Содержание

Благодарности

Спасибо Браму Моленару (Bram Moolenaar) за создание редактора Vim и всем, кто участвовал в его разработке. Это – образец программного обеспечения на все времена, и я с нетерпением жду появления новых его версий.

Спасибо всем сотрудникам издательства Pragmatic Bookshelf, работавшим со мной над этой книгой. Особое спасибо Кею Кепплеру (Kay Keppler), моему научному редактору, за то, что помог мне стать писателем и написать эту книгу, невзирая на сложности и мои вспышки раздражения. Спасибо также Катарине Дворак (Katharine Dvorak), моему литературному редактору, за работу над этим изданием. Я также хочу поблагодарить Дэвида Келли (David Kelly) за удовлетворение всех моих необычных пожеланий по форматированию текста.

Изначально книга «Практическое использование Vim» задумывалась совсем не как сборник рецептов, но Сюзанна Фалзер (Susannah Pfalzer) сказала, что будет лучше придерживаться именно этого формата. Было довольно сложно переписать такой объем информации, но мне удалось сделать это с первого раза, чем я был очень доволен. Сюзанна знает, как будет лучше, и я благодарен ей за ее видение.

Спасибо Дэйву Томасу (Dave Thomas) и Энди Ханту (Andy Hunt) за создание издательства Pragmatic Bookshelf. Я рад, что представлен именно этим издательством, и для меня большая честь, что моя книга находится в их каталоге.

Книга «Практическое использование Vim» едва ли появилась бы на свет без моих технических обозревателей. Каждый из вас внес свой вклад и помог в создании книги. Спасибо вам, Адам МакКри (Adam McCrea), Алан Гарднер (Alan Gardner), Алекс Кан (Alex Kahn), Али Олвейсити (Ali Alwasity), Андерс Джанмайр (Anders Janmyr), Эндрю Дональдсон (Andrew Donaldson), Ангус Нейл (Angus Neil), Чарли Танксли (Charlie Tanksley), Чез Мартин (Ches

Martin), Дэниел Бретой (Daniel Bretoi), Дэвид Моррис (David Morris), Денис Горин (Denis Gorin), Элиза Мендес Ризенд (Ely zer Mendes Rezende), Эрик Сент Мартин (Erik St. Martin), Федерико Галасси (Federico Galassi), Феликс Гейзендорфер (Felix Geisendörfer), Флориан Вален (Florian Vallen), Грэм Матисон (Graeme Mathieson), Ханс Хассельберг (Hans Hasselberg), Хенрик Них (Henrik Nyh), Хавьер Колладо (Javier Collado), Джефф Холланд (Jeff Holland), Джош Салливан (Josh Sullivan), Джошуа Флэнаган (Joshua Flanagan), Кана Натсуно (Kana Natsuno), Кент Фрайзер (Kent Frazier), Луис Мерино (Luis Merino), Матиас Мейер (Mathias Meyer), Мэтт Соузерден (Matt Southerden), Мислав Мароник (Mislav Marohnic), Митч Гатри (Mitch Guthrie), Морган Прайор (Morgan Prior), Пол Бэрри (Paul Barry), Питер Аронофф (Peter Aronoff), Питер Рин (Peter Rihn), Филип Робертс (Philip Roberts), Роберт Эванс (Robert Evans), Райан Стенхауз (Ryan Stenhouse), Чтивен Рагнарок (Steven Ragnarök), Тибор Симик (Tibor Simic), Тим Чейз (Tim Chase), Тип Поуп (Tim Pope), Тим Тирелл (Tim Tyrrell) и Тобиас Сайлер (Tobias Sailer).

Несмотря на всемерную помощью моих научных редакторов, в тексте могли остаться незамеченными некоторые ошибки. Я буду благодарен всем, кто сообщит мне об ошибках в книге, поможет их найти и исправить.

Документация к редактору Vim – потрясающий источник знаний, и я часто буду ссылаться на нее на протяжении всей книги. Я хотел бы поблагодарить Карло Тюбнера (Carlo Teubner) за публикацию документации к Vim на сайте vimhelp.appspot.com и за постоянное ее обновление.

Некоторые советы в первом издании оказались неудачными, но я все равно оставил их, потому что считал важными. В этом, пересмотренном издании я смог переписать их. Спасибо Кристиану Брабандту (Christian Brabandt) за реализацию качественно новой команды `gn`, благодаря которой я смог поправить совет 84 «Выполнение операций над полным совпадением». Спасибо Йегаппану Лакшманану (Yegappan Lakshmanan) за реализацию команды `cfdo` (и родственных ей), благодаря которой я смог поправить совет 97 «Поиск и замена в нескольких файлах». Я также хочу выразить благодарность Дэвиду Баргину (David Bürgin) за редакцию 7.3.850, что исправило мою «любимую» ошибку с командой `vimgrep`.

Я также хотел бы выразить благодарность всему сообществу пользователей Vim за распространение их знаний по всему Интернету. Многие рецепты, что приводятся в этой книге, были найдены

по тегу «Vim» на сайте StackOverflow и в списке рассылки vim_use.

Расширение rails.vim Тима Поупа (Tim Pope) оказало существенное влияние на серьезность моего отношения к Vim. Многие другие его расширения также стали обязательной частью Vim для меня. Немалую помощь в понимании Vim мне оказали также расширения, написанные Каном Натсуно (Kana Natsuno), текстовые объекты которых я считаю лучшим расширением базовой функциональности. Спасибо вам обоим, что помогли мне увидеть дополнительные преимущества.

Спасибо Джо Рознеру (Joe Rozner) за предоставленный исходный код, который я использовал для демонстрации команды `:make`. Спасибо Олегу Ефимову (Oleg Efimov) за его мгновенные ответы по проблемам nodelint. Спасибо Бену Кормаку (Ben Cormack) за иллюстрации.

В январе 2012-го мы приехали в Берлин, где сообщество технических специалистов вдохновило меня на создание данной книги. Я благодарен Грегору Шмидту (Gregor Schmidt) за основание Берлинской группы пользователей Vim и Яну Шульцу-Хофену (Jan Schulz-Hofen) за предоставленное место для встречи. Общение с пользователями Vim по-настоящему помогло мне привести свои мысли в порядок, поэтому я благодарен всем, кто посещал наши встречи в Берлине. Спасибо вам, Дэниел и Нина Холле (Daniel и Nina Holle), за то, что предоставили нам свой дом. Это замечательное место, где можно жить и активно работать.

В марте 2011-го мне потребовалась хирургическая операция на кишечнике. К сожалению, в этот момент я оказался далеко от дома, но, к счастью, моя жена была со мной. Ханна (Hannah) признала, что в южно-синайской больнице мне был обеспечен превосходный уход. Я хочу поблагодарить всех сотрудников больницы за помощь и доктора Шавкета Гергеса (Shawket Gerges) за удачно проведенную операцию.

Когда моя мама узнала, что мне нужна операция, она бросила все и ближайшим рейсом вылетела в Египет, несмотря на начавшуюся там революцию. Это был смелый поступок! Мне трудно представить, как мы с Ханной справились бы с трудностями без поддержки моей мамы и ее жизненного опыта. Я безмерно счастлив, что в моей жизни есть две такие замечательные женщины.

Предисловие к первому изданию

Существует расхожее мнение, что редактор Vim сложен в освоении. Я думаю, большинство пользователей Vim не согласятся с этим. Конечно, на первом этапе будут возникать некоторые затруднения, но как только вы пройдете обучение с помощью `vimtutor` и познакомитесь с устройством конфигурационного файла *vimrc*, вы получите почти все необходимые знания, обладая которыми, уже можно хоть как-то работать.

А что дальше? Интернет отвечает на этот вопрос: «читайте советы» – рецепты решения конкретных проблем. Вы можете искать конкретные рецепты, когда чувствуете, что текущее решение не является оптимальным, или наоборот – активно читать сборники популярных рецептов. Такой подход дает свои плоды – именно так я изучал работу с Vim, – но он очень медленный. Разумеется, полезно знать, что нажатие `*` запускает поиск слова под курсором, но едва ли это знание поможет вам думать на уровне знатока Vim.

Когда книга «Практическое использование Vim» попала в мои руки, я испытывал некоторый скепсис в отношении ее. Как пара сотен рецептов может помочь достичь того же уровня, для чего мне понадобилось изучить несколько тысяч рецептов? Однако, прочитав несколько страниц, я понял, что мое понятие «рецепт» было слишком узким. Вопреки моим ожиданиям рецепты в книге «Практическое использование Vim» не следуют шаблону проблема/решение, а представляют собой уроки, учащие думать категориями опытного пользователя Vim. В некотором смысле они больше похожи на притчи, чем на рецепты. Первые несколько рецептов учат широко использовать команду `.` – основной инструмент любого опытного пользователя Vim, для обнаружения которой без посторонней помощи мне потребовались годы.

Именно этим меня порадовала книга «Практическое использование Vim». Теперь, когда начинающие пользователи спрашивают меня: «Что дальше?», – я знаю, что им ответить. В конце концов, книга «Практическое использование Vim» даже мне преподнесла несколько приятных сюрпризов.

Тим Поуп (Tim Pope),
член основной команды разработчиков Vim,
апрель 2012 года

Это надо знать

Книга «Практическое использование Vim» предназначена для программистов, желающих повысить свой уровень. Возможно, вам доводилось слышать, что в руках мастера Vim кромсает текст со скоростью мысли. Прочитав данную книгу, вы сделаете еще один шаг к овладению этим мастерством.

Книга «Практическое использование Vim» – это самый короткий путь к овладению Vim. Она не требует наличия опыта использования Vim, но для начинающих пользователей совсем нелишним будет получить предварительные знания, воспользовавшись интерактивной обучающей программой `vimtutor`, распространяемой вместе с Vim[1]. «Практическое использование Vim» основана на этих базовых сведениях и демонстрирует приемы идиоматического использования.

Редактор Vim имеет огромное количество настроек. Однако настройки сильно зависят от личных предпочтений, поэтому я старался избегать каких-либо рекомендаций о том, что должно быть или чего не должно быть в файле *vimrc*. Вместо этого я расскажу об основных функциональных возможностях редактора, поддерживаемых им всегда, независимо от того, работаете ли вы с Vim удаленно, через SSH, или используете локальную версию GVim с массой дополнительных расширений. Овладев основами Vim, вы получите мощный и переносимый инструмент редактирования текста.

Организация книги

Книга «Практическое использование Vim» – это сборник рецептов. Ее не обязательно читать по порядку, от начала до конца. (Я изначально предполагал такую возможность! В начале следующей главы я сразу посоветую вам пропустить ее и немедленно перейти к действию.) Каждая глава – это сборник рецептов, объединенных

[1] http://vimdoc.sourceforge.net/htmldoc/usr_01.html#vimtutor

общей темой, а каждый рецепт демонстрирует применение какой-то одной особенности. Некоторые рецепты не связаны с другими. Иные требуют знания сведений, которые приводятся где-то в другом месте в этой книге. Такие рецепты включают ссылки на необходимые сведения, поэтому вы легко сможете найти их.

В целом книга «Практическое использование Vim» не следует шаблону «от простого к сложному», но каждая глава использует именно такой порядок изложения. Малоопытные читатели могут сначала просто пробежаться по книге и прочитать лишь первые рецепты в каждой главе. Более опытные пользователи могут сосредоточиться на более сложных рецептах или выбирать разделы для чтения по своему усмотрению.

Примечания к примерам

Любую задачу в Vim можно решить несколькими способами. Например, в главе 1 «Путь Vim» собраны задачи, на которых иллюстрируется применение команды «точка», но каждую из этих задач можно также решить с помощью команды `:substitute`.

Время от времени при знакомстве с моими решениями у вас может возникнуть мысль: «Разве *такое-то и такое-то* решение не будет проще?» И вы будете правы! Мои решения иллюстрируют конкретные приемы. Не заостряйте внимания на их простоте, а попробуйте отыскать сходство с задачами, которые вам приходится решать ежедневно. Именно в этом случае описываемые приемы помогут вам сэкономить свое время.

От переводчика: *в процессе работы над книгой я опробовал приводимые здесь примеры в версиях Vim/gVim для Linux и MS Windows и обнаружил одно неприятное обстоятельство: в версиях для ОС Windows некоторые комбинации клавиш действуют не так, как описывается в этой книге (а иногда вообще не действуют). Поэтому будьте внимательны, если предполагаете пользоваться редактором Vim/gVim в Windows.*

Научитесь печатать вслепую, а затем изучайте Vim

Если вы постоянно будете смотреть на клавиатуру, вы нескоро начнете получать выгоды от знания Vim. Владение слепой печатью является обязательным условием.

Свою родословную Vim ведет от классических текстовых редакторов ОС Unix, `vi` и `ed` (см. врезку «О происхождении Vim (и семейства)» в главе 5). Они были созданы задолго до появления мыши и интерфейсов типа «укажи и щелкни». В редакторе Vim все операции можно выполнить с клавиатуры. Для владеющих слепой печатью это означает, что Vim *ускоряет* работу.

Прочитайте всеми забытое руководство

В книге «Практическое использование Vim» я буду показывать примеры, а не описывать их. Это непростая задача, если использовать только письменное слово. Чтобы показать шаги, выполняемые в ходе сеанса редактирования, я буду использовать простую нотацию, иллюстрирующую нажатия клавиш и содержимое буфера Vim.

Если вы горите желанием немедленно приступить к действию, просто пропустите этот раздел. Он описывает соглашения, используемые в книге «Практическое использование Vim», многие из которых и без того понятны. Возможно, в какой-то момент вы столкнетесь с непонятным вам символом, тогда вы сможете вернуться к этому разделу и поискать нужное вам описание.

Ознакомьтесь со встроенной документацией Vim

Самый простой способ ознакомиться с документацией Vim – потратить некоторое время на ее чтение. Чтобы помочь вам в этом, я включил в текст книги «гиперссылки» на разделы документации. Например, вот «гиперссылка» на учебник Vim: `:h vimtutor`  http://vimdoc.sourceforge.net/htmldoc/usr_01.html#vimtutor.

Ярлык несет двойную функцию. Во-первых, он служит визуальным признаком полезной справочной информации. Во-вторых, если вы читаете электронную версию книги на устройстве, подключенном к Интернету, вы можете щелкнуть на ярлыке и открыть соответствующий раздел электронной документации Vim. В этом смысле ярлык является настоящей гиперссылкой.

Как быть, если вы читаете печатное издание книги? Не волнуйтесь. Если у вас имеется редактор Vim под рукой, просто введите команду, что приводится перед ярлыком.

Например, введите `:h vimtutor` (`:h` – это сокращенная версия команды `:help`). Считайте строку `vimtutor` уникальным адресом в документации, своеобразной строкой URL. В этом смысле команду получения справки можно считать гиперссылкой во встроенной документации Vim.

Нотация представления Vim на странице

Модальный интерфейс Vim отличается от интерфейса большинства других текстовых редакторов. Если провести аналогию с музыкальными инструментами, клавиатуру компьютера можно сравнить с клавиатурой фортепиано. Пианист может проигрывать мелодию, нажимая клавиши по одной или целыми группами, извлекая аккорды. В большинстве текстовых редакторов клавиатурные сокращения соответствуют нажатию какой-то клавиши при одной или более удерживаемых клавишах-модификаторах, таких как **Control** и **Command**. Такие клавиатурные сокращения являются эквивалентом аккордов.

Некоторые команды Vim также запускаются проигрыванием аккордов, но в режиме Normal (командный режим, или режим по умолчанию) команды вводятся как последовательные нажатия клавиш. Это компьютерный эквивалент проигрыванию мелодии нажатием клавиш фортепиано по одной.

`Ctrl-S` – это типичное соглашение, используемое для обозначения команд, вызываемых аккордами из комбинаций клавиш. В данном случае эта последовательность означает: «Нажать клавиши **Control** и **S** одновременно». Но подобное соглашение плохо подходит для описания команд Vim. В этом разделе мы увидим нотацию, используемую по всей книге «Практическое использование Vim» для иллюстрации приемов работы с редактором.

Проигрывание мелодий

В командном режиме команды вводятся последовательным нажатием одной или более клавиш. Выглядят эти команды следующим образом:

Нотация	Значение
`x`	Нажать одну клавишу `x`
`dw`	Нажать сначала `d`, потом `w`
`dap`	Нажать сначала `d`, потом `a`, потом `p`

Большинство таких последовательностей состоят из двух или трех нажатий на клавиши, но есть и более длинные. Расшифровка значения команд в командном режиме может оказаться непростым делом, но, немного попрактиковавшись, вы легко будете справляться с этим.

Аккорды

Обозначения, такие как `<C-p>`, не означают: «Нажать `<`, затем `C`, затем `-` и т. д.». Запись `<C-p>` эквивалентна записи `Ctrl-p`, то есть: «Нажать клавиши `<Ctrl>` и `p` одновременно».

Я выбрал такую форму записи вполне обоснованно. Она используется во встроенной документации Vim (`:h key-notation` ⓘ http://vimdoc.sourceforge.net/htmldoc/intro.html#key-notation), и мы также будем использовать ее для определения комбинаций клавиш. Некоторые команды Vim запускаются как комбинации аккордов и последовательных нажатий на клавиши, и такая форма записи отлично описывает их, например:

Нотация	Значение
`<C-n>`	Нажать одновременно `<Ctrl>` и `n`
`g<C-]>`	Нажать `g`, затем одновременно нажать `<Ctrl>` и `]`
`<C-r>0`	Нажать одновременно `<Ctrl>` и `r`, затем нажать `0`
`<C-w><C-=>`	Нажать одновременно `<Ctrl>` и `w`, затем нажать одновременно `<Ctrl>` и `=`

Символы-заполнители

Многие команды Vim требуют последовательного нажатия двух или более клавиш. Некоторые команды должны сопровождаться вводом некоторой последовательности символов, тогда как другие – вводом единственного символа. Символы, которые должны вводиться после команды, я буду обозначать как последовательности, заключенные в фигурные скобки, например:

Нотация	Значение
`f{char}`	Нажать `f`, затем ввести любой символ
`` `{a-z} ``	Нажать `` ` ``, затем ввести любой символ нижнего регистра
`m{a-zA-Z}`	Нажать `m`, затем ввести любой символ нижнего или верхнего регистра
`d{motion}`	Нажать `d`, затем ввести любую команду перемещения курсора
`<C-r>{register}`	Нажать одновременно `<Ctrl>` и `r`, затем ввести адрес регистра

Обозначение специальных клавиш

Некоторые клавиши имеют специальные обозначения. Некоторые из них перечислены в таблице ниже:

Нотация	Значение
`<Esc>`	Клавиша Escape (Esc)
`<CR>`	Клавиша возврата каретки (также известна как <Enter>)
`<Ctrl>`	Клавиша Control (Ctrl)
`<Tab>`	Клавиша табуляции (Tab)
`<Shift>`	Клавиша Shift
`<S-Tab>`	Одновременное нажатие клавиш <Shift> и <Tab>
`<Up>`	Клавиша со стрелкой «вверх»
`<Down>`	Клавиша со стрелкой «вниз»
`␣`	Пробел

Обратите внимание, что клавиша пробела обозначается как `␣`. Команда `f{char}` с этой клавишей будет записываться так: `f␣`.

Обозначение команд в разных режимах

При работе с редактором Vim довольно часто приходится переключаться между командным режимом и режимом вставки. В разных режимах одна и та же клавиша может вызывать разные действия. Нажатия клавиш в режиме **Insert** (режиме вставки) я буду обозначать иначе, чтобы их проще было отличать от нажатий клавиш в командном режиме.

Взгляните на такой пример: `cwreplacement<Esc>`. В командном режиме команда `cw` удалит текст до конца текущего слова и переключит редактор в режим вставки. Затем мы вводим слово «replacement» в режиме вставки и нажимаем клавишу `<Esc>`, чтобы переключиться обратно в командный режим.

Форма представления нажатий клавиш в командном режиме также будет использоваться для обозначения нажатий клавиш в режиме **Visual** (визуальный режим), а форма представления нажатий клавиш в режиме вставки будет использоваться для обозначения нажатий клавиш в режимах **Command-Line** (командной строки) и **Replace** (замены). Какой режим является текущим, будет очевидно из контекста.

Режим командной строки

В некоторых рецептах мы будем выполнять команды из командной строки – либо команды командной оболочки, либо внутренние

команды Vim. Вот как выглядит запуск команды `grep` в командной оболочке:

```
⇛ $ grep -n Waldo *
```

А так выглядит запуск встроенной в Vim команды `:grep`:

```
⇛ :grep Waldo *
```

В примерах в этой книге символ `$` будет указывать, что команда выполняется внешней командной оболочкой, а символ `:` будет указывать, что команда выполняется во внутреннем режиме командной строки. Иногда мы будем видеть и другие приглашения к вводу, включая следующие:

Приглашение	Значение
$	Введенная команда будет выполнена внешней командной оболочкой
:	Используется режим командной строки редактора Ex
/	Используется режим командной строки для поиска вперед
?	Используется режим командной строки для поиска назад
=	Используется режим командной строки для вычисления выражения на языке сценариев Vim

Всякий раз, когда в тексте будет встречаться команда редактора Ex, такая как `:write`, будет предполагаться, что она завершается нажатием клавиши `<CR>`, иначе команда просто не будет выполнена, то есть нажатие клавиши `<CR>` будет подразумеваться неявно.

Напротив, команда поиска в Vim отыскивает первое совпадение до нажатия клавиши `<CR>` (см. рецепт 82 в главе 13). Когда в тексте будет встречаться команда поиска, такая как `/pattern<CR>`, нажатие клавиши `<CR>` будет указываться явно. Если `<CR>` будет отсутствовать в команде поиска, знайте, что это сделано преднамеренно и означает, что вы не должны нажимать клавишу **Enter**.

Отображение позиции курсора в буфере

При просмотре содержимого буфера иногда полезно иметь возможность увидеть, где находится курсор. В следующем примере показано, что курсор находится в позиции первой буквы в слове «One»:

```
One two three
```

Когда мы в несколько этапов вносим некоторое изменение, содержимое буфера проходит через последовательность промежуточных состояний. Для отображения этого процесса я буду использовать таблицу, в левом столбце которой будут показаны выполняемые команды, а в правом – содержимое буфера. Например:

Нажатия клавиш	Содержимое буфера
{start}	One two three
dw	two three

Во второй строке выполняется команда dw, удаляющая слово под курсором. В этой же строке показано, как изменилось состояние буфера сразу после выполнения команды.

Подсветка совпадений, найденных при поиске

При демонстрации команды поиска полезно иметь возможность выделить все совпадения, найденные в буфере. В следующем примере выполняется поиск строки «the» и обнаруживаются четыре совпадения, выделенные цветом:

Нажатия клавиш	Содержимое буфера
{start}	the problem with these new recruits is that they don't keep their boots clean.
/the<CR>	the problem with these new recruits is that they don't keep their boots clean.

Загляните в рецепт 81 в главе 13, где рассказывается, как включить в Vim подсветку совпадений, найденных при поиске.

Выделение текста в визуальном режиме

В визуальном режиме имеется возможность выделить текст в буфере и затем выполнить с ним какие-либо операции. В следующем примере для выделения содержимого тега <a> используется текстовый объект it:

Нажатия клавиш	Содержимое буфера
{start}	Practical Vim
vit	Practical Vim

Обратите внимание, что выделение в визуальном режиме выглядит так же, как подсветка совпадений в режиме поиска. Когда в книге будет приводиться такое оформление, из контекста должно быть понятно, представляет оно выделение в визуальном режиме или подсветку совпадений в режиме поиска.

Загружаемые примеры

Примеры в книге обычно будут начинаться с демонстрации содержимого файла *до* его изменения. Заголовки листингов будут также включать путь к файлу:

macros/incremental.txt
http://media.pragprog.com/titles/dnvim/code/macros/incremental.txt

```
partridge in a pear tree
turtle doves
French hens
calling birds
golden rings
```

Каждый раз, когда вы увидите такое имя файла со строкой пути к нему, это будет означать, что данный пример можно загрузить. В каждом таком случае я рекомендую открыть файл в Vim и попытаться выполнить упражнение самостоятельно. Это самый лучший способ изучения!

Прежде чем продолжить, загрузите все примеры и исходные коды с сайта издательства Pragmatic Bookshelf[1]. Если вы читаете электронную версию книги на устройстве, подключенном к Интернету, вы можете загрузить любой файл отдельно, просто щелкнув на имени файла. Попробуйте сделать это в примере выше.

Используйте настройки Vim по умолчанию

Редактор Vim может настраиваться в очень широких пределах. Если какие-то настройки по умолчанию не устраивают вас,

[1] http://pragprog.com/titles/dnvim/source_code

измените их. В этом нет ничего плохого, но это может вызвать путаницу, если вы собираетесь следовать за примерами в данной книге, используя версию Vim, настроенную по своему вкусу. Вы можете неожиданно обнаружить, что что-то работает совсем не так, как описывается в книге. Если вы подозреваете, что причиной являются произведенные вами настройки, выполните следующий простой тест. Попробуйте запустить Vim со следующими параметрами:

```
⇛ $ vim -u NONE -N
```

Флаг `-u NONE` сообщает редактору Vim, что он не должен загружать настройки из файла *vimrc*. То есть настройки не будут выполнены, а расширения будут отключены. Когда Vim запускается таким способом, он возвращается к режиму совместимости с `vi`, в котором отключены многие полезные особенности. Флаг `-N` предотвращает это, устанавливая «несовместимый» режим.

Для большинства примеров в этой книге трюк с командой `vim -u NONE -N` гарантирует, что примеры будут выполняться в точности как описывается, кроме пары исключений. Некоторые встроенные особенности Vim реализованы в виде сценариев Vim, то есть они будут работать, только когда включена поддержка расширений. Следующий файл содержит минимальный набор настроек, необходимых для включения встроенных расширений Vim:

essential.vim
http://media.pragprog.com/titles/dnvim/code/essential.vim

```
set nocompatible
filetype plugin on
```

Чтобы использовать этот файл с настройками вместо стандартного *vimrc*, редактор Vim следует запустить командой:

```
⇛ $ vim -u code/essential.vim
```

При этом не забудьте исправить путь *code/essential.vim*, если необходимо. Включив встроенные расширения, вы сможете использовать такие особенности Vim, как `netrw` (см. рецепт 44 в главе 7) и `omni-completion` (см. рецепт 119 в главе 19), а также многие другие. Под настройками по умолчанию я подразумеваю включенную поддержку расширений и выключенный режим совместимости с `vi`.

Обязательно заглядывайте в раздел «Подготовка» в начале каждого рецепта. Чтобы воспроизвести пример, который приводится в рецепте, необходимо настроить Vim соответственно. Запуская Vim с настройками по умолчанию и применяя дополнительные настройки «на лету», вы сможете воспроизвести шаги, описываемые в рецепте, без каких-либо проблем.

Если вы по-прежнему будете испытывать проблемы, загляните в раздел «О версиях Vim» ниже.

О роли сценариев Vim

Сценарии Vim дают возможность добавлять в редактор новые функциональные возможности или изменять имеющиеся. Vim поддерживает полноценный язык сценариев, достойный отдельной книги, но «Практическое использование Vim» – не та книга.

Однако мы не будем сторониться этой темы полностью. Поддержка сценариев Vim всегда к нашим услугам, и мы рассмотрим несколько примеров использования их в повседневной работе в рецептах: 16 (глава 3), 71 (глава 11), 95 и 96 (глава 14).

Эта книга описывает приемы использования базовой функциональности Vim. Иными словами, мы не будем касаться сторонних расширений. Однако в рецепте 87 (глава 13) я сделаю исключение. Здесь я буду рекомендовать расширение `visual-star.vim`, добавляющее новые особенности, которые я считаю совершенно необходимыми. Его применение потребует написать совсем немного кода – менее десятка строк. Этот пример продемонстрирует, как легко расширяется функциональность Vim. Реализация `visual-star.vim` будет представлена без дополнительных пояснений. Взглянув на содержимое этих файлов, вы получите представление о том, как выглядят сценарии Vim и чего можно добиться с их помощью. Если эта демонстрация разбудит в вас интерес, тем лучше.

О версиях Vim

Все примеры в книге «Практическое использование Vim» были протестированы с последней версией Vim, каковой на момент написания этих строк была версия 7.3. Однако большинство примеров с успехом будут работать в любой версии 7.x, а многие особенности, обсуждаемые здесь, также доступны в версиях 6.x.

Некоторые особенности Vim могут отключаться на этапе компиляции. Например, при подготовке к сборке можно указать параметр `--with-features=tiny`, который отключит все особенности, кроме самых основных (существуют также наборы особенностей, помеченные как `small`, `normal`, `big` и `huge`). Посмотреть список особенностей можно с помощью команды `:h +feature-list` ℹ http://vimdoc.sourceforge.net/htmldoc/various.html#+feature-list.

Если вы столкнетесь с отсутствием какой-либо особенности, обсуждаемой в книге, возможно, вы пользуетесь минимальной сборкой Vim. Проверьте доступность особенности с помощью команды `:version`:

```
⇛ :version
  VIM - Vi IMproved 7.3 (2010 Aug 15, compiled Sep 18 2011 16:00:17)
  Huge version with MacVim GUI. Features included (+) or not (-):
  +arabic +autocmd +balloon_eval +browse +builtin_terms +byte_offset
  +cindent +clientserver +clipboard +cmdline_compl +cmdline_hist
  +cmdline_info +comments
  ...
```

В современных компьютерах нет причин использовать набор особенностей Vim меньше, чем `huge`!.

Vim в терминале или Vim с графическим интерфейсом? Выбирать вам!

Традиционно Vim используется в терминале, без графического интерфейса, можно даже сказать, что Vim имеет только текстовый интерфейс. Если большую часть дня вы проводите в командной строке, для вас это будет выглядеть вполне естественно.

Если вы привыкли пользоваться текстовыми редакторами с графическим интерфейсом, тогда роль моста в мир Vim для вас может сыграть GVim (или MacVim в OS X) (см. `:h gui` ℹ http://vimdoc.sourceforge.net/htmldoc/gui.html#gui). GVim поддерживает большое количество шрифтов и больше цветов для подсветки синтаксиса. Кроме того, он позволяет пользоваться мышью и следует некоторым соглашениям, принятым в операционной системе. Например, в MacVim поддерживаются: доступ к системному буферу обмена, с помощью комбинаций `Cmd-X` и `Cmd-V`, возможность сохранения документа комбинацией клавиш `Cmd-S` и закрытия окна комбинацией

клавиш Cmd-W. Вы можете использовать их, если вам это удобно, но знайте, что всегда есть более удачный путь.

Для нашего обсуждения совершенно не важно, какой версией вы будете пользоваться, редактором Vim в терминале или редактором GVim с графическим интерфейсом. Все наше внимание мы сконцентрируем на базовых командах, которые доступны в любом из этих редакторов. Мы узнаем, как выполнять свою работу, следуя *путем Vim*.

Глава 1. Путь Vim

Выполняемая нами работа содержит множество повторений. Будь то внесение одних и тех же небольших изменений в нескольких местах или перемещение между похожими фрагментами документа, мы повторно выполняем одни и те же действия. Все, что сможет упростить выполнение повторяющихся операций, поможет нам сэкономить наше время.

Редактор Vim оптимизирует такие повторяющиеся действия. Его эффективность во многом обусловлена возможностью отслеживать последовательность последних выполняемых операций. Мы всегда можем воспроизвести последние изменения нажатием одной клавиши. Это действительно очень мощная особенность, но она становится совершенно бесполезной для тех, кто не умеет фиксировать свои действия так, чтобы они выполняли полезную работу при повторном воспроизведении. Овладение данной концепцией является ключом к эффективному использованию Vim.

Команда «точка» станет нашей отправной точкой. Эта, казалось бы, простая команда – самый мощный инструмент в нашем арсенале и первый шаг на пути к овладению редактором Vim. Мы рассмотрим несколько простых задач редактирования, которые с помощью команды «точка» могут быть решены в мгновение ока. Хотя все задачи отличаются друг от друга, их решения во многом сходны. В каждом случае мы определим единую формулу редактирования, для выполнения которой требуется нажатие всего одной клавиши.

Рецепт 1. Встречайте: команда «точка»

Команда «точка» позволяет повторять последнее изменение. Это самая мощная и гибкая команда в редакторе Vim.

Документация к редактору Vim просто отмечает, что команда «точка»: «повторяет последнее изменение» (см. `:h .` ❶ http://

vimdoc.sourceforge.net/htmldoc/repeat.html#). Вроде бы ничего особенного, но за этим простым определением скрывается мощная основа, делающая модель модального редактирования в Vim такой эффективной. Для начала выясним, что это за «последнее изменение».

Чтобы осознать всю мощь команды «точка», нужно понимать, что под «последним изменением» может скрываться все, что угодно. Изменением могут быть действия над отдельными символами, строками или даже над целым файлом.

Для демонстрации воспользуемся следующим фрагментом текста:

the_vim_way/0_mechanics.txt
http://media.pragprog.com/titles/dnvim/code/the_vim_way/0_mechanics.txt

```
Line one
Line two
Line three
Line four
```

Команда `x` удаляет символ под курсором. Когда команда «точка» используется в этом контексте, под «последним изменением» Vim будет понимать удаление символа под курсором:

Нажатия клавиш	Содержимое буфера
{start}	Line one Line two Line three Line four
x	ine one Line two Line three Line four
.	ne one Line two Line three Line four
..	one Line two Line three Line four

Вернуть файл в исходное состояние можно, нажав клавишу `u` несколько раз, чтобы отменить изменения.

Команда `dd` тоже выполняет удаление, но она удаляет текущую строку целиком. Если использовать команду «точка» после команды

dd, тогда под «последним изменением» Vim будет понимать удаление текущей строки:

Нажатия клавиш	Содержимое буфера
{start}	Line one Line two Line three Line four
dd	ine one Line two Line three Line four
.	Line three Line four

Наконец, команда >G увеличивает отступ, начиная с текущей строки, до конца файла. После выполнения этой команды под «последним изменением» Vim будет понимать увеличение отступа от текущей строки до конца файла. Этот пример мы начали с того, что поместили курсор в начало второй строки, чтобы подчеркнуть отличия.

Нажатия клавиш	Содержимое буфера
{start}	Line one Line two Line three Line four

>G	Line one
	  Line two
	  Line three
	  Line four

j	Line one
	  Line two
	  Line three
	  Line four

.	Line one
	  Line two
	  Line three
	  Line four

j.	Line one
	  Line two
	  Line three
	  Line four

Все команды – x, dd и > – выполняются в командном режиме, но изменения также создаются каждый раз при входе в режим вставки.

От момента входа в режим вставки (например, нажатием клавиши `i`) до выхода в командный режим (нажатием `<Esc>`) Vim записывает все нажатия клавиш. После создания такого изменения команда «точка» будет повторять эти нажатия (но прочитайте предупреждение во врезке «Перемещение по тексту в режиме вставки закрывает последнее изменение» в главе 2).

Команда «точка» — микромакроопределение

Ниже, в главе 11 «Макросы», будет показано, как в редакторе Vim организовать запись последовательности из произвольного количества нажатий клавиш для повторного воспроизведения ее в будущем. Эта возможность позволяет сохранять повторяющиеся операции и воспроизводить их нажатием одной клавиши. Команду «точка» можно считать миниатюрным, или «микро-» (если хотите), макроопределением.

На протяжении этой главы будет показано несколько вариантов применения команды «точка». Кроме того, пара приемов работы с командой «точка» будет показана в рецептах 9 (глава 2) и 43 (глава 4).

Рецепт 2. Не повторяйся

Для такой типичной операции, как добавление точки с запятой в конец строки, Vim предоставляет отдельную команду, объединяющую два шага в один.

Допустим, что у нас имеется следующий фрагмент кода на JavaScript:

the_vim_way/2_foo_bar.js
http://media.pragprog.com/titles/dnvim/code/the_vim_way/2_foo_bar.js

```
var foo = 1
var bar = 'a'
var foobar = foo + bar
```

Нам требуется добавить точку с запятой в конец каждой строки. Для этого нужно переместить курсор в конец текущей строки и затем перейти в режим вставки, чтобы выполнить изменение. Команда `$` выполнит такое перемещение, после чего мы сможем нажать последовательность `a;<Esc>`, чтобы выполнить изменение.

Чтобы выполнить задание полностью, нам могло бы потребоваться повторить ту же самую последовательность нажатий с остальными двумя строками, но это дело слишком уж нехитрое. Команда «точка» повторяет последнее изменение, поэтому мы можем просто выполнить `j$.` дважды. Одно нажатие (`.`) экономит три (`a;<Esc>`). Экономия получается не очень большая, но эффективность приема будет расти с ростом количества повторений.

А теперь рассмотрим поближе этот шаблон: `j$.`. Команда `j` перемещает курсор на одну строку вниз, а затем команда `$` перемещает его в конец строки. Нам пришлось нажать две клавиши, только чтобы вывести курсор в позицию, где можно применить команду «точка». Не кажется ли вам, что здесь есть что улучшить?

Избавляйтесь от лишних перемещений

Команда `a` переходит в режим вставки в позицию, находящуюся за позицией курсора. Однако есть команда `A`, выполняющая переход в режим вставки в позицию, находящуюся в конце текущей строки. Где бы ни находился курсор, нажатие на клавишу `A` выполнит переход в режим вставки и переместит курсор в конец строки. Иными словами, эта команда совмещает в себе последовательность команд `$a`. Во врезке «Две по цене одной» ниже мы узнаем, что в редакторе Vim имеется несколько подобных составных команд.

Ниже показано измененное решение предыдущей задачи:

Нажатия клавиш	Содержимое буфера

`{start}`	`var foo = 1`
	`var bar = 'a'`
	`var foobar = foo + bar`

`A;<Esc>`	`var foo = 1;`
	`var bar = 'a'`
	`var foobar = foo + bar`

`j`	`var foo = 1;`
	`var bar = 'a'`
	`var foobar = foo + bar`

`.`	`var foo = 1;`
	`var bar = 'a';`
	`var foobar = foo + bar`

`j.`	`var foo = 1;`
	`var bar = 'a';`
	`var foobar = foo + bar;`

Используя `A` вместо `$a`, мы расширяем команду «точка». Вместо того чтобы перемещать курсор в *конец* строки, которую требуется изменить, мы просто перемещаем курсор в эту строку (в *любую позицию* в ней!). Теперь мы можем внести изменения в последовательность строк, просто нажимая `j.` столько раз, сколько потребуется.

Одно нажатие – чтобы переместить курсор, и одно нажатие – чтобы выполнить изменение. Разве это не здорово?! Запомните этот прием, потому что мы встретимся с ним еще в нескольких примерах.

Несмотря на свою потрясающую простоту, эта формула не является универсальным решением. Представьте, что нам потребовалось добавить точку с запятой в пятьдесят строк, следующих друг за другом. В этом случае нажатие пары клавиш `j.` превращается в утомительный труд. Альтернативное решение этой задачи вы найдете в рецепте 30 в главе 5.

Две по цене одной

Мы могли бы сказать, что команда `A` состоит из двух действий (`$a`), совмещенных в одном нажатии клавиши. Однако она не является единственной командой такого рода. Многие команды Vim можно рассматривать как одноклавишные версии последовательностей из двух или более команд. В таблице ниже перечислено несколько таких команд. Сможете ли вы определить, что их объединяет?

Составная команда	Эквивалентная последовательность
`C`	`c$`
`s`	`cl`
`S`	`^C`
`I`	`^i`
`A`	`$a`
`o`	`A<CR>`
`O`	`ko`

Если вы вдруг обнаружите, что все время выполняете последовательность `ko` (или хуже того, `k$a<CR>`), остановитесь! Подумайте о том, что вы делаете. Затем вспомните, что можно было бы использовать более простую команду `O`.

Сумеете ли вы заметить другую общую черту, объединяющую эти команды? Все они выполняют переход из командного режима в режим вставки. Поразмышляйте над этим и над тем, как это может влиять на команду «точка».

Рецепт 3. Шаг назад, три вперед

Мы можем дополнить единственный символ пробелами (один слева, другой справа), используя идиоматическое для Vim решение. Сначала оно будет казаться странным, но это решение дает возможность повторения, что позволяет выполнить задание с минимумом усилий.

Представьте, что имеется такая строка кода:

the_vim_way/3_concat.js
http://media.pragprog.com/titles/dnvim/code/the_vim_way/3_concat.js

```
var foo = "method("+argument1+","+argument2+")";
```

Конкатенация строк в JavaScript никогда не выглядела особенно удобочитаемо, тем не менее мы могли бы повысить удобочитаемость, окружая каждый знак + пробелами, чтобы строка кода выглядела, как показано ниже:

```
var foo = "method(" + argument1 + "," + argument2 + ")";
```

Делайте изменения повторяемыми

Описываемая задача имеет следующее идиоматическое решение:

Нажатия клавиш	Содержимое буфера
{start}	var foo = "method("+argument1+","+argument2+")";
f+	var foo = "method("+argument1+","+argument2+")";
s␣+␣<Esc>	var foo = "method(" + argument1+","+argument2+")";
;	var foo = "method(" + argument1 + ","+argument2+")";
.	var foo = "method(" + argument1 + ","+argument2+")";
;.	var foo = "method(" + argument1 + "," + argument2+”)”;
;.	var foo = "method(" + argument1 + "," + argument2 + ")";

Команда s объединяет два шага в один: она удаляет символ под курсором и переходит в режим вставки. После удаления символа + мы вводим ␣+␣ и покидаем режим вставки.

Один шаг назад и затем три вперед. Этот странный танец на клавиатуре сначала выглядит малопонятным, но он дает нам большой выигрыш: мы можем повторять изменения с помощью команды «точка»: все, что для этого нужно, – переместить курсор к следующему символу +, после чего команда «точка» повторит этот маленький танец.