247
вырезать-и-вставить, 37, 45
многооконность в Vim и, 202
выход, 29
выход из ex (в vi), 78
выход из vi, 28, 84

## Г

главные режимы (major modes), vile, 412
глобальная замена, 91
подтверждение заменам, 92
поиск по шаблону, 94
правила соответствия шаблону
метасимволы в строках замены, 98
метасимволы для поиска по шаблону, 95
трюки при заменах, 101
примеры, 102
глобальный поиск (ex), 83
глоссарий, преобразование к формату troff (пример), 128
горизонтальная прокрутка
elvis, редактор, 370
nvi, редактор, 352
vile, редактор, 405
графический интерфейс пользователя (GUI), 175
клоны vi, 152
группы (подсветка синтаксиса), 307

## Д

двоичные данные, редактирование, 161
elvis, редактор, 370
nvi, редактор, 352
vile, редактор, 403
двоичные файлы, редактирование, 320
дефис (-)
для предыдущих строк файла (ex), 81
команда перемещения курсора, 58
диапазон строк, 79, 83
диграфы, 322
дизъюнкция, 152
добавление текста, 37, 39
к именованным буферам, 72
к сохраненным файлам, 86
документация
elvis, редактор, 356
nvi, редактор, 345
vile, редактор, 384
документы heredoc, 140

## З

завершение команд
elvis, редактор, 366
nvi, редактор, 349
vile, редактор, 400
зависимость от регистра, 24
заглавные буквы, смена на строчные, 42
заголовочный файл (C), 300
закладка, установка, 73
замена текста, 37, 39
глобально, 91
метасимволы в строках замены, 98
подтверждение заменам, 92
трюки при заменах, 101
по символам, 41
по словам, 40
по строкам, 40
при поиске, 62
записи termcap и terminfo, 117
запись буфера
autosave и autowrite, опции, 71
отмена режима «только чтение», 70
запуск vi, опции командной строки, 68
запуск текста из буфера, 136
знак доллара ($)
команда перемещения курсора, 36, 58
метасимвол, 95
обозначение последней строки файла (ex), 80
отметка конца строки, 39

знак плюс (+)
для следующих строк файла (ex), 81
запуск команды при старте vi, 68
команда перемещения курсора, 58
метасимвол, 153, 348
знак процента (%)
любой символ строки (ex), 92
метаинформация, извлечение, 176
обозначение текущего файла, 89
представление произвольной строки (ex), 80
совпадение скобок, 146
знак равно (=), команда инициализации номера строки, 80
знак решетки (#)
команда отображения номеров строк, 80
метаинформация, извлечение, 176
обозначение альтернативного файла, 89

## И

изменение (замена) текста, 37
глобально, 91
метасимволы в строках замены, 98
подтверждение заменам, 92
трюки при заменах, 101
по символам, 41
по словам, 40
по строкам, 40
при поиске, 62
именованные буферы, 48, 71, 89
запуск содержимого, 136
инициализация
nvi, редактор, 346
vile, редактор, 386
Vim, 175
инкрементный поиск, 162
nvi, редактор, 352
vile, редактор, 404
инструменты, программирование, 272
интегрированная среда разработки (IDE), 272
Интернет, vi и, 483
интерфейсы для клонов vi, 152
elvis, редактор, 360
история, командная строка, 160
elvis, редактор, 366
vile, редактор, 400
исходник, найденный файл инициализации, 253
исходный код, редактирование, 143
использование тегов, 147, 154
контроль за отступами, 143
совпадение скобок, 146

## К

кавычка (XXX_DQUOTE), команда, 72
каретка (^)
команда перемещения курсора, 58
метасимвол, 95
каталоги, переход и смена, 326
квадратные скобки ([ ])
метасимвол, 95
[. .], метасимволы, 97
[= =], метасимволы, 98
[[, ]] (перемещение курсора), команды, 59
совпадение, 146
клавиши
backspace
перемещение, 34
удаление в режиме вставки, 33
Caps Lock, 52
Enter
новая строка в режиме вставки, 35
перемещение, 34, 58
ESC, командный режим, вход, 28
запоминание при помощи команды :map, 126
полезные примеры, 129
функциональные и специальные клавиши, 133
классы символов, 97
классы эквивалентности, 98
клоны vi, 149
GUI-интерфейсы
elvis, редактор, 360, 372
vile, редактор, 389
set, команда, список опций, 458
графические интерфейсы, 152
многооконное редактирование, 151
nvi, редактор, 346
поддержка программистов
elvis, редактор, 371
vile, редактор, 407
регулярные выражения
elvis, редактор, 366
nvi, редактор, 348
vile, редактор, 398
Vim, редактор, 198
суммарный список функций, 167
улучшенные возможности, 160
улучшенные теги, 154
ключевые слова, автозавершение, 293
коды форматирования, 22
команда vi (UNIX)
редактирование нескольких файлов, 87
командная строка
инициализация многооконности из, 203
опции, 68, 422
elvis, редактор, 355
nvi, редактор, 344
vile, редактор, 383
параметры, 189
синтаксис, 421
командный режим, 22, 23, 28, 32, 425
gvim, использование мыши, 253
индикаторы режима, 164
отображение комбинаций клавиш, 126
полезные примеры, 129
функциональные и специальные клавиши, 133
команды, 22, 482
cw, изменить слово, 23
ex, 29

execute, 230
i (вставка), 23
wq, сохранение и выход, 29
w!, перезапись файла, 30
w, сохраниение файла, 29
автокоманды
группы, 242
удаление, 243
в .exrc файлах, 119
в нижней строке, 24
вставки, 431
завершения вставки, 294
перемещения, 428
работа с окнами (Vim), 434
сокращения в Vim, 338
сохранение, 124
строки состояния, 427
команды ex, 29, 76
адреса строк, 76, 79
диапазон строк, 79
адресация строк
относительная адресация, 81
переопределение текущей строки, 82
символы адресации, 80
номера строк, 79
сохранение и выход, 28, 84
сочетания команд, 83
команды vi, 32
запуск при старте vi, 68
общий вид, 41
числовые аргументы команд, 35, 51
комментарии
в скриптах ex, 142
поместить символы вокруг строки (пример), 131
компиляция исходного кода программ
elvis, редактор, 371
контекст начала строки, 287
контекстно-зависимая глобальная замена, 93
контекст сеанса Vim, 175
контроль за отступами, 144
конфигурационные файлы gvim, 252
копии файлов, работа с буфером, 25
копирование текста, 37, 47
в именованный буфер, 48, 72, 89
в нумерованный буфер, 47
копировать-и-вставить, 37
по строкам, 78
копирование файла в другой файл, 86
кофейные кружки с логотипом vi, 485
круглые скобки ( )
\(...\), метасимволы, 200
метасимволы ..., 96
найти и удалить, 131
( и ) (перемещение курсора), команды, 59
совпадение, 146
курсорные клавиши, 34
курсор, перемещение, 33, 57
xvile, интерфейс, 394
команды, 65
открытие файла в указанном месте, 68
поиском по шаблону, 60
по меткам, 73
по текстовым блокам, 59

## Л

логотип «vi Powered», 484
локальные .exrc-файлы, 119, 120, 346

## М

макросы, 436
массивы (Vim), 235
меню
Windows, 255
использование в gvim, 255, 258
настройка, 263
поддержка в xvile, 395
метаинформация, 176
метасимволы, 94
расширенные регулярные выражения
nvi, редактор, 348
Vim, редактор, 198
метки (визуальный режим vile), 406
многооконное редактирование
elvis, редактор, 358
vile, редактор, 387
инициализация, 203
Моленар, Брам (Moolenaar, Bram), 171, 493
Морган, Кларк (Morgan, Clark), 382

## Н

настройка среды редактирования, 117
не-ASCII символы, 322
непечатаемые символы
перевод строки, 35
пробелы в именах файлов, 76
разделители предложений, 59
удаление слов с ними, 44
номера строк, 36
в командах ex, 76
диапазон строк, 79
относительная адресация, 81
переопределение текущей строки, 82
отображение, 64, 80
перемещение, 64
символы для номеров, 80
нормальный режим (gvim), 254
нумерованные буферы для удаления/копирования, 45, 71

## О

обозначение альтернативного файла (#), 89
обозначение текущего файла (%), 89
обратная кавычка (`)
команда перехода на метку, 73
`` (переход на метку), команда, 64, 73
обратная косая черта (\), метасимвол, 95
\u и \l, метасимволы, 100
обратный поиск, 61
объединение строк, 52
однострочная, 280

окна, 206
закрыть, выйти, 223
изменение размера, 213
курсор, перемещение между, 209
перемещение, 211
разделенные по вертикали, 206
онлайн-справка
nvi, редактор, 345
vile, редактор, 384
учебник по vi, 486
опции
set, команда, 117
текущие установки, 118
команды vi, 68
типа переключателя (ex), установка значений, 117
открытие файлов
в указанном месте, 68
несколько файлов сразу, 87
предыдущий файл, 89
режим «только чтение», 70
открытый режим (elvis), 374
отмена, 49
бесконечная (клоны vi), 162
elvis, редактор, 369
nvi, редактор, 351
vile, редактор, 402
восстановление удалений, 72
удаления текста, 45
отмена отмен, 332
относительные адреса строк (ex), 81
относительный путь, 26
отображение команд, 24
отображения, 126
для режима вставки, 132
именованный буфер, содержимое как, 136
полезные примеры, 129
функциональные и специальные клавиши, 133
ошибки, компиляция и поиск, 314

## П

панели инструментов, 266
elvis, 363
переименование буфера (ex), 85
переменные, 228
буфера, 240
глобальные, использование в скриптах Vim, 234
типы, 246
перемещение
между несколькими файлами, 88
перестановка записей в базе данных (пример), 111
пронумерованное удаление/буфер копирования, 46
строк, 78
текста (удалить-и-вставить), 37
текстового блока по шаблону, 103
перемещение курсора, 34, 57
команды, 65
открытие файла в указанном месте, 68
поиском по шаблону, 60
по меткам, 73
по текстовым блокам, 36, 59
перенаправление в vile, 416
перенос поиска, 60, 62
перерисовка экрана, 57
перестановка
записей в базе данных (пример), 111
символов, 46
слов, 47
переформатирование текста (vile), 416
переходы между состояниями, Vim, 176
печать строк, 76
поведение мыши
elvis, редактор, 362
gvim, 253
повторение команд, 48
глобальные замены, 101
поиск в нумерованных буферах, 72
поиск по шаблону, 61, 63
с помощью :g, 113
поддержка интернационализации
elvis, редактор, 374
nvi, редактор, 352
поддержка программистов, 165, 272
использование тегов, 147, 154
исходный код, редактирование
использование тегов, 147, 154
контроль за отступами, 143
совпадение скобок, 146
подсветка синтаксиса, 166
elvis, режимы отображения, 376
редактор Vim, 174
ускорение цикла редактирование-компиляция, 165
elvis, редактор, 371
vile, редактор, 407
подсветка синтаксиса, 166, 305
elvis, редактор, 408
режимы отображения, 372
настройка, 306
подстановка (:s), команда (ex), 77, 91
контекстно-зависимая замена, 93
метасимволы в строках замены, 98
метасимволы для поиска по шаблону, 95
примеры использования шаблонов, 102
трюки при заменах, 101
подтверждение заменам, 92
поиск
в нумерованных буферах, 72
метасимволов, 95
ошибок, vile, 407
по шаблону, 60
в строке, 63
глобальные правила соответствия шаблону
метасимволы в строках замены, 98
метасимволы для поиска по шаблону, 95

примеры использования шаблонов, 102
трюки при заменах, 101
игнорирование регистра при поиске по шаблонам, 117
инкрементный поиск (клоны vi), 162
настройка, опции для, 120
открытие файла в указанном месте, 68
перенос поиска, 60, 62
совпадение скобок, 146
по шаблону строк
команды ex для поиска, 81, 83
полосы прокрутки
gvim, 257
xvile, 393
получить исходный код, vile, редактор, 417
поля
повторение больших вставок, 49
установка, 35
поменять местами слова (пример), 127, 129
пометка места, 73
последняя строка файла
переход, 64
символ для (ex), 80
постобработка (Vim), 176
постчтения, постзаписи, файлы (elvis), 357, 379
правое поле, установка, 35
предложения
перемещение по предложениям, 59
разделители, 59
предчтения, предзаписи, файлы (elvis), 357, 379
предыдущий файл, переключение в, 89
преобразование регистра, 42
приглашение командной строки, 26
пробел, отступы, 144
пробелы в имени файла, 26
проверка орфографии, 23
прозрачное редактирование, 176
прокрутка, 56
elvis, редактор, 370
nvi, редактор, 352
vile, редактор, 405
без перемещения курсора, 56
прописные буквы, смена на строчные, 100
процедурный язык, vile, 415

## Р

разделитель в пути к файлу, 26
разделы, перемещение по, 59
разрешение на запись, 28, 30
расширения, 247
расширенные регулярные выражения
elvis, редактор, 366
vile, редактор, 398
Vim, редактор, 198
расширенные теги, формат файла, 351, 368
регистр, смена, 42, 100
черная дыра, 176
регулярные выражения, 94
nvi, редактор, 348
Vim, редактор, 198
метасимволы
в поиске по шаблону, 95
в строках замены, 98
трюки при заменах, 101
примеры использования шаблонов, 102
редактирование
ex, команды в командной строке, 160
elvis, редактор, 366
nvi, редактор, 349
vile, модель редактирования, 411
восстановление буфера, 70
в редакторе ex, 78
использование нескольких окон, 151
исходного кода
использование тегов, 147, 154
контроль за отступами, 144
совпадение скобок, 146
команды, 432
настройка среды редактирования, 117
нескольких файлов, 87
режим «только чтение», 69
список файлов, 130
улучшенное в клонах vi, 349, 366, 400
режим, 28
вставки, 22, 28, 426
gvim, использование мыши, 253
индикаторы режима, 164
отображения клавиш для, 132
сокращения слов, 124
выделения (gvim), 254
замещения, 42
контуров, 273
отображения, elvis, 372, 376
отступов, 281
просмотра, 28
строкового редактирования, 27
только чтение, 69
резервные копии, 328
ручное сворачивание, 276

## С

свертка, 273
сеансы (Vim), 202
сигнальный режим, 28
символ переноса строки, 35, 161
символы
замена (изменение) поодиночке, 41
метка при помощи ` (vile), 406
перемещение по символам, 34
перестановка, 46
поиск в строках, 63
удаление, 33
смена регистра, 100
синтаксические расширения Vim, 174
синтаксический метод сворачивания, 283
системный буфер, xvile и, 395
системный сбой, восстановление после, 70
скрипты
ex, 137

Vim, 175, 225
скрытые буферы, 218
слова, 53
замена (изменение), 40
отмена удаления по словам, 45
перемещение по словам, 36
перестановка слов, 47
поиск общего класса, 103
поменять местами, 126, 127, 129
поместить код жирного шрифта вокруг, 130
удаление скобок вокруг (пример), 131
смена позиции экрана, 56
совпадение скобок, 146
сокращения команд, 124, 338
сокращения слов, 124
сортировка
по алфавиту текстовых блоков (пример), 141
сохранение команд, 124
сохранение правок, 84, 434
добавление к сохраненным файлам, 86
сохранение буфера, 70
сохранение фрагментов файлов, 85
циклическое перемещение по списку файлов, 130
специальные буферы, 218
средняя строка, переход, 57
стандарт POSIX, 173
стеки тегов, 157, 302
elvis, редактор, 367
nvi, редактор, 350
Solaris vi, 148, 157
vile, редактор, 401
строки
замена (изменение), 39
копирование, 47
метка при помощи ' (vile), 406
объединение, 52
ограничение длины, 131, 161
открытие файла на указанной строке, 68
отмена удаления по строкам, 45
отображаемые на экране, опция для, 118
перемещение в строке, 35
перемещение по строкам, 58
переход на заданную, 64
печать, 76
поиск в строке, 63
преобразование регистра, 42
разместить комментарии C/C++ вокруг (пример), 131
сбор с помощью :g, 114
символов, 60
строковые редакторы, 21
строчные буквы, смена на прописные, 42, 100

## Т

теги, окна, команды, 221
текст
вставка, 50
в режиме вставки, 28
поддержка больших вставок, 49, 131, 161
замена, глобально, 91
замена (изменение), 37, 39, 62
контроль за отступами, 144
копирование (копировать-и-вставить), 37, 47
найти и удалить скобки (пример), 131
перемещение, 45
перестановка записей в базе данных (пример), 111
перемещение (удалить-и-вставить), 37
перестановка символов, 46
смена регистра, 100
удаление, 37, 64
в именованные буферы, 48, 72, 89
в нумерованные буферы, 71
восстановление удалений, 71
в редакторе ex, 81
отмена удаления, 45
посимвольное, 33
по словам, 43
текстовые блоки
визуальный режим, 164
диапазон строк, 83
перемещение, 36, 59
перемещение по шаблону, 103
сортировка, 141
сохранение фрагментов файлов, 85
текстовые редакторы, 21
редактирование в ex, 78
текущая строка (ex)
переопределение, 82
. символ для, 80
тильда (~)
в левом столбце экрана, 26
как последний текст замены, 97
команда смены регистра, 42
метасимвол, 99
:~ (подстановка с использованием шаблона последнего поиска), команда (ex), 101
тип терминала, 27
точка (.)
в имени файла, 26
метаинформация, извлечение, 176
метасимвол, 95
повторение команды, 48, 93
символ текущей строки (ex), 80
точка с запятой (;)
для диапазона строк (ex), 83
команда повторного поиска, 63

## У

удаление
восстановление удалений, 71
скобки (пример), 131
строк, 78
текста, 37, 43, 64
в именованные буферы, 48, 72, 89
в нумерованные буферы, 45, 71
в редакторе ex, 81

по символам, 33
по словам, 43
по строкам, 44
улучшенные теги, формат файла, 154
умные отступы, 284
условное выполнение, 226
устранение проблем, удаление текста, 45

## Ф

файл сеанса, elvis, 356
файлы
доступ к нескольким, 434
запуск скриптов ex для, 137
имена файлов, 26, 76
копирование в другие файлы, 86
многооконное редактирование и, 203
окружения vi, чтение из, 120
открытие, 26
в указанном месте, 68
нескольких файлов, 87
предыдущего файла, 89
режим «только чтение», 70
переименование буфера (ex), 85
проблемы при открытии, 479
расширение, 236
редактирование из других мест, 324
синтаксиса, 311
сохранение (*см.* сохранение правок), 480
текущий и альтернативный (% и #), 89
удаление, 30
циклическое перемещение по списку, 130
фигурные скобки ({})
метасимвол, 153
поиск и совпадение, 146
фрагменты файлов, сохранение, 85
функции
strftime, 227
определение, 231
функциональные клавиши, отображения, 133

## Ц

цвета
GUI-интерфейсы, 360
схемы, 225
циклы в скриптах командной строки, 138
цитаты vi, 493

## Ч

числовые аргументы команд, 35, 51
чувствительность к регистру, 52, 100
поиск по шаблону, 120
игнорирование регистра при поиске по шаблонам, 117

## Щ

щелчок мыши, elvis, 362

## Э

экранные редакторы, 21
экраны
многооконное редактирование, 151
перерисовка, 57
прокрутка, 56
смена позиции, 56
установить количество строк, отображаемых на экране, 118

# Издательство “СИМВОЛ-ПЛЮС”

*Основано в 1995 году*

## О нас

Наша специализация – книги компьютерной и деловой тематики. Наши издания – плод сотрудничества известных зарубежных и отечественных авторов, высококлассных переводчиков и компетентных научных редакторов. Среди наших деловых партнеров издательства: O’Reilly, Pearson Education, NewRiders, Addison Wesley, Wiley, McGraw-Hill, No Starch Press, Packt, Dorset House, Apress и другие.

O’REILLY®

Apress®

## Где купить

Наши книги вы можете купить во всех крупных книжных магазинах России, Украины, Белоруссии и других стран СНГ. Однако по минимальным ценам и оптом они продаются:

Санкт-Петербург:
*главный офис издательства* –
В.О. 16 линия, д. 7 (м. Василеостровская),
тел. (812) 380-5007

Москва:
*московский филиал издательства* –
ул. 2-я Магистральная, д. 14В
(м. Полежаевская/Беговая),
тел. (495) 638-5305

## Заказ книг

через Интернет http://www.symbol.ru

*Бесплатный каталог книг высылается по запросу.*

## Приглашаем к сотрудничеству

www.symbol.ru

Мы приглашаем к сотрудничеству умных и талантливых авторов, переводчиков и редакторов. За более подробной информацией обращайтесь, пожалуйста, на сайт издательства www.symbol.ru.

Также на нашем сайте вы можете высказать свое мнение и замечания о наших книгах. Ждем ваших писем!