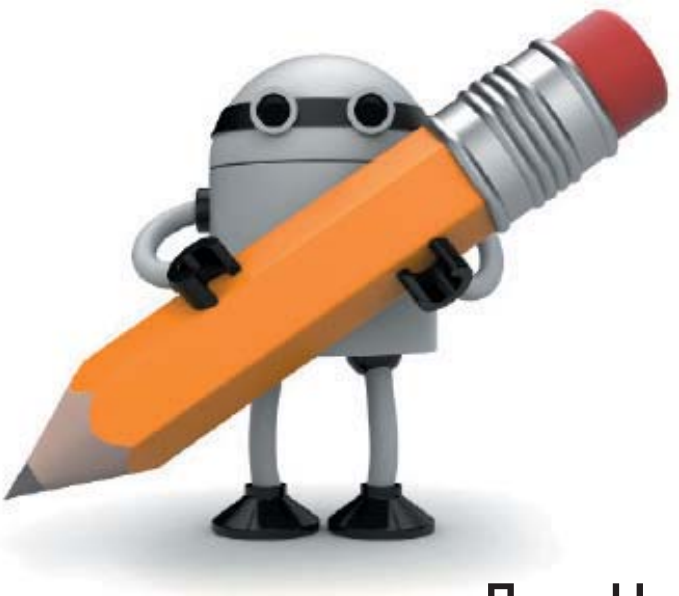
The Pragmatic Programmers

Практическое использование Vim

Редактируйте текст со скоростью мысли



Дрю Нейл

Pragmatic Bookshelf

ДМК ПРЕСС ИЗДАТЕЛЬСТВО

УДК 004.912Vim
ББК 32.973.26-018.2
Н38

Нейл Д.
Н38 Практическое использование Vim. – М.: ДМК Пресс, 2014. – 384 с.: ил.

ISBN 978-5-94074-972-1

Vim – быстрый и эффективный текстовый редактор, способный повысить скорость и эффективность разработки. С помощью более чем 100 рецептов вы быстро освоите основные возможности Vim и сможете заняться решением своих самых необычных задач, связанных с созданием и правкой текста.

В данной книге вы найдете новые и эффективные приемы работы с редактором, независимо от того, являетесь ли вы начинающим или опытным пользователем Vim.

УДК 004.912Vim
ББК 32.973.26-018.2

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

Материал, изложенный в данной книге, многократно проверен. Но, поскольку вероятность технических ошибок все равно существует, издательство не может гарантировать абсолютную точность и правильность приводимых сведений. В связи с этим издательство не несет ответственности за возможные ошибки, связанные с использованием книги.

ISBN 978-1-934356-98-2 (анг.)

ISBN 978-5-94074-972-1 (рус.)

Copyrigth © 2012 The Pragmatic Bookshelf

© Оформление, перевод, ДМК Пресс, 2014

Содержание

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 .` 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.

Делайте перемещения повторяемыми

В этом примере присутствует еще одна хитрость. Команда `f{char}` сообщает редактору Vim, что он должен найти следующее вхождение указанного символа и переместить курсор в его позицию (см. `:h f` http://vimdoc.sourceforge.net/htmldoc/motion.html#f). Поэтому, когда вводится последовательность `f+`, курсор перепрыгивает к следующему символу `+`. Подробнее о команде `f{char}` рассказывается в рецепте 49 в главе 8.

Выполнив первое изменение, мы могли бы перейти к следующему символу `+`, повторив команду `f+`, но есть более простой путь. Команда `;` повторяет последний поиск, произведенный с помощью команды `f`. Поэтому вместо команды `f+` мы можем использовать эту команду.

Теперь все вместе

Команда `;` переносит курсор в позицию следующего символа `+`, а команда `.` повторяет последнее изменение, поэтому мы можем внести оставшиеся изменения, трижды введя последовательность `;.`. Не кажется ли вам такой алгоритм действий знакомым?

Вместо того чтобы бороться с модальной моделью ввода в Vim, мы используем ее преимущества и убеждаемся, насколько она способна упростить решение поставленной задачи.

Рецепт 4. Действие, повтор, возврат

Сталкиваясь с повторяющимися задачами, мы можем выработать еще более оптимальную стратегию редактирования, делая одновременно перемещения и изменения повторяющимися. В редакторе Vim имеется все необходимое для этого. Он запоминает наши действия и сохраняет их так, чтобы мы легко могли воспользоваться ими. В этом рецепте мы познакомимся с каждым из действий, которые Vim способен повторить, и узнаем, как отменить их.

Мы уже знаем, что команда «точка» повторяет последнее ***изменение***. Поскольку существует огромное количество команд, которые могут интерпретироваться как изменения, команда «точка» оказывается чрезвычайно универсальной. Но некоторые команды могут повторяться другими средствами. Например, `@:` может повторить любую команду редактора Ex (как описывается в рецепте 31 в главе 5). Точно так же, нажав `&` (см. рецепт 92 в главе 14), можно повто-

рить последнюю команду подстановки `:substitute` (которая также является командой редактора Ex).

Если известно, как повторять действия, избегая необходимости вводить их каждый раз, мы можем добиться высочайшей эффективности. Сначала мы выполняем действие, затем повторяем его.

Но когда многого можно добиться лишь несколькими нажатиями клавиш, приходится быть внимательными. Слишком легко допустить ошибку. Многократное повторение `j.j.j.` снова и снова начинает напоминать барабанную дробь. Что, если по ошибке мы нажмем клавишу `j` два раза подряд? Или хуже того, два раза нажмем клавишу `.`?

Всякий раз, когда редактор Vim упрощает повторение действия или перемещения, он всегда предоставляет способ отступить назад, если, войдя в раж, мы зайдем слишком далеко. В случае с командой «точка» мы всегда можем нажать клавишу `u`, чтобы отменить последнее изменение. Если лишний раз нажать клавишу `;` после использования команды `f{char}`, можно пропустить искомый символ. Но мы можем вернуться назад, нажав клавишу `,`, которая повторяет последнюю команду поиска `f{char}` в обратном направлении (см. рецепт 49 в главе 8).

Всегда полезно знать, как включается обратная передача, на случай, если по ошибке мы зайдем слишком далеко. В табл. 1 перечислены команды повторения, поддерживаемые редактором Vim, и соответствующие им обратные команды. В большинстве случаев можно использовать команду отмены. Неудивительно, что самая потертая клавиша на моей клавиатуре – это клавиша `u`!

Таблица 1.1. Команды повторения и обратные им команды

Цель	Команда	Повторение	Возврат
Выполнить изменение	`{edit}`	`.`	`u`
Найти следующий символ в строке	`f{char}`/`t{char}`	`;`	`,`
Найти предыдущий символ в строке	`F{char}`/`T{char}`	`;`	`,`
Найти следующее совпадение в документе	`/pattern<CR>`	`n`	`N`
Найти предыдущее совпадение в документе	`?pattern<CR>`	`n`	`N`
Выполнить подстановку	`:s/target/replacement`	`&`	`u`
Выполнить последовательность изменений	`qx{changes}q`	`@x`	`u`

Рецепт 5. Поиск и замена вручную

Для выполнения поиска с заменой в Vim имеется команда `:substitute`, но существует альтернативный прием, когда мы изменяем первое вхождение вручную, а затем отыскиваем и заменяем каждое последующее вхождение по одному. Команда «точка» могла бы спасти нас от переутомления, но существует другая, более привлекательная команда, осуществляющая переход между совпадениями.

В следующем фрагменте текста в каждой строке встречается слово «content»:

the_vim_way/1_copy_content.txt
http://media.pragprog.com/titles/dnvim/code/the_vim_way/1_copy_content.txt

```
...We're waiting for content before the site can go live...
...If you are content with this, let's go ahead with it...
...We'll launch as soon as we have the content...
```

Допустим, что нам понадобилось заменить каждое слово «content» словом «copy». Вы можете подумать, что сделать это очень просто – достаточно лишь воспользоваться командой подстановки:

```
⇛ :%s/content/copy/g
```

Но минутку! В этом случае фраза «If you are content with this» («Если вас это устраивает») превратится во фразу «If you are 'copy' with this» («Если вас это копия»), лишенную смысла!

Мы столкнулись с проблемой, потому что слово «content» имеет два смысла. Один из них – синоним слова «copy» (копия), а другой – синоним слова «happy» (довольный). Технически мы имеем дело с омонимами (словами, разными по значению, но одинаковыми по написанию), но не это здесь главное. Главное здесь то, что вы должны следить за своими действиями.

Мы не можем слепо заменить каждое вхождение слова «content» словом «copy». Мы должны оценить каждое из найденных вхождений и ответить «да» или «нет» на вопрос о замене. Команда подстановки имеет свою область применения, и мы познакомимся с ней поближе в рецепте 89 в главе 14. А пока займемся альтернативным решением, соответствующим теме данной главы.

Будьте экономны: выполняйте поиск без ввода лишних символов

Возможно, вы уже догадались, что команда «точка» – моя любимица. На втором месте у меня стоит команда *. Она выполняет поиск слова под курсором (см. :h * http://vimdoc.sourceforge.net/htmldoc/pattern.html#star).

Мы можем выполнить поиск слова «content», напечатав его в строке приглашения к вводу:

```
⇛ /content
```

или просто поместив курсор в требуемое слово и нажав клавишу *. Взгляните, как выглядит весь процесс:

Нажатия клавиш	Содержимое буфера
{start}	...We're waiting for content before the site can go live... ...If you are content with this, let's go ahead with it... ...We'll launch as soon as we have the content...
*	...We're waiting for content before the site can go live... ...If you are content with this, let's go ahead with it... ...We'll launch as soon as we have the content...
cwcopy<Esc>	...We're waiting for content before the site can go live... ...If you are content with this, let's go ahead with it... ...We'll launch as soon as we have the copy...
n	...We're waiting for content before the site can go live... ...If you are content with this, let's go ahead with it... ...We'll launch as soon as we have the content...
.	...We're waiting for copy before the site can go live... ...If you are content with this, let's go ahead with it... ...We'll launch as soon as we have the content...

Процесс начинается с установки курсора в слово «content», после чего вызывается команда * поиска. Попробуйте сами выполнить это упражнение. В результате случится следующее: курсор перепрыгнет вперед, к следующему совпадению, и все найденные вхождения будут выделены цветом. Если вы не увидите подсвеченные совпадения, попробуйте выполнить команду :set hls и затем обратиться к рецепту 80 в главе 13 за более подробными разъяснениями.

Выполнив поиск слова «content», перейти к следующему совпадению можно простым нажатием клавиши n. В данном случае нажатие *nn выполнит обход всех совпадений и вернет нас туда, откуда мы начали.

Делайте изменения повторяемыми

После установки курсора в начало слова «content» мы можем изменить его. Для этого требуется выполнить два шага: удалить слово «content» и затем ввести слово замены. Команда `cw` удаляет символы от курсора до конца слова и выполняет переход в режим вставки, после чего можно ввести слово «copy». Vim записывает все нажатия клавиш, вплоть до выхода из режима вставки, поэтому вся последовательность `cwcopy<Esc>` рассматривается как единое изменение. Нажатие клавиши `.` удалит символы до конца текущего слова и заменит его словом «copy».

Теперь все вместе

Теперь все готово к действию! При каждом нажатии клавиши `n` курсор перемещается к следующему вхождению слова «content». А при нажатии клавиши `.` слово под курсором замещается словом «copy».

Если бы нам потребовалось слепо заменить все вхождения, мы могли бы просто отстучать на клавиатуре `n.n.n.` столько раз, сколько потребовалось бы для замены всех вхождений (хотя в этом случае можно было бы также использовать команду `:%s/content/copy/g`). Но нам нужно пропускать слова, не подлежащие замене. Поэтому после нажатия клавиши `n` мы проверяем смысл найденного совпадения и решаем, следует ли заменить его словом «copy». Если слово должно быть заменено, нажимаем клавишу `.`. Если нет – не нажимаем. Независимо от принятого решения, чтобы перейти к следующему вхождению, снова нажимаем клавишу `n`. И так, пока не будет выполнена замена всех требуемых вхождений.

Рецепт 6. Формула точки

Мы рассмотрели три простые задачи редактирования. Несмотря на различия между ними, мы смогли решить их с помощью команды «точка». В данном рецепте мы сравним найденные решения и выявим общий шаблон – оптимальную стратегию редактирования, которую я назвал «формула точки».

Обзор решений трех задач редактирования с помощью команды «точка»

В рецепте 2 нам потребовалось добавить точку с запятой в конец каждой строки. Мы изменили первую строку нажатием клавиш `A;<Esc>`, благодаря чему появилась возможность повторить изменение в каждой последующей строке с помощью команды «точка». Мы могли бы перемещаться между строками, выполняя команду `j`, и выполнить оставшиеся изменения, просто последовательно нажимая `j.` столько раз, сколько потребуется.

В рецепте 3 нам понадобилось окружить каждый символ `+` пробелами с обеих сторон. Чтобы найти этот символ в тексте, мы воспользовались командой `f+` и затем задействовали команду `s`, чтобы заменить один символ тремя. Благодаря такому подходу мы получили возможность выполнить поставленную задачу, нажав клавиши `;.` несколько раз.

В рецепте 5 нам нужно было заменить каждое вхождение слова «content» словом «copy». Для инициации поиска требуемого слова мы использовали команду `*` и затем выполнили команду `cw` для изменения первого вхождения. После этого мы получили возможность использовать команду `n` для перехода к следующему вхождению и команду `.` для применения того же изменения. Мы могли бы выполнить поставленную задачу, просто нажимая клавиши `n.` столько раз, сколько потребуется.

Идеальное решение: одна клавиша для перехода и одна для изменения

Во всех примерах мы использовали команду «точка», чтобы повторить последнее изменение. Но это не единственная общая черта. Для перемещения к следующей позиции применения изменений также потребовалось единственное нажатие клавиши.

Мы использовали одно нажатие для перемещения курсора и одно нажатие для выполнения изменения. Разве можно придумать что-то еще более удобное? Это – идеальное решение. Мы еще не раз столкнемся с этой стратегией редактирования, поэтому для краткости будем называть ее «формула точки».

Часть I. РЕЖИМЫ

Редактор Vim имеет модальный пользовательский интерфейс. То есть результат нажатия любой клавиши зависит от режима, активного в данный момент. Очень важно знать, какой режим активен в данный момент и как переключать режимы Vim. В этой части книги вы узнаете, как действует каждый режим и для каких целей они используются.