

Просто о Vim

v0.51 (для Vim версии 7)



Источник: <http://www.swaroopch.com/notes/Vim>

Vim

Введение

"Просто о *Vim*" это книга, которая поможет вам изучить редактор *Vim* (версии 7), даже если все, что Вы умеете, - это пользоваться компьютерной клавиатурой.

Первая часть книги предназначена для новичков, которые хотят понять, чем является *Vim* и узнать, как его использовать.

Вторая часть этой книги - для людей, которые уже знают, как использовать *Vim*, и хотят изучить особенности, которые делают *Vim* по настоящему мощным, такие как окна и вкладки, управление личной информацией, возможность использования его в качестве редактора для программиста, возможность расширить *Vim* с помощью собственных плагинов и многое другое.

Читай сейчас

Читайте всю книгу [онлайн](#).

Если вы обнаружили опечатки/орфографические ошибки, пишите на [почту](#)!

Купить книгу

Отпечатанную книгу [можно приобрести](#) для чтения в автономном режиме, и для поддержки дальнейшего развития и совершенствования этой книги. А можно просто [сделать пожертвования](#).

Скачать

- [PDF \(1.5MB\)](#)
- [Mediawiki XML dump](#) (197K)] (только для продвинутых пользователей)

Что сказали читатели

- Jay -- "Молодцы! Я использую *Vim* только 2/3 недели, и могу сказать, что он просто идеально подходит для начинающих, таких как я"
- Yosi Izaq -- "Книга очень хорошая и её интересно читать. Спасибо, что выложили её."
- Deepak -- "Ваши книги должны продаваться как горячие пирожки из-за их содержания."
- Joseph Sullivan -- "Отлично! Спасибо за вашу напряженную работу. Особенно понравилось, что все начинается с азов. После использования *Vim* в течение нескольких лет я многое забыл, как ни странно это покажется, так что мне приятно освежить память. Я уверен, что вы своей книгой пропагандируете *Vim*. ;-)"
- "wooden nickels" -- "Что я хочу сказать, если у вас появился компьютер, вы должны сразу же скачать и установить *Vim* на него и улучшить вашу жизнь. Лучшая возможность узнать о *Vim* - это прочитать эту книгу, она произведет революцию в вашем сознании о будущем текстовых редакторов."
- Josh Nichols -- "Листая 'Просто о *Vim*' нашел много нового для себя и это при том, что я использую *Vim* в течение многих лет."
- Raseel Bhagat -- Великая книга!! Хотя я использую *Vim* постоянно в качестве редактора, из этой книги я узнал, сколько еще он может сделать."
- Hiran Venugopalan --Замечательно! Это одна из самых необходимых книг. Я работал с *Vim* в течение последних лет, но никогда не видел большую часть возможностей его! Спасибо за книгу, Swaroop!
- Anonymous -- Это хорошая книга. Я давний пользователь *Vim*, но так и не удалось засунуть в мою голову *Vim* сценарии (кроме исправления некоторых ошибок в скриптах других). Это наилучшее введение в сценарии *Vim* (написание плагинов, синтаксис файлов, ...), из виденных мной. Спасибо, что выложили её в Интернете!
- Eduard Fabra -- "Спасибо Swaroop! Я начал читать её и должен сказать, что она очень хорошо написана. И я не сомневаюсь, что большое сообщество *Vim* пользователей будет улучшать её путем исправления, дополнения или небольшой правки — формат вики это отличная идея."
- А кроме того:

- книгу рекомендовал Bram Moolenaar (создатель VIM) на [официальном сайте Vim](#);
- книга была в топ списке в декабре 2008 на [Official Vim Tips вики](#).

Лицензия и условия

1. Эта книга лицензирована под лицензией [Creative Commons Attribution-Share Alike 3.0 Unported](#).

- Это означает следующее:
 - Вы свободно получаете, т.е. копируете, устанавливаете и передаете эту книгу
 - Вы свободно изменяете, т.е. адаптируете эту книгу при соблюдении следующих условий:
 - Авторство. Вы должны указывать авторство работы в порядке, предусмотренном автором или лицензией (но не таким способом, который предполагает, что они поддерживают вас или использование этой книги) .
 - Права копирования. Если вы изменяете или опираетесь на эту работу, вы можете распространять полученное произведение только на условиях такой же или аналогичной лицензии на неё.
 - Для любого повторного использования или распространения вы должны ясно дать понять другие условия лицензии на эту книгу.
 - Любое из вышеперечисленных условий может быть снято, если у вас есть разрешение владельца авторских прав.
 - Ничего в этой лицензии не ослабляет и не ограничивает моральных прав автора.

2. В атрибутах должна быть указана ссылка <http://www.swaroopch.com/notes/Vim> и ясно дано указание о том, что оригинальный текст можно взять из этого места.

3. Все коды/скрипты представлены в этой книге под лицензией [BSD 3-ей версии](#), если не указано иное.

4. Некоторые примеры используемых текстов в этой книге были получены из <http://en.wikipedia.org> и <http://en.wikiquote.org> под [GNU Free Documentation License](#).

5. Вклад добровольцев в эту оригинальную книгу должен быть под этой же лицензией и авторские права должны быть отнесены к главному автору этой книги.

Переводы

Если вы заинтересованы в вычитывании или создании перевода этой книги, на другие человеческие языки, см. [Переводы](#).

Внешние ссылки

<http://www.swaroopch.com/buybook>
<http://www.swaroopch.com/byteofdonate>
http://www.swaroopch.com/files/byteofvim/byte_of_vim_v050.pdf
http://www.swaroopch.com/files/byteofvim/byte_of_vim_v050.xml
http://groups.google.com/group/vim_use/msg/e1625069d4ea0ef9
http://groups.google.com/group/vim_use/msg/09ca306a67b9d2cd
<http://twitter.com/peerlessdeepak/status/1024279089>
http://groups.google.com/group/vim_use/msg/362a82a4af132317
<http://woodennickels.posterous.com/text-editing-your-way-to-heave>
<http://twitter.com/techpickles/status/1025775542>
<http://twitter.com/raseel/status/1024291090>
<http://www.swaroopch.com/blog/a-free-book-on-vim/#comment-116472>
<http://www.swaroopch.com/notes/Talk:Vim>
http://groups.google.com/group/vim_use/msg/dac94f3332f733e4
http://vim.wikia.com/wiki/Main_Page#Did_you_know.3F_view_archive
<http://creativecommons.org/licenses/by-sa/3.0/>
<http://www.opensource.org/licenses/bsd-license.php>
http://en.wikipedia.org/wiki/Wikipedia:Text_of_the_GNU_Free_Documentation_License

Vim :Содержание

- Титульная страница
- Переводы
- 1. Предисловие
- 2. Введение
- 3. Установка
- 4. Первые шаги
- 5. Режимы
- 6. Умение печатать
- 7. Перемещение
- 8. Помощь (Help)
- 9. Основы редактирования
- 10. Продвинутое редактирование
- 11. Множественность
- 12. Управление персональной информацией
- 13. Сценарии
- 14. Плагины
- 15. Редактор для программистов
- 16. Разное
- 17. Что дальше
- 18. Обратная связь
- 19. Благотворительность
- 20. Послесловие
- 21. Версии

Vim :Предисловие

О Vim

Vim - это компьютерная программа, используемая для создания и редактирования текстовых файлов. Она также предоставляет ряд возможностей, которые помогут вам делать это лучше.

Почему Vim?

Посмотрим правде в глаза, очень редко, кто делает свои лучшие работы с первой попытки. Скорее всего, вы будете много раз пробовать, пока не станет получаться «хорошо».

Как Louis Brandeis однажды сказал: "Нет великих писателей, есть только великие переписыватели."

Внесение многочисленных быстрых изменений стало бы намного легче, если бы мы использовали редактор, который помогал бы нам в этом, и это именно то, с чем *Vim* прекрасно справляется, и он гораздо лучше по сравнению с большинством текстовых редакторов и богатых текстовых процессоров.

Почему написана эта книга?

Я использую *Vim* как редактор с тех пор, как я научился использовать старый *vi* во время занятий Unix в колледже. *Vim* является одной из нескольких программ, которые я использую по 10 часов в сутки. Я знал, что есть много функций, о которых я еще не знаю, но которые потенциально могут быть полезны для меня, поэтому я начал изучать *Vim* мало-помалу.

Для кристаллизации моего понимания и для помощи другим в изучении *Vim*, я начал писать эту коллекцию статей, и назвал это книгой.

Принципы, которые я постарался сохранить при написании этих заметок, это:

1. Простое изложение. Важно не забывать об этом.
2. Акцент на примерах с практическими рекомендациями.
3. Книга должна содержать информацию для читателей разного уровня, изучающих *Vim*, - от начальных знаний до изучения продвинутого материала.
4. Нужно научить пользователя, как сделать что-то в *Vim*, например, настройка режимов буфера. Большинство людей знают только основные команды *vi* и не пытаются узнать что-либо сверх того. Изучение таких понятий является переломным моментом, они становятся крутыми *Vim* пользователями т.е. *Vimmers*, это означает, что они извлекают максимальную пользу из *Vim*, вот это и является целью этой книги.
5. Много из представленного здесь является руководством для людей, которые используют *Vim* как IDE и т.д. Существуют различные способы сделать так или этак, пользователь может узнать, какие есть плагины, чтобы их использовать, в книге изложены основы для самостоятельной работы читателя.
6. Достаточный объем информации, чтобы заставить вас понять и использовать, а не перечисление всего подряд (принцип Парето).
7. Книгу не следует пытаться использовать как справочное руководство. Где необходимо, она должна просто указать на соответствующие возможности. Таким образом, нет избыточности, пользователь учится использовать удивительное встроенное справочное руководство, которое важно, а книга имеет свои собственные сильные стороны.

Подводя итог, повторяем мантру: Понятия. Примеры. Навык.

Статус книги

Эта книга находится в стадии разработки и я еще не готов дать ей версию "1.0". Конструктивные предложения только приветствуются. Пожалуйста, добавьте свои мысли и предложения в раздел "Обсуждение" в левой боковой панели на любой странице на официальном сайте, или же напишите мне.

Официальный Веб-сайт

Официальный сайт книги <http://www.swaroopch.com/notes/Vim>. С сайта вы можете читать книжку онлайн или скачать последнюю версию книги, и также отправить мне предложения.

Мысли на заметку

«Книги не пишутся - они переписываются. В том числе свои собственные. Одна из самых больших трудностей, - это признать, что после седьмой коррекции нужно еще что то переписать.» -- Майкл Крайтон.

«Совершенство достигается не тогда, когда больше нечего добавить, а когда нечего отнять.» -- Антуан де Сент-Экзюпери.

Внешние ссылки

<http://www.vim.org>

<http://www.swaroopch.com/contact/>

<http://creativecommons.org/licenses/by-sa/3.0/>

<http://www.opensource.org/licenses/bsd-license.php>

<http://en.wikipedia.org/wiki/>

Vim : Введение

Что такое Vim?

Vim это компьютерная программа, используемая для написания любого текста, будь то список покупок, книга или программный код.

Что делает *Vim* уникальным, так это то, что он является одновременно простым и мощным.

Простота позволяет легко начать работу с *Vim*. Простота означает, что он имеет минималистский интерфейс, который позволяет вам сконцентрироваться на основной задаче — писать. Простота означает, что он построен на нескольких основных концепциях, понимание которых поможет вам легко изучить более глубокий функционал.

Мощность заключается в том, что многие вещи могут быть сделаны быстрее, лучше и легче. Мощные средства делают не очень простые вещи возможными. Мощные не значит, что они должны быть сложными. Мощные средства следуют парадигме "Минимум усилий. Максимум эффекта".

Что может делать Vim?

Я слышу, как вы говорите: "Итак - это текстовый редактор. Подумаешь большое дело?"

Да, большое.

Давайте рассмотрим несколько случайных примеров, сравнив *Vim* с обычным редактором. Цель этого упражнения для вас - ответить в каждом примере на вопрос: «Как бы я это сделал в используемом мной сейчас редакторе?».

Примечание: Не слишком сосредотачивайтесь сейчас на деталях команд *Vim*, основная задача этих примеров - познакомить вас с возможностями *Vim*, не объясняя, как это работает. Тому, как это работает, будет посвящена оставшаяся часть книги.

Редактирование	В <i>Vim</i>
Как вы переместите курсор вниз на 7 строк?	Нажмите <code>7j</code>
Как вы удалите слово? Именно слово.	Нажмите <code>dw</code>
Как вы ищете в файле слово, на котором в данный момент находится курсор?	Нажмите <code>*</code>
Как произвести поиск и замену только в строках с 50-ой по 100-ую?	Выполнить <code>:50,100s/old/new/g</code>
Что делать, если вы хотите посмотреть две разные части одного и того же файла одновременно?	Выполнить <code>:sp</code> для разделения ('split') показываемого
Что делать, если вы хотите открыть файл, имя которого прописано в текущем документе и курсор стоит на нем?	Нажмите <code>gf</code> (что означает 'g'o для этого 'file)
Что делать, если вы хотите выбрать лучшую цветовую схему дисплея?	Выполнить <code>:colorscheme desert</code> , для выбора цветовой схемы "desert" (моей любимой)
Что делать, если вы хотите, чтобы сочетание клавиш Ctrl-S сохраняло файл?	Выполните <code>:nmap <c-s> :w<CR></code> . Где <CR> это "c'arriage r'eturn, т.е. возврат каретки.
Что делать, если вы хотите сохранить все открытые файлы и измененные вами настройки, так, чтобы вы могли продолжить редактирование позже?	Выполните <code>:mksession ~/latest_session.vim</code> , и откройте <i>Vim</i> в следующий раз вот так <code>vim -S ~/latest_session.vim</code> .
Что делать, если вы захотели увидеть ваш код с подсветкой синтаксиса?	Выполните <code>:syntax on</code> . Если <i>Vim</i> неправильно распознает язык, то используйте <code>:set filetype=Wikipedia</code> , к примеру.
Что делать, если вы хотите исключить из просмотра некоторые части вашего файла с тем, чтобы вы могли сосредоточиться только на одной части?	Выполните <code>:set foldmethod=indent</code> предполагается, что ваш файл имеет правильные отступы. Есть также и другие методы складывания.

<p>Что делать, если вы хотите открыть несколько файлов во вкладках?</p>	<p>Используйте <code>:tabedit <file></code> для открытия нескольких файлов в "tabs" (как в любимом браузере), и используйте <code>ctrl-pgup/ctrl-pgdn</code> для переключения между вкладками.</p>
<p>Вы часто используете какие-то слова в документе, как можно быстро вставлять их в следующий раз?</p>	<p>Нажмите <code>ctrl-n</code> и смотрите список завешений ("completions") для текущего слова, основанный на всех словах, которые вы используете в текущем документе. Или используйте <code>:ab mas Maslow's hierarchy of needs</code> для раскрытия аббревиатуры автоматом, когда вы наберете <code>m a s <space></code>.</p>
<p>У вас есть какие-то данные, где только первые 10 символов в каждой строке являются полезными, а остальное уже не нужно для вас. Как вы получите эти данные?</p>	<p>Нажмите <code>ctrl-v</code>, выделите текст и нажмите <code>y</code> для копирования выделенных строк и столбцов текста.</p>
<p>Что делать, если вы получили документ от кого-то, в котором все буквы находятся в верхнем регистре, это вас раздражает и вы хотите преобразовать их в нижний регистр?</p>	<p>В <i>Vim</i>, выполните следующее:</p> <pre data-bbox="807 674 1469 763">:for i in range(0,line('\$')) :call setline(i,tolower(getline(i))) :endfor</pre> <p>Не волнуйтесь, подробности будут рассмотрены в последующих главах. Более короткий способ сделать это же - запустить <code>:%s#\(\.\)\#\ I\ 1#g</code>, но с другой стороны, способ выше проще вспомнить. Существует еще более простой способ выбрать весь текст (<code>ggVG</code>) и использовать оператор <code>u</code>, чтобы конвертировать в нижний регистр, но опять же это слишком легко, и не так круто, как представленный выше способ заставить <i>Vim</i> делать это.</p>

Уф. Вы уже убеждены?

В этих примерах можно увидеть силу *Vim* в действии. Любой другой редактор сделает это безумно сложно для одинакового уровня функциональности. И тем не менее, удивительно, что вся эта мощь достигается так просто.

Обратите внимание, что мы ни разу не использовали мышь в этих примерах! Это хорошо. Подсчитайте, сколько раз вы переносите руку между клавиатурой и мышью за день, и вы поймете, почему это хорошо, избегайте этого, когда это возможно.

Я надеюсь, вы не ошеломлены. В *Vim* хорошо то, что вам не нужно знать все возможности для продуктивной работы с ним, вам просто нужно знать несколько основных понятий.

После изучения основных концепций, все остальные возможности могут быть легко освоены, когда они вам понадобятся.

Vim :Установка

Давайте посмотрим, как получить и установить *Vim* на ваш компьютер.

Windows

Если вы используете Microsoft Windows, следующие шаги помогут вам скачать и установить последнюю версию установщика *Vim 7* на ваш компьютер:

1. Зайдите на <http://www.vim.org/download.php#pc>
2. Скачайте "Self-installing executable" ([gvim72.exe](#), как здесь написано)
3. Двойным кликом на файле запустите его и установите *Vim* как любую другую программу для Windows.

Mac OS X

Если вы используете Mac OS X, вы уже имеете установленную терминальную версию *Vim*. Запустите меню команд `Finder` → `Applications` → `Utilities` → `Terminal`. В терминале запустите команду `vim` и нажмите `enter`, вы должны увидеть экран приглашения *Vim*.

Если вы хотите использовать графическую версию *Vim*, скачайте последнюю версию [Cocoa-based MacVim project](#). Двойным кликом на файле (типа `MacVim-7_2-stable-1_2.tbz`), он будет разархивирован и будет создана директория `MacVim-7_2-stable-1_2`. Откройте каталог, и скопируйте приложение `MacVim` в ваш каталог приложений.

Для большей информации о `MacVim`, включая как запустить `MacVim` из терминала, смотри руководство `macvim`:

1. Нажмите на `Finder` → `Applications` → `MacVim`.
2. Наберите `:help macvim` и нажмите клавишу `Enter`.

Linux/BSD

Если вы используете Linux или *BSD систему, вы имеете уже установленную минимальную консольную версию *Vim*. Откройте терминал типа `konsole` или `gnome-terminal`, запустите `vim` и вы увидите окно приглашения *Vim*.

Если вы получите сообщение `«vim: command not found»`, значит *Vim* не установлен. Вы можете установить *Vim* имеющимся в системе установщиком, таким как `aptitude` в Ubuntu/Debian Linux, `yum` в Fedora Linux, `pkg_add` или `port` во FreeBSD, `yast` в openSuSE и т.д. Пожалуйста, посмотрите документацию по вашему установщику или узнайте на форуме, как установить новый пакет.

Если вы хотите графическую версию, установите `vim-gnome` пакет или альтернативный, `gvim` пакет.

Итоги

В зависимости от того, как он установлен, вы можете запустить команду `vim` в терминале или использовать меню операционной системы, чтобы открыть графическую версию приложения *Vim*.

Теперь, когда мы установили *Vim* на вашем компьютере, перейдем к его использованию, в следующей главе.

Внешние ссылки

<ftp://ftp.vim.org/pub/vim/pc/gvim72.exe>
<http://code.google.com/p/macvim/>

Vim :Первые шаги

Запуск Vim

Первым делом, конечно, нужно узнать, как запустить *Vim*.

Графическая версия

Windows: Нажмите `Start` → `Programs` → `Vim 7` → `gVim`.

Mac OS X: Нажмите `Finder` → `Applications` → `MacVim`.

Linux/BSD: Нажмите `Applications` → `Accessories` → `GVim Text Editor`, или нажмите `Alt+F2`, наберите `gvim` и нажмите клавишу `enter`.

Терминальная версия

Windows: Нажмите `Start` → `Run`, наберите `vim` и нажмите клавишу `enter`.

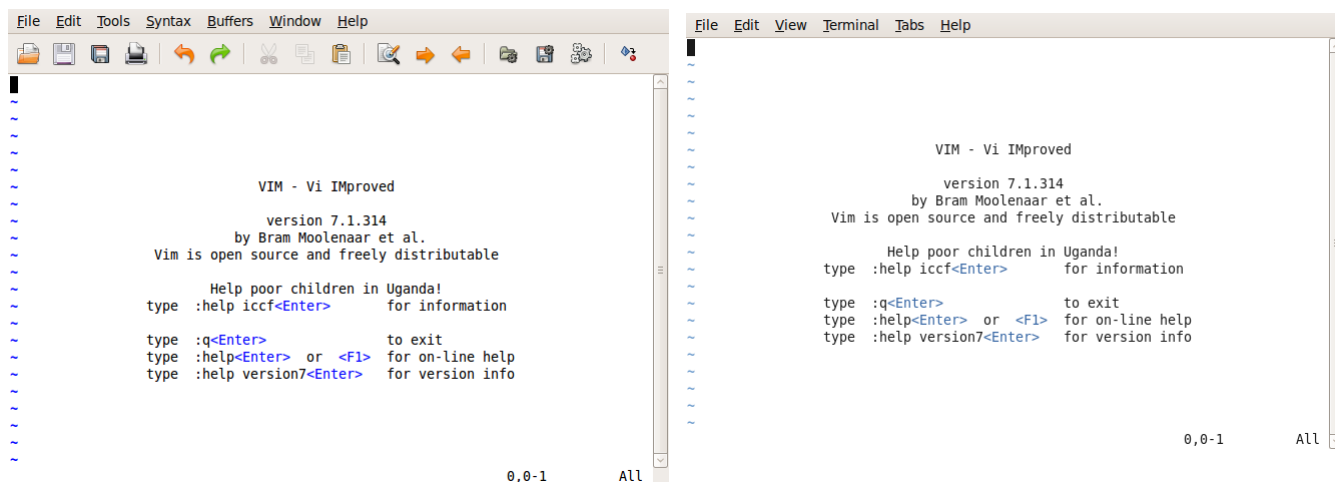
Mac OS X: Нажмите `Finder` → `Applications` → `Utilities` → `Terminal`, наберите `vim` и нажмите клавишу `enter`.

Linux/BSD: Нажмите `Applications` → `Accessories` → `Terminal`, или нажмите `Alt+F2`, наберите `konsole/gnome-terminal` и нажмите клавишу `enter`. Затем, наберите `vim` и нажмите клавишу `enter`.

Отныне, когда мы говорим «открыть *Vim*», воспользуйтесь одним из методов, упомянутых выше.

Примечание: Когда вы запустите *Vim*, вы увидите, что вы не можете сразу начать печатать текст. Не паникуйте, все будет объяснено через некоторое время.

Графическая или терминальная?



Графическая версия *Vim* имеет меню в верхней части приложения, а также различные опции, доступные с помощью мыши, но учтите, что это совершенно необязательно. Вы можете получить доступ ко всем функциям *Vim* только с помощью клавиатуры.

Почему это важно? Потому что работа человека будет намного эффективна, если использовать только клавиатуру, чем при использовании мыши, в этом случае человек гораздо быстрее набирает текст и делает меньше ошибок.

Это происходит потому, что нет лишних движений руки, необходимых для переключения между клавиатурой и мышью, и нет переключения контекста в сознании человека при переносе руки между клавиатурой и мышью. Нужно взять за привычку использовать клавиатуру как можно больше, Вы экономите ценные движения руки.

Конечно, это субъективно. Некоторые люди предпочитают мышь, а некоторые предпочитают клавиатуру. Я рекомендую вам использовать клавиатуру как можно больше, чтобы узнать реальную мощь *Vim*.

Введение в режимы

Представьте себе, что в субботу вечером вы устали от телепрограмм. Вы хотите посмотреть старый

любимый фильм. Вы переключаете телевизор в режим видео так, чтобы он показывал то, что отображает DVD-плеер, вместо кабельных каналов. Обратите внимание, что телевидение по-прежнему транслирует видео, но вы переключаете контекст на просмотр DVD или на транслируемые телепередачи.

Аналогично, *Vim* имеет режимы. Например, *Vim* имеет режим для ввода текста, режим выполнения команд и т.д. Все они связаны основной целью редактирования текста, но вы переключаете контекст в зависимости от того, нужно просто набрать текст, или вы хотите выполнить некоторые команды над текстом.

Разве это не просто?

Традиционно концепция режимов является наиболее часто приводимой причиной начинающих о том, почему они считают *Vim* "запутанным". Я сравниваю его с ездой на велосипеде — стоит вам проехать несколько раз и, как только у вас начнет получаться, вы будете удивляться, какая суета была раньше.

Так почему же в *Vim* есть режимы? Чтобы сделать все как можно проще, даже если их использование может на первый взгляд показаться «странным».

Что я подразумеваю под этим? Давайте возьмем пример: одна из ключевых целей в *Vim* - делать все с клавиатуры, чтобы не нужно было использовать мышь (вы можете использовать мышь, если вы хотите, но это строго по желанию). В таком случае, как бы вы различали текст, который вы хотите написать, и команды, которые вы хотите запустить?

Решение, предлагаемое в *Vim*, состоит в том, чтобы иметь "нормальный" режим, где вы можете выполнять команды, и режим "вставки", в котором вы просто вводите текст. Вы можете постоянно переключаться между двумя режимами.

Например, нажав **i** вы переключаете *Vim* в режим вставки, а нажатием **<Esc>** вы переключаете *Vim* в нормальный режим.

Как это делают традиционные редакторы, как они отличают команды и ввод текста? С помощью графического меню и комбинаций клавиш. Проблема в том, что это решение не масштабируется.

Прежде всего, если у вас есть сотни команд, создание меню для каждой из этих команд будет безумным и запутанным. Во-вторых, настройка использования каждой из этих команд будет еще более трудным делом.

Давайте возьмем конкретный пример. Предположим, вы хотите заменить в документе все вхождения слова "from" на слово "to". В традиционном редакторе, вы можете открыть меню, Правка -> Заменить (или использовать сочетание клавиш, типа Ctrl-R), а затем ввести слово "from" и слово "to", а затем нажать на кнопку "Заменить". После проверки можно выбрать вариант "Заменить все". В *Vim*, вы просто выполняете :
`%s/from/to/g` в нормальном режиме. Команда `:s` является аналогом команды "заменить".

Посмотрите, не правда ли, просто?

Что делать, если вы теперь хотите запустить эту замену только в 10 первых строках текста и вы хотите иметь подтверждения да/нет для каждой замены? В традиционных текстовых редакторах, получить подтверждения да/нет легко, сняв флажок "Заменить все", но сначала, вы должны найти, что менять и затем с помощью мыши щелкнуть на опции (или использовать длинную последовательность клавиш клавиатуры). Но как вы укажете, что нужно заменить только в первых 10 строках? В *Vim*, вы можете просто запустить :
`0,10s/from/to/gc`. Новая опция **c** ('c'onfirmation), которую мы используем означает, что мы хотим подтверждение для каждой замены.

Разделение режимов на ввод (вставка) и командный (нормальный) делает *Vim* легким для нас, и простым при переключении режимов.

Обратите внимание, что первые шаги в использовании *Vim* кажутся немного "странными", немного "чуждыми", но как только вы попробуете его в действии, поймете что это имеет смысл. Самое приятное то, что эти основные концепции помогут вам понять все, что вам нужно знать о том, как использовать *Vim*.

Теперь вы понимаете разницу между обычным режимом и режимом вставки. Работая в нормальном режиме вы можете использовать различные команды, и вы можете сразу начать использовать их. Сравните это с изучением новых команд в традиционных редакторах, где сперва нужно прочесть множество документации, много искать по меню методом проб и ошибок, или просто просить кого-то помочь.

Лично я считаю, что названия режимов непонятны для начинающих. Я предпочитаю называть режим вставки как режим "записи" и нормальный режим работы, как режим "перезаписи", но мы будем придерживаться стандартной терминологии *Vim*, чтобы избежать путаницы.

Примечание: Все команды в нормальном режиме должны заканчиваться клавишей **enter**, чтобы сказать *Vim*, что мы написали полную команду. Так что когда мы говорим «выполните команду `:help vim-modes-intro`», это означает, что вы должны ввести `:help vim-modes-intro`, а затем нажать клавишу **enter** для завершения команды.

Запись в файл

Давайте теперь посмотрим, как открыть, отредактировать и закрыть файл в *Vim*.

1. Откроем *Vim*.
2. Наберем `:edit hello.txt` и нажмем `enter`.
3. Нажмем `i`.
4. Введем текст `Hello World`.
5. Нажмем клавишу `<Esc>`.
6. Наберем `:write` и нажмем клавишу `enter`.
7. Закроем *Vim*, запустив `:q`.



Поздравляю! Вы создали свой первый файл :-).

Много пришлось сделать операций?

Да, на первый взгляд, это так. Это потому что в первый раз, потом мы привыкаем к открытию *Vim*, записи в файл и закрытию *Vim*. Вы должны понимать, что это будет лишь незначительная часть вашего времени по сравнению с фактическим временем, которое уходит на редактирование содержания документов.

Давайте посмотрим, что делают вышеперечисленные команды.

- `:edit hello.txt` или просто `:e hello.txt` — открывает файл для редактирования. Если файл с указанным именем не существует, то *Vim* создаст его при сохранении файла.
- Нажатие `i` - переключает *Vim* в режим вставки.
- Набор текста `Hello World` — ввод нужного вам текста в файл.
- Нажатие `<Esc>` - возвращает *Vim* в нормальный режим
- `:write` или просто `:w` — говорит *Vim*, что нужно записать текст (который пока сохранен в памяти компьютера) в файл на жесткий диск. Это означает, что все, что мы написали, будет теперь храниться постоянно.
- `:quit` или просто `:q` - закрыть файл в текущем окне. Если это было единственное открытое окно в *Vim*, это также приведет к закрытию *Vim* (Концепция окон будет обсуждаться в следующей главе).

Попробуйте повторить этот процесс несколько раз с разными именами файлов, другим текстом, и т.д., пока вы не привыкнете к базовым действиям в использовании *Vim*.

Обратите внимание, что когда вы находитесь в режиме вставки, *Vim* показывает - INSERT - в нижнем левом углу. При переключении в нормальный режим, он ничего не показывает. Это потому, что нормальный режим является режимом по умолчанию, в котором работает *Vim*.

Потратьте некоторое время, чтобы отдохнуть от полученной информации, это, пожалуй, самый тяжелый урок, который нужно узнать о *Vim*, остальное все просто:)

И не волнуйтесь, помощь не слишком далеко. На самом деле, просто запустите команду `:help`. Например, выполните `:help :edit`, и вы увидите открытую документацию. Идем дальше, пробуйте.

Итоги

Мы уже обсудили основные понятия и использование *Vim*. См. `:help notation` и еще `:help keycode`.

Убедитесь, что вы досконально поняли эту концепцию. Как только вы начнете "думать в *Vim*", остальные функции *Vim* будут очень простыми.

Vim :Режимы

Введение

У нас была первая встреча с режимами в предыдущей главе. Теперь, давайте изучим концепцию режимов детальнее и узнаем, что мы можем сделать в каждом режиме.

Типы режимов

Есть три основных режима в *Vim* - нормальный, вставки и визуальный.

- Нормальный режим — это тот, в котором вы можете выполнять команды. Это основной режим, в который переходит *Vim* после запуска.
- Режим вставки — это режим, в котором вы можете набирать нужный текст.
- Визуальный режим — это где вы визуально выбираете часть текста, которую нужно обработать командой/операцией.

Нормальный режим

Изначально, вы попадете в нормальный режим. Давайте посмотрим что вы можете делать в этом режиме.

Наберите `:echo "hello world"` и нажмите `enter`. Вы увидите знаменитые слова `hello world`. То, что вы только что сделали, это запустили команду *Vim*, называемую `:echo`, и вы добавили текст к ней, который и был успешно выведен.

Наберите `/hello` и нажмите клавишу `enter`. *Vim* будет искать эту фразу и перейдет к первому её вхождению.

Это было только два простых примера команд, доступных в нормальном режиме.

Мы рассмотрим много других команд в последующих главах.

Как использовать помощь

Почти так же важно, как знать о нормальном режиме, это уметь пользоваться командой `:help`. Здесь вы можете больше узнать о командах, доступных в *Vim*.

Запомните, вам не нужно знать все команды, доступные в *Vim*, достаточно просто знать, где найти их, когда они вам понадобятся. Например, смотрим `:help usr_toc` дает нам оглавление справочного руководства. Вы можете посмотреть `:help index` для поиска для поиска определенной темы, интересной вам, для примера, запустите `/insert mode` для просмотра информации относительно режима вставки.

Если вы не можете запомнить эти две темы, нажмите `F1` или просто запустите `:help`.

Режим вставки

Когда *Vim* запускается, он открывается в нормальном режиме, вы можете использовать `i` для перехода в режим вставки.

Есть и другие способы переключения из нормального режима в режим вставки, такие как:

- Запустите `:e dapping.txt`
- Нажмите `i`
- Введите следующий абзац (включая все опечатки и ошибки, мы исправим их позже): `means being determined about being determined and being passionate about being passionate`
- Нажмите клавишу `<Esc>` для переключения назад в нормальный режим.
- Запустите `:w`

Упс, мы, кажется, упустили слово в начале строки, а наш курсор находится в конце строки, что нам теперь делать?

Наиболее эффективным способом было бы перейти в начало строки и вставить пропущенное слово? Должны ли мы использовать мышь, чтобы переместить курсор в начало строки? Должны ли мы использовать клавиши со стрелками для перемещения к началу строки? Должны ли мы нажать клавишу `home` и затем нажать `i` для переключения снова в режим вставки?

Оказывается, что наиболее эффективным способом будет нажатие `I` (верхний регистр `I`):

- Нажмите `I`
- Напишите `Dappin`
- Нажмите клавишу `<Esc>` для переключения назад в нормальный режим.

Обратите внимание, что мы использовали другой ключ для переключения в режим вставки, его особенностью является то, что он перемещает курсор в начало строки, а затем переключает в режим вставки.

Также обратите внимание, это важно, что нужно вернуться в нормальный режим, как только вы закончите вводить текст. Эта привычка будет полезна, потому что большинство ваших работ (после начальной фазы написания статьи) будет проводиться в нормальном режиме - где происходят все важные действия: перезапись, редактирование, полировка.

Теперь, давайте возьмем другой вариант команды **i**. Обратите внимание, что нажатие **i** поместит курсор на предыдущую позицию и включит режим вставки. Чтобы разместить курсора после текущей позиции, нажмите **a** ('a'fter).

- Нажмите **a**
- Наберите **g** (для завершения слова "Dapping")
- Нажмите **<Esc>** для переключения в нормальный режим

Как и в отношениях между **i** и **I** ключами, существует связь между ключами **a** и **A** - если вы хотите добавить текст в конце строки, нажмите клавишу **A**.

- Нажмите **A**
- Наберите **.** (поставьте точку для завершения предложения)
- Нажмите **<Esc>** для переключения в нормальный режим

Подведем итог, что мы узнали о четырех клавишах:

Команда	Действие
i	вставить текст до курсора
I	вставить текст с начала строки
a	добавить текст после курсора
A	добавить текст с конца строки

Обратите внимание, как команды в верхнем регистре "больше" версий команд в нижнем регистре.

Теперь, когда мы умеем быстро двигаться в текущей строке, давайте посмотрим, как перейти на новые строки. Если вы хотите создать ('o'pen) новую строку и начать ввод данных, нажмите клавишу **o**.

- Нажмите **o**
- Наберите **I'm a rapper.**
- Нажмите **<Esc>** для переключения в нормальный режим.

Хм, было бы более интересно, если бы новое предложение мы написали в новом абзаце.

- Нажмите **O** (верхний регистр 'O')
- Нажмите **<Esc>** для переключения в нормальный режим

Подводя итог двум новым ключам мы запоминаем:

Команда	Действие
o	открыть новую строку ниже
O	открыть новую строку выше

Обратите внимание, как верхний и нижний режим команды 'o' противоположны по направлению, в котором они открывают строку.

Было ли что-то не так в тексте, который мы только что написали? А да, это должно быть "Dapper", а не rapper! Один символ мы должны изменить, как эффективно это сделать?

Мы можем нажать **i** для переключения в режим вставки, нажать клавишу **** для удаления **r**, набрать **d** и затем нажать **<Esc>** для переключения и выхода из режима вставки. Но это четыре шага для такого простого изменения! Есть ли что то лучше? Вы можете использовать клавишу **s** - **s** для замены ('s'ubstitute).

- Перемещаем курсор к символу **r** (или просто нажимаем **b** для перехода назад ['b'ack] к началу слова)
- Нажимаем **s**
- Набираем **d**

- Нажмите `<Esc>` для переключения в нормальный режим

Ну, ладно, возможно в данном случае это не сильно нас спасло, но представьте себе что этот процесс повторяется снова и снова в течение всего дня! Создание таких элементарных, максимально быстрых операций, это выгодно, потому что они помогают нам сконцентрировать наши усилия на более творческих и интересных аспектах работы. Как говорит Линус Торвалдс, "это означает не только то, что можно сделать что-то быстро, но поскольку это быстро, метод, которым вы делаете вашу работу, кардинально меняется".

Опять же, есть `S`, большая версия клавиши `s`, которая заменяет всю строку, а не текущий символ.

- Нажмите `S`
- Наберите `Be a sinner.`
- Нажмите `<Esc>` для переключения в нормальный режим

Команда	Действие
<code>s</code>	замена текущего символа
<code>S</code>	замена текущей строки

Давайте вернемся к нашему последнему действию... Не могли бы мы сделать его более эффективно, раз мы хотим заменить (`'r'eplace`) только один символ? Да, мы можем использовать клавишу `r`.

- Двигаем курсор к первому символу слова `sinner`.
- Нажимаем `r`
- Набираем `d`

Обратите внимание, что мы еще в нормальном режиме и нам не нужно нажимать `<Esc>`.

Большая версия `r` называется `R`, она заменяет последовательно расположенные символы.

- Установите курсор на `'i'` в `sinner`.
- Нажмите `R`
- Наберите `app` (слово теперь становится `'dapper'`)
- Нажмите `<Esc>` для переключения в нормальный режим.

Команда	Действие
<code>r</code>	Заменить текущий символ
<code>R</code>	Заменить несколько последовательно расположенных символов

Текст должен выглядеть так:

```
Dapping means being determined about being determined and being
passionate about being passionate. Be a dapper.
```

Уф. Мы рассмотрели много в этой главе, но я гарантирую, что это самый трудный шаг. Как только вы усвоите все это, вы поймете сердцем и душой как работает *Vim*, и все другие функции в *Vim* - это просто глазурь на торте.

Повторюсь, понимание того, как работают режимы и как переключаться между режимами, является ключом к пониманию *Vim*, так что если вы еще не понимаете, пересмотрите вышеизложенные примеры еще раз, пожалуйста, не стесняйтесь читать их снова. Делайте это когда вам понадобится.

Если вы хотите прочитать больше о этих командах, смотри [:help inserting](#) и [:help replacing](#).

Режим визуализации

Предположим, что вы хотите выделить несколько слов и заменить их полностью каким-то новым текстом, который вы хотите написать. Что вы будете делать?

Один из способов заключается в использовании мыши, щелкните в начале текста, который вас интересует, удерживая левую кнопку мыши, перетащите мышью до конца соответствующего текста, а затем отпустите левую кнопку мыши. Но зачем там много работать?

Мы могли бы использовать клавиши `` или `<Backspace>`, чтобы удалить все символы, но кажется это еще хуже по эффективности.

Наиболее эффективным способом было бы поместить курсор в начало текста, нажать `v`, чтобы начать

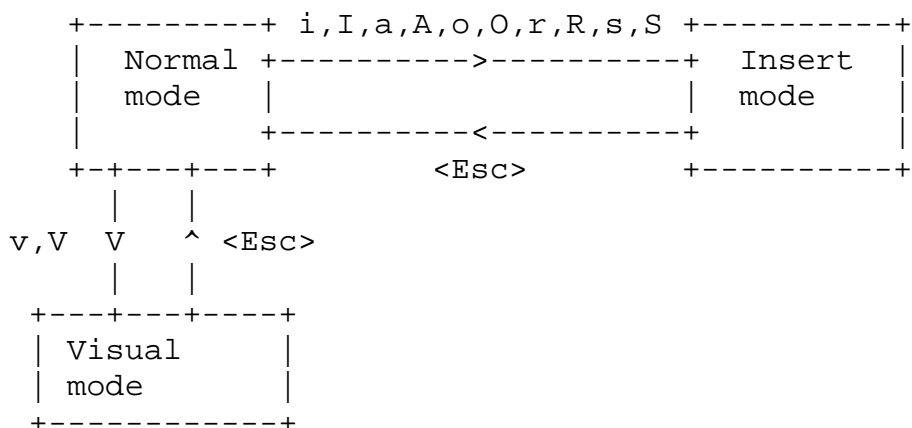
визуальный режим, используя клавиши со стрелками или какие-либо текстовые команды перейти в конец соответствующего текста (например, нажмите `5e`, для перехода к концу пятого слова от текущей позиции курсора), а затем нажмите `c`, чтобы изменить ("change) текст. Обратите внимание на возросшую эффективность.

В этой конкретной операции (команда `c`), вы будете переведены в режим вставки, поэтому после его окончания нажмите `<Esc>`, чтобы вернуться в нормальный режим.

Команда `v` работает с символами. Если вы хотите работать со строками, используйте верхний регистр `V`.

Итоги

Это схема отношений между различными режимами:



(Это нарисовано с использованием *Vim* и плагина [Dr.Chip's DrawIt](#))

Смотри `:help vim-modes-intro` и `:help mode-switching` для подробного описания различных режимов и переключение между ними, соответственно.

Если вы все еще не уверены в том, что концепция режимов является основной силой, обеспечивающей мощь и простоту *Vim*, читайте статьи [«Почему Vi»](#) и [о модели ввода vi](#), которые показывают, что это лучший способ редактирования.

Внешние ссылки

http://vim.sourceforge.net/scripts/script.php?script_id=40

<http://www.viemu.com/a-why-vi-vim.html>

<http://blog.ngedit.com/2005/06/03/the-vi-input-model/>

Vim :Умение печатать

Введение

Мы изучили, как переключаться между тремя базовыми режимами в *Vim*. Но запомнить все эти ключи достаточно сложно! Как вы запомните, какой ключ для какой операции? Да, это "ключевой" вопрос. Ответ в том, что вы не должны «помнить», ваши пальцы должны автоматически знать, что делать!. Это должно быть (буквально) на кончиках ваших пальцев.

А как мы это сделаем? Сделайте это привычкой. Навык хождения не появляется у человека с рождением, но после некоторых попыток, и по привычке, мы учимся ходить. То же самое с *Vim*, хотя и требует меньше усилий.

Vim был разработан для людей, которые дружат с клавиатурой. Кто это? Поскольку мы проводим большую часть времени в задачах редактирования, которые широко применяют клавиатуру, то чем быстрее мы сможем вводить, тем быстрее мы выполним работу.

Давайте начнем с базовой техники, чтобы вы комфортно работали с клавиатурой.

Техника домашней строки

Разместите ваши пальцы на [домашней строке](#) клавиатуры, а руки так, чтобы пальцы вашей левой руки были на клавишах `ASDF`, а пальцы вашей правой руки - на клавишах `JKL`, как показано на рисунке ([автор](#)

[неизвестен](#)).



Разместите ваши руки удобно, это важно в обучении эффективным пользованием клавиатурой. Идея в том, что вы должны иметь возможность нажать любую клавишу, используя палец, который находится ближе всего к этой кнопке, а затем ваш палец должен автоматически возвратиться в исходное положение. Это положение рук на клавиатуре покажется сложным в начале, но, попробовав несколько раз, вы увидите, что таким образом вы будете набирать гораздо быстрее.

Примечание: клавиатура имеет маркеры на клавишах F и J которые напоминают вам, где должны находиться ваши пальцы.

Теперь наберите алфавит от A до Z и от A до Я, используя технику домашней строки.

Есть также [бесплатный онлайн учебник](#), который доступно объясняет основы навыков набора. Я призываю вас потратить всего десять минут и потренироваться.

Графическая шпаргалка по Vim

Если вы хотите знать, как какая клавиша может использоваться в *Vim*, смотри эту [графическую шпаргалку](#) по *Vim* от 'JNG'.

Хотя тут перечислены многие команды, вам нужно научиться только базовым клавишам "hjkl" которые соответственно перемещают налево, вниз, вверх, вправо. Вы узнаете об этом больше в следующей главе.

vi / vim graphical cheat sheet

Esc
normal mode

~ toggle case	! external filter	@ play macro	# prev ident	\$ eol	% goto match	^ "soft" bol	& repeat :s	* next ident	(begin sentence) end sentence	_ "soft" bol down	+ next line
· goto mark	1	2	3	4	5	6	7	8	9	0 "hard" bol	- prev line	= auto-format
Q ex mode	W next WORD	E end WORD	R replace mode	T back 'till	Y yank line	U undo line	I insert at bol	O open above	P paste before	{ begin parag.	}	end parag.
q record macro	w next word	e end word	r replace char	t 'till	y yank	u undo	i insert mode	o open below	p paste after	[misc]	misc
A append at eol	S subst line	D delete to eol	F "back" find ch	G eof/goto ln	H screen top	J join lines	K help	L screen bottom	. ex cmd line	" reg. spec	bol/goto col	
a append	s subst char	d delete	f find char	g extra cmds	h ←	j ↓	k ↑	l →	. repeat ; t/T/f/F	' goto mk. bol	\ not used!	
Z quit	X back-space	C change to eol	V visual lines	B prev WORD	N prev (find)	M screen mid'l	< un-indent reverse	> indent	? find (rev.)			
Z extra cmds	X delete char	c change	v visual mode	b prev word	n next (find)	m set mark		.	/ find			

- motion** moves the cursor, or defines the range for an operator
 - command** direct action command, if red, it enters insert mode
 - operator** requires a motion afterwards, operates between cursor & destination
 - extra** special functions, requires extra input
- q· commands with a dot need a char argument afterwards
- bol = beginning of line, eol = end of line, mk = mark, yank = copy
- words: quux(foo, bar, baz);
WORDS: quux(foo, bar, baz);

Main command line commands ('ex'):
:w (save), :q (quit), :q! (quit w/o saving)
:e f (open file f),
:%s/x/y/g (replace 'x' by 'y' filewide),
:h (help in vim), :new (new file in vim),

Other important commands:
CTRL-R: redo (vim),
CTRL-F/-B: page up/down,
CTRL-E/-Y: scroll line up/down,
CTRL-V: block-visual mode (vim only)

Visual mode:
Move around and type operator to act on selected region (vim only)

- Notes:**
- (1) use "x before a yank/paste/del command to use that register ('clipboard') (x=a..z,*) (e.g.: "ay\$ to copy rest of line to reg 'a')
 - (2) type in a number before any action to repeat it that number of times (e.g.: 2p, d2w, 5i, d4j)
 - (3) duplicate operator to act on current line (dd = delete line, >> = indent line)
 - (4) ZZ to save & quit, ZQ to quit w/o saving
 - (5) zt: scroll cursor to top, zb: bottom, zz: center
 - (6) gg: top of file (vim only), gf: open file under cursor (vim only)

For a graphical vi/vim tutorial & more tips, go to www.viemu.com - home of ViEmu, vi/vim emulation for Microsoft Visual Studio

Итоги

Обратите внимание, наша эффективность в использовании Vim, прямо пропорциональна эффективности использования клавиатуры.

Внешние ссылки

- http://en.wikipedia.org/wiki/Home_row
- http://www.bigpants.ca/juggling/images/Controls_Keyboard_HomeRow.gif
- <http://www.typeonline.co.uk/lesson1.html>
- http://www.viemu.com/a_vi_vim_graphical_cheat_sheet_tutorial.html

Vim :Перемещение

Введение

После того как вы написали исходный текст, редактирование и переписывание требует большого объема перемещения между различными частями документа. Например, вы пишете рассказ и вдруг у вас появилась идея для нового поворота сюжета, но, чтобы развить этот сюжет, вам нужно вернуться в ту часть, где главный герой попадает в новый город (или нечто подобное) ... Как быстро переместиться по тексту так, чтобы вы не потеряли ход мысли?

Давайте посмотрим несколько примеров того, как это быстро сделать в *Vim*.

Примечание: Все перемещения работают из текущей позиции курсора.

- Нужно переместить курсор на следующее слово? Нажмите **w**.
- Нужно переместить курсор на следующий параграф? Нажмите **}**.
- Нужно переместить курсор на 3-й по счету символ 'h'? Нажмите **3fh**.
- Нужно переместить курсор на 35 строк вниз? Нажмите **35j**.
- После одного из выше перечисленных перемещений, хотите перейти обратно на прежнее место? Нажмите **ctrl-o**.

Хотите узнать, как все это работает? Давайте разберем подробнее.

Сперва, откроем файл с именем `chandrayaan.txt` и наберем следующий текст ([взято из Wikipedia](#)):

```
Chandrayaan-1 is India's first mission to the moon. Launched by India's national space agency the Indian Space Research Organisation (ISRO). The unmanned lunar exploration mission includes a lunar orbiter and an impactor. The spacecraft was launched by a modified version of the PSLV XL on 22 October 2008 from Satish Dhawan Space Centre, Sriharikota, Andhra Pradesh at 06:23 IST (00:52 UTC). The vehicle was successfully inserted into lunar orbit on 8 November 2008. The Moon Impact Probe was successfully impacted at the lunar south pole at 20:31 hours on 14 November 2008.
```

```
The remote sensing satellite had a mass of 1,380 kilograms (3,042 lb) at launch and 675 kilograms (1,488 lb) at lunar orbit and carries high resolution remote sensing equipment for visible, near infrared, and soft and hard X-ray frequencies. Over a two-year period, it is intended to survey the lunar surface to produce a complete map of its chemical characteristics and 3-dimensional topography. The polar regions are of special interest, as they might contain ice. The lunar mission carries five ISRO payloads and six payloads from other international space agencies including NASA, ESA, and the Bulgarian Aerospace Agency, which were carried free of cost.
```

Перемещение курсора, способ Vim

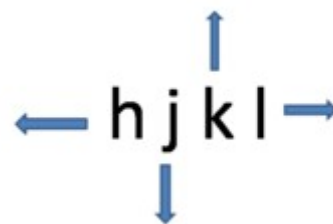
Основные клавиши, которые вы должны использовать, это клавиши "**hjkl**". Эти 4 кнопки соответствуют: влево, вниз, вверх и вправо соответственно. Обратите внимание, эти клавиши расположены прямо под правой рукой, когда руки размещены на домашней строке.

Но почему бы не использовать клавиши со стрелками? Проблема в том, что они расположены в отдельном месте на клавиатуре, и это требует столько же движений руки, как и использование мыши.

Запомните, пальцы правой руки должны всегда быть на клавишах **jkl** (и большой палец на пробеле). Теперь давайте посмотрим, как использовать эти 4 клавиши:

Использование **h,j,k,l**, вместо клавиш со стрелками

h	Вы должны переместить ваш указательный палец (который находится на 'j'), влево чтобы нажать на 'h'. Это самая левая клавиша и означает идти налево.
----------	---



j	Клавиша 'j' перемещает в низ.
k	Направленная вверх клавиша 'k' перемещает вверх.
l	Клавиша 'l' перемещает в право.

Обратите внимание, что мы можем повторять операцию, используя в префиксе количество. Например, `2j` повторит операцию `j` 2 раза.

Откройте текстовый файл `chandrayaan.txt` и начните тренироваться с этими клавишами:

- Установите ваш курсор на первом символе 'C' в документе.
- Нажмите `2j` и курсор должен пропустить текущую строку, еще строку и перейти на вторую строку т.е. на второй абзац.
- Нажмите `2k`, чтобы вернуться назад, где мы были. Или, как альтернатива, нажмите `ctrl-o` для перехода назад.
- Нажмите `5l` для движения на 5 символов в право.
- Нажмите `5h` для движения влево на 5 символов. Или, как альтернатива, нажмите `ctrl-o` для возврата назад.

Сделайте привычкой использование клавиш "hjkl" вместо клавиш со стрелками. После нескольких попыток вы заметите, насколько быстрее вы можете использовать эти клавиши.

Аналогично, есть более простые клавиши, которые заменяют следующие специальные движения. Обратите внимание, что это опять-таки имеет целью снизить количество движений рук. В этих конкретных случаях люди склонны к поиску и использованию таких специальных клавиш, так что мы тоже хотим избежать лишних движений рук.

Обычно	Vim
'home' клавиша перемещает в начало строки	<code>^</code> клавиша (переход в начало строки)
'end' клавиша перемещает в конец строки	<code>\$</code> клавиша (переход в конец строки)
'pgup' клавиша перемещает выше на один экран	<code>ctrl-b</code> перемещение на один экран назад ('b'ackward)
'pgdn' клавиша перемещает ниже на один экран	<code>ctrl-f</code> перемещение на один экран вперед ('f'orward)

Если вы знаете абсолютный номер строки, на которую вы хотите перейти, скажем 50, нажмите `50G` и Vim перейдет на строку 50. Если номер не определен, `G` переместит вас на последнюю строку в файле. Как теперь перейти в начало файла? Просто, нажмите `1G`. Обратите внимание как одна клавиша может делать так много.

- Переместить курсор в первую строку — нажмите `1G`.
- Переместить на 20 символов в право — нажмите `20l`.
- Переместить назад, на первый символ в строке — нажмите `^`.
- Перейти на последний символ в строке — нажмите `$`.
- Нажмите `G` для перехода на последнюю строку файла.

Что делать, если вы хотите перемещаться по тексту, который в данный момент отображается в окне?

- Нажмите `H` для перехода в верхнюю ('h'igh) позицию (первая строка в окне)
- Нажмите `M` для перехода в среднюю ('m'iddle) строку в окне
- Нажмите `L` для перехода в нижнюю ('l'ow) позицию (последняя строка, показанная на экране)

Вы должно быть уже заметили стремление пользоваться методом слепой печати и никогда не убирать руки с основного местоположения. Это будет правильно.

Слова, предложения, абзацы

Мы рассмотрели, как переходить к символам и строкам. Но мы намерены рассматривать текст как набор слов и способ их компоновки в предложения, абзацы, разделы, и так далее. Так почему бы не организовать перемещение по таким частям текста, т. е. по "текстовым объектам"?

Давайте возьмем несколько первых слов из нашего текста:

```
The polar regions are of special interest, as they might contain ice.
```

Сперва, давайте поместим курсор на первый символ нажав `^`.

```
[T]he polar regions are of special interest, as they might contain ice.
```

Примечание: мы использовали квадратные скобки чтобы отметить позицию курсора.

Нужно перейти на следующее слово ('word)? Нажмите **w**. Курсор переместится на 'p' в 'polar'.

```
The [p]olar regions are of special interest, as they might contain ice.
```

Как перейти на 2 слова вперед? Нужно добавить количество префиксом к 'w': **2w**.

```
The polar regions [a]re of special interest, as they might contain ice.
```

Аналогично, для движения к концу ('end) следующего слова, нажмите **e**.

```
The polar regions ar[e] of special interest, as they might contain ice.
```

Для перехода на слово назад ('backward), нажмите **b**. Цифра 2 в префиксе, **2b** переместит назад на 2 слова.

```
The polar [r]egions are of special interest, as they might contain ice.
```

Смотри [:help word-motions](#) для получения детальной информации.

Мы рассмотрели перемещение по символам и по словам, давайте перейдем к предложениям.

```
[C]handrayaan-1 is India's first mission to the moon. Launched by India's national space agency the Indian Space Research Organisation (ISRO). The unmanned lunar exploration mission includes a lunar orbiter and an impactor. The spacecraft was launched by a modified version of the PSLV XL on 22 October 2008 from Satish Dhawan Space Centre, Sriharikota, Andhra Pradesh at 06:23 IST (00:52 UTC). The vehicle was successfully inserted into lunar orbit on 8 November 2008. The Moon Impact Probe was successfully impacted at the lunar south pole at 20:31 hours on 14 November 2008.
```

Установите курсор на первом символе (^).

Для перемещения на следующее предложение, нажмите **)**.

```
Chandrayaan-1 is India's first mission to the moon. [L]aunched by India's national space agency the Indian Space Research Organisation (ISRO). The unmanned lunar exploration mission includes a lunar orbiter and an impactor. The spacecraft was launched by a modified version of the PSLV XL on 22 October 2008 from Satish Dhawan Space Centre, Sriharikota, Andhra Pradesh at 06:23 IST (00:52 UTC). The vehicle was successfully inserted into lunar orbit on 8 November 2008. The Moon Impact Probe was successfully impacted at the lunar south pole at 20:31 hours on 14 November 2008.
```

Разве это не круто ?

Для перемещения на предыдущее предложение, нажмите **(**.

Продолжайте, пробуйте и посмотрите, как быстро вы можете перемещаться. Опять же, вы можете использовать префикс, так **3)** переместит вперед на 3 предложения.

Теперь используйте весь текст и попробуйте перемещаться по абзацам. Нажмите **}**, чтобы перейти к следующему абзацу и **{**, чтобы переместиться к предыдущему абзацу.

Обратите внимание, что большие ('bigger') скобки для больших текстовых объектов. Если вы уже заметили это, то я вас поздравляю, вы уже начали думать, как победитель, то есть "думать как пользователь Vim (Vimmer)".

И снова повторяюсь, не пытайтесь запомнить все клавиши, постарайтесь ввести это в привычку, сделать так, чтобы ваши пальцы естественно использовали эти клавиши.

Смотри [:help cursor-motions](#) для получения детальной информации.

Создание ваших закладок

Вы пишете какой-то текст и вдруг вспоминаете, что вам необходимо обновить связанный раздел в том же документе, но вы хотите запомнить, где вы находитесь в данный момент, чтобы вернуться к этому месту позже. Что вы сделаете?

Можно будет прокрутить текст до нужного раздела, обновить его, а затем прокрутить туда, где вы были. Как много лишних движений, и мы, возможно, забудем, где мы в последний раз были.

Мы можем сделать это в *Vim* намного умнее. Переместим курсор на 5-ю строку в следующем тексте (слова Джона Леннона). Используем **ma** для создания закладки с именем 'a'. Переместим курсор туда, куда нам нужно, например **4j**.

I am eagerly awaiting my next disappointment. —Ashleigh Brilliant
Every man's memory is his private literature. —Aldous Huxley
Life is what happens to you while you're busy making other plans. —John Lennon
Life is really simple, but we insist on making it complicated. —Confucius
Do not dwell in the past, do not dream of the future, concentrate the mind on the present moment. —
Buddha
The more decisions that you are forced to make alone, the more you are aware of your freedom to choose.
—Thornton Wilder

Нажмите 'a (то есть после одной кавычки следует название отметки) и, вуаля, Vim переходит (назад) к строке, в которой была создана закладка.

Вы можете использовать любой символ алфавита (A-Za-z), чтобы назвать закладку, что означает, что вы можете иметь до 52 именованных меток для каждого файла.

Переходы назад и вперед

В различных перемещениях, изученных нами, мы часто хотели бы после перемещения возвратиться обратно на прежнее место или к следующей закладке. Для этого достаточно просто нажать `ctrl-o` для перехода на прежнее место и `ctrl-i`, чтобы перейти к следующему месту.

Части текста

Существует много способов, которыми вы можете выделить часть текста в Vim и затем выполнить над ним команду. Например, если вы хотите визуально выделить часть текста, а затем конвертировать регистр (из верхнего в нижний или из нижнего в верхний регистр), используйте клавишу `~`.

Откройте файл `dapping.txt`, который мы создали в предыдущих главах. Используя различные клавиши переместитесь на первый символ слова 'dapper' во втором абзаце. Подсказка: используйте `}, j, w`.

```
Dapping means being determined about being determined and being  
passionate about being passionate.  
Be a dapper.
```

Нажмите `v` для начала визуального режима, и нажмите `ap` для выбора, 'a' и 'p'aragraph. Нажмите `~` для переключения регистра текста. Если вы хотите отменить выделение, просто нажмите `<Esc>`.

```
Dapping means being determined about being determined and being  
passionate about being passionate.  
bE A DAPPER.
```

Другая мнемоника для операции над текстовыми объектами: `aw` - означает 'a' и 'w'ord; `a "` означает строку в кавычках (например, "это строка в кавычках"); `ab` - это 'a' и 'b'lock, что означает что-либо внутри пары скобок, и так далее.

Смотри [:help object-motions](#) и [:help text-objects](#) для подробной информации.

Итоги

Мы видели богатые возможности, которые дает нам Vim для перемещения по тексту. Не обязательно помнить каждое из этих перемещений, более важно сделать их привычкой, особенно те, которые наиболее актуальны для вас, и, когда они становятся привычкой, они уменьшают число движений вашей руки, вы становитесь быстрее, и в конечном итоге тратите больше времени на обдумывание содержания вашего документа, а не на программное обеспечение, которое вы используете, чтобы его написать.

Смотри [:help various-motions](#) а также [:help motion](#) для изучения интересных возможностей перемещений.

Внешние ссылки

<http://en.wikipedia.org/wiki/Chandrayaan-1>

Vim :Help (помощь)

Введение

Vim имеет такой разнообразный список команд, горячих клавиш, буферов, и так далее. Невозможно запомнить, как они все работают. На самом деле и не нужно помнить их все. Правильный выход в том, чтобы знать, как найти описание основного функционала в *Vim*, когда вам это понадобится.

Например, вы хотите избежать необходимости вводить длинное имя каждый раз, и тут вы вдруг вспомнили, что в *Vim* есть возможность работать с сокращениями, которая поможет вам сделать именно это, но вы не помните, как ей пользоваться. Что вы делаете?

Давайте посмотрим различные пути поиска подсказок о том, как пользоваться *Vim*.

Команда `:help`

Первым и самым важным местом, куда можно обратиться за помощью, является встроенная документация, и *Vim* имеет одно из наиболее полных руководств пользователя, виденных мной когда-либо.

В нашем случае, просто запустите команду `:help abbreviation`, вы попадете на страницу помощи по сокращениям и вы можете прочитать о том, как использовать команды `:ab` и `:iab`.

Иногда это может быть так просто, как в этом примере. Если же вы не знаете, что искать, то вы можете запустить `:help user-manual` и просматривать список содержимого всего руководства пользователя и читать главы, которые, как вы думаете, имеют отношение к тому, что вы пытаетесь сделать.

Как читать раздел `:help`

Давайте возьмем какой-нибудь образец вывода команды `:help`:

```
:ab[breviate] [<expr>] {lhs} {rhs}
  add abbreviation for {lhs} to {rhs}. If {lhs} already
  existed it is replaced with the new {rhs}. {rhs} may
  contain spaces.
  See |:map-<expr>| for the optional <expr> argument.
```

Обратите внимание, что способ оформления помощи в *Vim* является стандартным, чтобы сделать использование подсказки простым для нас, и мы могли посмотреть те части, которые нужны нам, вместо того, чтобы вычитывать всю команду.

Первая строка объясняет синтаксис, то есть как пользоваться этой командой.

Квадратные скобки в `:ab[breviate]` показывают, что оставшаяся часть от полного имени команды является не обязательной. Вы должны набрать минимум `:ab`, чтобы *Vim* распознал команду. Вы также можете использовать `:abb` или `:abbr` или `:abbre` и так далее до полного имени `:abbreviate`. Большинство людей склонны использовать сокращенные формы.

Квадратные скобки в `[<expr>]` снова показывают, что 'expression' не обязательный параметр.

Фигурные скобки в `{lhs} {rhs}` указывают, что это место для заполнения фактическими аргументами, которые подставляются. Это сокращения от 'left hand side' (левая рука) и 'right hand side' (правая рука), соответственно.

В абзаце, начинающемся со следующей строки с отступом, кратко объясняется, что эта команда делает.

Обратите внимание на второй абзац, который указывает вам на дополнительную информацию. Вы можете установить курсор на текст между двумя вертикальными символами и нажать `ctrl-]`, чтобы перейти по ссылке на следующий раздел `:help`. Чтобы вернуться назад, нажмите `ctrl-o`.

Команда `:helpgrep`

Если вы не знаете название темы, вы можете произвести поиск по всей документации для фразы, используя `:helpgrep`. Предположим, вы хотите найти слово `beginning`, просто запустите `:helpgrep beginning`.

Вы можете использовать `:cnext` и `:cprev` для перемещения к следующей или предыдущей части документации, где встречается эта фраза. Используйте `:clist` для просмотра полного списка всех мест, где встречается искомая фраза.

Быстрая подсказка

Скопируйте следующий текст в *Vim* и выполните его:

```
:let &keywordprg=':help'
```

Теперь, установите курсор в любом месте на ключевом слове и просто нажмите **K**. Вы перейдете на страницу помощи по этому слову. Эта ссылка позволяет избежать необходимости вводить `:help` и ключевое слово.

Форум и IRC

Если вы все еще не в состоянии понять, что вам нужно делать, то тогда вам лучше всего обратиться к другим пользователям *Vim*, чтобы они помогли вам. Не волнуйтесь, это на самом деле очень просто, и удивительно то, что другие *Vimmers* готовы помочь вам.

Первый шаг заключается в поиске списка рассылки *Vim* и поиске по нему, может уже кто-то ответил на ваш вопрос. Просто зайдите на [страницу поиска Vim](#), а затем введите ключевые слова вашего запроса. В большинстве случаев многие общие вопросы будут уже освещены, так как в этой рассылке высокая активность участников, т.е. много-много людей задают вопросы и дают ответы в этой группе.

Если вы не можете найти подходящий ответ, вы можете зайти на форум *Vim IRC*. Откройте IRC приложение, к примеру [XChat](#) (доступно для Windows, Linux, BSD) или [Colloquy](#) (for Mac OS X), подключитесь к сети "FreeNode", войдите в канал #vim, правильно задайте свой вопрос и ждите ответа. Возможно пройдет не одна минута пока не последует ответ.

Если никто не отвечает на ваш вопрос, наверное, они все заняты, поэтому повторите попытку позже или попробуйте перефразировать ваш вопрос так, чтобы он стал легким для того, кто хочет помочь вам. В противном случае, отправьте сообщение в список рассылки, упомянутый выше.

Итоги

Существует большое количество информации о том, как что-то сделать в *Vim*, и многие *Vimmers* с радостью помогут вам. Сообщество *Vim* является одним из самых сильных сторон редактора *Vim*, поэтому не забывайте использовать ресурсы и присоединяйтесь к растущему сообществу.

Истинное наслаждение в поиске знания, а не в самом знании. -- Isaac Asimov

Внешние ссылки

http://tech.groups.yahoo.com/group/vim/msearch_adv

<http://www.silverex.org/download/>

<http://colloquy.info/>

Vim :Основы редактирования

Введение

Давайте изучим основные команды в *Vim* для чтения/создания файлов, cut/copy/paste, undo/ redo и поиска.

Чтение и создание файлов

Буферы

Когда вы редактируете файл, то *Vim* переносит текст из файла на жестком диске в оперативную память компьютера.

Это означает, что копия файла сохраняется в памяти компьютера, и все ваши изменения происходят в памяти компьютера, и сразу же отображаются. После того, как вы закончите редактирование файлов, вы можете сохранить файл, что означает, что *Vim* запишет текст из памяти компьютера на жесткий диск. Память компьютера, используемая для временного хранения текста, называется «буфер». Заметим, что эта же концепция является причиной, почему у нас есть пункт меню "save" во всех редакторах или текстовых процессорах, которыми мы пользуемся.

Теперь откроем *Vim*, запишем слова Hello World и сохраним это как файл hello.txt. Если вам нужно напомнить, как это сделать, пожалуйста пересмотрите главу Первые шаги.

Swap

Теперь вы можете видеть, что в том же каталоге был создан другой файл с примерным названием `hello.txt.swp`. Выполните следующую команду, чтобы узнать точное имя файла:

```
:swapname
```

Что это за файл? *Vim* делает копию буфера в файл, который регулярно сохраняется на жестком диске, так что если что-то пойдет не так (в случае выключения компьютера или вылета *Vim*), у вас будет сохраненная версия правок, созданных вами после последнего сохранения в оригинальный файл. Этот файл называется "swap file", поскольку *Vim* сохраняет содержимое буфера памяти компьютера на жесткий диск (свопирует). Смотри [:help swap-file](#) для подробной информации.

Сохранение моего файла

Теперь, когда файл был загружен, давайте сделаем незначительное редактирование. Нажмите клавишу `~` для изменения регистра символа, на котором стоит курсор. Обратите внимание, что *Vim* теперь пометил файл как измененный (например, в графической версии *Vim* появляется знак `+` в строке заголовка). Вы можете открыть этот файл в другом редакторе и проверить, что эти изменения еще не сохранены. *Vim* пока изменил только буфер и не сохранил эти исправления на жесткий диск.

Вы можете записать буфер в оригинальный файл на жестком диске запусив:

```
:write
```

Примечание: Для простоты сохранения, добавьте следующую строку в ваш `vimrc` файл:

```
" To save, ctrl-s.  
nmap <c-s> :w<CR>  
imap <c-s> <Esc>:w<CR>a
```

Теперь вы можете простым нажатием комбинации `ctrl-s` сохранить файл.

Работа в моем каталоге

Vim запускается, используя ваш домашний каталог как каталог по умолчанию и все операции будут происходить в этом каталоге.

Для открытия файлов, расположенных в других каталогах, вы можете использовать полный или относительный путь к ним:

```
:e ../tmp/test.txt  
:e C:\\shopping\\monday.txt
```

Или вы можете переключить *Vim* на другой каталог:

```
:cd ../tmp
```

`:cd` это сокращение от `'c'hange 'd'irectory` (изменить директорию).

Чтобы посмотреть рабочий каталог, в котором *Vim* ищет файлы, выполните:

```
:pwd
```

`:pwd` это сокращение от слов `'p'rint 'w'orking 'd'irectory`.

Cut, Copy u Paste

Как сказал Sean Connery в фильме ['Finding Forrester'](#):

Не думайте — пишите. Вы должны писать сначала от вашего сердца. А переписывать от головы. Первый ключ к писательству... пишите, не думайте!

Когда мы переписываем, мы часто изменяем порядок абзацев или предложений, т.е. мы должны иметь возможность вырезать/копировать/вставлять текст. В *Vim* мы используем несколько иную терминологию:

desktop world	vim world	operation
cut	delete	d

copy	yank	y
paste	paste	p

В нормальной терминологии дестоба, вырезать ('cut'ing) - это удалить текст и поместить его в буфер обмена. В *Vim* эта операция означает, что текст из буфера файла удаляется и сохраняется в 'регистре' (часть памяти компьютера). Поскольку мы можем выбрать регистр, где мы будем хранить текст, эта операция называется операцией удаления ("delete").

Аналогично, в нормальной терминологии десктопа, копировать ('copy') текст означает копирование текста в буфер обмена. *Vim* делает то же самое Дергая ("yanks") текст и размещая текст в регистре.

Вставить ("Paste") имеет одинаковый смысл в обоих терминологиях.

Мы увидели, как вы можете производить операции cut/copy/paste в *Vim*. Но как вы определите, с каким текстом выполнять эти операции? Хорошо, что мы уже изучили в предыдущем разделе текстовые объекты.

Сочетание операции и текстовых объектов означает, что мы имеем бесчисленное количество способов управления текстом. Давайте посмотрим несколько примеров.

Напишем этот текст в *Vim* (точно так, как показано):

```
This is the rthe first paragraph.
This is the second line.

This is the second paragraph.
```

Разместите курсор в верхнюю левую позицию, нажав **I** и **G**, это переместит нас в первую строку и первый столбец соответственно.

Давайте рассмотрим первый случай: Мы напечатали лишний 'r', который мы должны удалить. Нажмите **3w** для перемещения на 3 слова. Теперь, нам нужно удалить один символ в текущей позиции курсора.

Примечание: есть два способа сделать это:

Операция	Текстовый блок/Перемещение
Удаление - Delete	Один символ в текущей позиции курсора
d	I (L в нижнем регистре)

Таким образом, мы должны просто нажать **dl** и мы удаляем один символ! Обратите внимание, что мы можем использовать **I**, хотя это перемещение.

Теперь мы замечаем, что это слово является лишним, потому что мы два раза ввели "the". Давайте подумайте о том, какую быструю комбинацию клавиш нам использовать, чтобы это исправить? Возьмите паузу, чтобы подумать и самим это понять. Если не додумались, то читайте ниже.

Операция	Текстовый блок/Перемещение
Удаление - Delete	Слово - Word
d	w

Итак, нажав **dw**, вы удалите слово. Ура! Так просто и так красиво. Вся прелесть в том, что такие простые комбинации можно комбинировать для получения большого спектра возможностей.

Как мы можем сделать ту же операцию для строки? Строки - это отдельный случай в *Vim*, потому что строки, как правило, это отдельные мысли в нашем тексте. Коротко, если вы повторите имя операции дважды, она будет произведена над строкой. Так, **dd** удалит текущую строку а **yy** скопирует текущую строку.

Наш пример текста в *Vim* должен выглядеть примерно так:

```
This is the first paragraph.
This is the second line.
This is the second paragraph.
```

Перейдите на вторую линию, нажав **j**. Теперь нажмите **dd** и строка должна быть удалена. Теперь вы должны увидеть:

```
This is the first paragraph.
```

```
This is the second paragraph.
```

Давайте посмотрим далее: как мы выдернем (удалим) текущий абзац?

Операция	Текстовый блок/Перемещение
Выдернуть - Yank	Текущий абзац
<code>y</code>	<code>ap</code>

Таким образом, `yap` скопирует текущий абзац. Теперь, когда мы скопировали текст, как мы можем вставить его? Просто нажав `'p'`.

Теперь мы увидим:

```
This is the first paragraph.  
This is the first paragraph.  
  
This is the second paragraph.
```

Обратите внимание, что пустая строка также будет скопирована, когда мы сделаем `yap`, так как `p` добавляет дополнительные пустые строки.

Существуют два вида вставить (`paste`), это аналогично двум видам вставок (`inserts`), которые мы рассмотрели раньше:

<code>p</code> (нижний регистр)	вставить после текущей позиции курсора
<code>P</code> (верхний регистр)	вставить до текущей позиции курсора

Поняв идею, далее мы можем комбинировать их в более эффективные методы.

Как поменять два символа? Нажмите `xp`.

- `x` → удалить один символ в текущей позиции курсора
- `p` → вставить после текущей позиции курсора

Как поменять два слова? Нажмите `dwwP`.

- `d` → удалить
- `w` → одно слово
- `w` → перейти к следующему слову
- `P` → вставить до текущей позиции курсора

Важно понять, что не нужно заучивать эти операции. Комбинация этих операций и текстовых блоков/перемещений должны быть автоматическими для ваших пальцев, без необходимости прикладывать умственные усилия. Это произойдет, когда войдет в привычку.

Маркировка вашей территории

Вы пишете, и вдруг понимаете, что вы должны изменить предложение в предыдущем абзаце, чтобы подкрепить то, что вы пишете в этом разделе. Проблема в том, что вы должны запомнить, где вы находитесь сейчас, что бы вы смогли вернуться к нему позже. Не запомнит ли *Vim* это для меня? Это можно сделать с помощью закладок.

Вы можете создать закладку нажав `m` и затем имя закладки, которое должно состоять из простых символов `a-zA-Z`. Например, нажатие `ma` создаст закладку с именем `'a'`.

Нажмите `'a` и курсор вернется на строку с закладкой. Нажатием `'a` вы будете переведены точно на строку и колонку закладки.

Самое приятное то, что вы можете переходить к этой позиции с помощью этих закладок любое время после этого.

Смотрите [:help mark-motions](#) для более подробной информации.

Машина времени с использованием `undo/redo`

Предположим, что вы переписывали абзац, но в итоге вы запутались в том, что вы пытались переписать, и вы хотите отменить исправления и вернуться к написанному ранее. Здесь мы можем применить `"undo"`, чтобы отменить сделанное. Если же позже мы захотим вернуться к тому, что мы имеем сейчас, мы можем

использовать "redo", чтобы мы вернули сделанное. Учтите, что изменения касаются изменений в тексте и не учитывают движение курсора и другие вещи, не относящиеся к тексту.

Предположим вы имеете текст:

```
I have coined a phrase for myself - 'CUT to the G':  
1. Concentrate  
2. Understand  
3. Think  
4. Get Things Done
```

Step 4 is eventually what gets you moving, but Steps 2 and 3 are equally important. As Abraham Lincoln once said "If I had eight hours to chop down a tree, I'd spend six hours sharpening my axe." And to get to this stage, you need to do Step 1 which boils down to one thing - It's all in the mind. That's why it's so hard. (Abraham Lincoln однажды сказал "Если бы я имел восемь часов, чтобы срубить дерево, я бы потратил шесть часов точа свой топор." И, чтобы добраться до этой стадии, что вам нужно сделать шаг 1, который сводится к одному - Это все в голове. Вот почему это так трудно.)

Теперь давайте начнем редактирование с первой строки:

1. Нажмите **S** для замены ('s'ubstitute) всей строки.
2. Наберите текст After much thought, I have coined a new phrase to help me solidify my approach:.
3. Нажмите `<esc>`.

Теперь подумайте о том изменения, которые мы только что сделали. Есть предложения лучше? Хм, предыдущий текст был лучше? Как мы можем его вернуть?

Нажмите **u** для отмены (undo) последних изменений и посмотрите, что было до этого. Вы теперь видите текст I have coined a phrase for myself - 'CUT to the G':. Для восстановления последних изменений нажмите `ctrl-r` и теперь вы увидите строку After much thought, I have coined a new phrase to help me solidify my approach:.

Важно отметить, что *Vim* ведет неограниченную историю undo/redo, но, как правило, она ограничена настройками undolevels в *Vim* и объемом памяти в вашем компьютере.

Теперь, давайте посмотрим примеры, которые действительно демонстрируют большую функциональность undo/redo в *Vim*, таким возможностям другие редакторы будут завидовать: *Vim* это не только ваш редактор, он также выступает в качестве машины времени.

Например,

```
:earlier 4m
```

вы вернетесь на 4 минуты, т.е. состояние текста которое было 4 минут назад "earlier".

Сила в том, что *Vim* превосходно все отменяет и повторяет. Например, если вы внесли изменения, а затем отменили его, а затем продолжили редактирование, то изменения на самом деле невозможно извлечь с помощью простого **u** снова. Но это возможно в *Vim* с использованием команды `:earlier`.

Вы также можете двинуться вперед во времени:

```
:later 45s
```

интервал составит 45 секунд.

Или, если вы хотите более простой подход для отмены 5 внесенных изменений:

```
:undo 5
```

Вы можете просмотреть дерево undo используя:

```
:undolist
```

Смотри `:help :undolist` для объяснения работы этой команды.

Смотри `:help undo-redo` и `:help usr_32.txt` для подробной информации.

Мощная поисковая машина, но не dotcom

Vim имеет мощный встроенный поисковый механизм, который можно использовать, чтобы найти именно то, что вы ищете. Нужно немного привыкнуть к силе, которую он предоставляет, поэтому давайте начнем.

Давайте вернемся к нашему знакомому примеру:

I have coined a phrase for myself - 'CUT to the G':

1. Concentrate
2. Understand
3. Think
4. Get Things Done

Step 4 is eventually what gets you moving, but Steps 2 and 3 are equally important. As Abraham Lincoln once said "If I had eight hours to chop down a tree, I'd spend six hours sharpening my axe." And to get to this stage, you need to do Step 1 which boils down to one thing - It's all in the mind. That's why it's so hard.

Предположим мы хотим найти слово "Step". В нормальном режиме, наберем /Step<cr> (т.е. /Step и затем клавишу enter). Мы перейдем на первое вхождение этого слова. Нажмите **n** и вы перейдете в следующему ('n'ext) вхождению, а **N** перейдет к следующему слову в обратном направлении, т.е. к предыдущему вхождению слова.

Что, если вы знаете только часть фразы или не знаете точного написания? Не было бы полезно, если бы *Vim* начал поиск при вводе поисковой фразы? Вы можете включить это командой:

```
set incsearch
```

Вы также можете сказать *Vim* игнорировать регистр (верхний или нижний регистр) текста, который вы ищете:

```
set ignorecase
```

Или вы можете использовать:

```
set smartcase
```

Если у вас включен smartcase, то:

- Если вы ищете /step, т.е. текст введен в нижнем регистре, то *Vim* будет искать любую комбинацию верхнего и нижнего регистра текста. Например, это будет соответствовать всем четырем следующим вариантам - "Step", "Stephen", "stepbrother", "misstep".
- Если вы ищете /Step т.е. текст, введенный вами, содержит символы в верхнем регистре, то поиск будет только для текста в соответствующем регистре. Например, мы найдем "Step" и "Stephen", но не "stepbrother" или "misstep".

Примечание: Я рекомендую вам вставить эти две строки в ваш файл `vimrc` (о нем будет сказано позже, а пока посмотрите [:help vimrc-intro](#) для ознакомления), чтобы эти опции были включены по умолчанию.

Теперь, когда мы поняли основы поиска, давайте рассмотрим реальную силу поиска. Прежде всего следует отметить, что то, что вы даете *Vim*, может быть не только простыми фразами, но и может быть "выражением". Выражение позволяет определить «тип» текста для поиска, а не только точный текст для поиска.

Например, вы увидели, что /step найдет нам steps, а также step и даже footstep, если такое слово встречается в тексте. Что делать, если вы хотите найти исключительно слово step и ничего больше, и чтобы оно не являлось частью другого слова? Для этого мы можем использовать `\<step\>`. Структуры `\<` и `\>` показывают начальную и конечную позицию слова.

А что, если вы хотите найти какое то число? Поиск с конструкцией `\d` будет искать цифру ('digit'). Но число это просто группа цифр вместе. Мы можем указать "одну или более" цифр вместе, используя `\d+`. Если мы будем искать ноль или более символов, мы можем использовать `*` вместо `+`.

Есть целый ряд таких магических комбинаций, которые мы можем использовать в нашей модели поиска.

Смотрите [:help pattern](#) для детальной информации.

Итоги

Мы рассмотрели некоторые основные команды редактирования, которые мы будем использовать регулярно в работе с *Vim*. Очень важно, что вы повторяли эти операции снова и они сделались для вас привычными. Не обязательно заучивать каждую команду или опции этих команд. Если вы знаете, как использовать команду, и знаете, как найти больше, основываясь на ней, то вы правильный Vimmer.

Теперь, пойдём далее и начнем редактирование!

Внешние ссылки

<http://www.imdb.com/title/tt0181536/>

Vim :Углубленное редактирование

Введение

Давайте, опираясь на изученное в предыдущей главе, начнем углубленное изучение приемов редактирования в *Vim*.

Просмотр файлов и чтение команд

Мы уже умеем редактировать и записывать файлы, но *Vim* намного более функционален.

Что если вы хотите открыть файл на чтение и не хотите его редактировать? Этого можно добиться запустив команду `vim -R`, которая запускает *Vim* в режиме только-чтение. Или, если вы имеете файл, уже открытый в *Vim*, вы можете запустить `:set ro` для создания буфера только для чтения. Преимущество использования этого варианта будет видно при просмотре больших файлов. Это значительно ускоряет просмотр, поскольку *Vim* не нужно беспокоиться о создании изменений.

Что, если вы хотите вставить содержание иного файла в текущий файл? Просто запустите `:r another_file.txt` и содержание файла `another_file.txt` будет "прочитано" и вставлено в текущий буфер. Это полезно во многих случаях, таких как объединение двух различных файлов или когда нужно сделать копию файла для внесения небольших изменений, и т.д.

Крутым побочным эффектом команды `:r` является то, что вы можете использовать её для чтения вывода команд, а не только файлов.

Например, установим программу `GCal` и запустим её:

```
:r !gcal -s1 -K
```

Она вставляет календарь с текущим месяцем (`!gcal`), с неделей, начинающейся с Monday (Понедельника) (`s1`), а также показывает номер недели (`K`). Текст будет выглядеть примерно так:

```
April 2007
Mo Tu We Th Fr Sa Su CW
                1 13
 2  3  4 <5> 6  7  8 14
 9 10 11 12 13 14 15 15
16 17 18 19 20 21 22 16
23 24 25 26 27 28 29 17
30                18
```

Представьте себе, что вы имеете возможность использовать внешние команды для добавления соответствующей информации в свой текст...

Регистры все запомнят

Давайте возьмем наш обычный простой текст:

```
I have coined a phrase for myself - 'CUT to the G':
```

1. Concentrate
2. Understand
3. Think
4. Get Things Done

```
Step 4 is eventually what gets you moving, but Steps 2 and 3 are equally important. As Abraham Lincoln once said "If I had eight hours to chop down a tree, I'd spend six hours sharpening my axe." And to get to this stage, you need to do Step 1 which boils down to one thing - It's all in the mind. That's why it's so hard.
```

Теперь, предположим, что вы хотите скопировать 4 пункта в другое место, например в резюме. Вы также хотите сохранить второе предложение, чтобы потом вставить его где-то. Было бы неплохо сохранить эти два отдельных фрагмента текста в двух разных местах одновременно, продолжить работу, а затем позже их вставить? Это достигается с помощью регистров, которые (опять же) часть памяти компьютера, с помощью

которых можно быстро сохранять и извлекать текст.

Например, вы можете разместить курсор на строке, содержащей текст `1. Concentrate`, и набрать `"a4yy`:

```
"a    использовать регистр с именем 'a' для того, чтобы
4     выполнить 4 раза следующую операцию
yy    скопировать строку
```

В переводе это означает "копировать следующие 4 строки в регистр с именем 'a'".

Для следующего шага вы можете визуально выделить второе предложение в последнем параграфе и, набрав `"bd`, удалить (`d'elete`) текст в регистр с именем 'b'.

Теперь, когда мы имеем копию соответствующего текста в буферах, мы можем вставить текст где требуется — просто нажав `"ap`, где `p'aste` вставляет текст из регистра с именем 'a' и соответственно `"bp` вставляет текст из регистра с именем 'b' и так далее.

Для просмотра содержимого всех регистров выполните:

```
:registers
```

Видите, как *Vim* делает простой концепцию clipboard и делает её такой мощной!

Смотри [:help registers](#) для информации о различных типах регистров в *Vim*.

Форматирование текста

Если вы хотите разместить текст по центру? Давайте, скажем, возьмем такой текст:

```
THIS IS THE HEADING
```

Нужно запустить:

```
:set textwidth=70
:center
```

Вы должны получить следующий результат:

```
THIS IS THE HEADING
```

Установка `:set textwidth=70` определяет максимальную ширину всех абзацев в 70 символов, и если вы пишете строки длиннее чем 70 символов, *Vim* автоматически перенесет слово, превышающее эту длину, на следующую строку.

Например, возьмите такой текст:

```
Step 4 is eventually what gets you moving, but Steps 2 and 3 are equally
important. As Abraham Lincoln once said "If I had eight hours to chop down a
tree, I'd spend six hours sharpening my axe." And to get to this stage, you need
to do Step 1 which boils down to one thing - It's all in the mind. That's why
it's so hard.
```

Он был написан с длиной строки 80. Мы хотим переформатировать абзац, заполнив максимально 70 колонок. Давайте выполним команды:

```
:set textwidth=70
gwap
```

Вторую команду нужно понимать как:

```
gw говорит начать ('g'o) форматирование текста и вернуться 'w'here I was
ap means 'a' 'p'aragraph
```

Ура! Текст принимает такой вид:

```
Step 4 is eventually what gets you moving, but Steps 2 and 3 are
equally important. As Abraham Lincoln once said "If I had eight hours
to chop down a tree, I'd spend six hours sharpening my axe." And to
get to this stage, you need to do Step 1 which boils down to one thing
- It's all in the mind. That's why it's so hard.
```

Смотри [:help formatting](#) и [:help formatoptions](#) для детальной информации.

Также, аналогично команде `:center`, есть команды `:left` и `:right` для выравнивания текста по левому краю и по

правому краю соответственно.

Поиск и замена

Мы рассмотрели, как найти текст. Ну, а допустим, мы хотим найти и заменить ('search and replace')? Можно использовать команду `:s`.

Например, вы имеете текст:

```
Setp 4 is eventually what gets you moving, but Setps 2 and 3 are equally
important. As Abraham Lincoln once said "If I had eight hours to chop down a
tree, I'd spend six hours sharpening my axe." And to get to this stage, you need
to do Setp 1 which boils down to one thing - It's all in the mind. That's why
it's so hard.
```

Мы хотим заменить все орфографические ошибки `setp` на `step`. Для этого выполним:

```
:s/setp/step/g
```

Эту команду можно прочитать так:

```
:s/pattern/replacement text/options
```

Мы уже видели, что шаблон может быть гибким и мощным настолько, насколько нам нужно. Шаблон замены может иметь специальный синтаксис на основе шаблона поиска. Например, если мы хотим заменить два слова, мы можем использовать:

```
:s/(bachchan)\ |(amitabh)/\2\1/g
```

Эта команда конвертирует текст из `bachchan amitabh` в `amitabh bachchan`.

Опциями можно определить, как замена будет работать. По умолчанию, поиск и замена работают только на первых вхождениях поискового шаблона в строке. Для распространения замены на все вхождения мы указываем опцию `g`, которая означает глобально ('g'lobal). Если мы хотим подтверждать каждое изменение, мы указываем опцию `c`, которая означает «подтверждать каждую замену».

Аббревиатуры

Иногда вам приходится писать какой-то текст снова и снова. Почему бы не использовать короткие комбинации клавиш? Они называются в *Vim* аббревиатурами.

Например, если вы постоянно пишете текст `Highly Amazing Corporation Pvt. Ltd.`, вы можете сделать так:

```
:iab hac Highly Amazing Corporation Pvt. Ltd.
```

Теперь, когда вы пишете текст и введете `h`, `a`, `c`, `<space>`, автоматически подставится указанный выше текст!

Запустите `:verbose abbreviate` для просмотра списка текущих аббревиатур.

Смотри `:help :ab` и `:help :unab` для получения подробной информации.

Проверка орфографии

Важной особенностью, добавленной в 7 версии *Vim*, является проверка орфографии. Она позволяет *Vim* искать орфографические ошибки в тексте, чтобы вы их исправляли.

Точнее, *Vim* будет искать слова, которые присутствуют в "списке хороших слов", то есть в файле орфографии, а после пометит флагами остальные слова, как возможно ошибочные.

Давайте откроем следующий текст:

```
Setp 4 is eventually what gets you moving, but Setps 2 and 3 are equally
important. As Abraham Lincoln once said "If I had eight hours to chop down a
tree, I'd spend six hours sharpening my axe." And to get to this stage, you need
to do Setp 1 which boils down to one thing - It's all in the mind. That's why
it's so hard.
```

Для включения проверки орфографии запустим:

```
:setlocal spell spelllang=en_us
```

Где 'en' стандарт для 'English' и 'us' стандарт для USA. Вы можете выбрать язык и локализацию в

зависимости от текста. Однако, соответствующие файлы с орфографией должны быть установлены в `$VIMRUNTIME/spell/` каталоге. Если они не установлены, *Vim* спросит, нужно ли автоматически скачать файл орфографии с сайта *Vim*.

Vim должен показать красную волнистую линию под "Setp" и аналогичными ошибками. Цвет будет зависеть от установленной цветовой темы.

Нажмите `Js` для перехода к следующему 'плохому слову', т.е., с ошибочной орфографией.

Теперь нажмите `z=`, чтобы спросить у *Vim* варианты правильных слов. Вам будет показан список на выбор. Вы можете ввести номер слова в этом списке, которое вы считаете правильным, и нажать `enter` для замены слова выбранным, либо просто нажать `enter` для отмены.

Если вы хотите увидеть оценку для каждого слова насколько хорош ('good') выбор замены слова, запустите `:set verbose=1` и затем запустите `z=`.

В нашем случае, мы можем заменить первое вхождение "Setp" на "Step", но в тексте это слово встречается неоднократно. Было бы лучше, если бы мы могли сделать эту замену во всем тексте. Для этого мы можем запустить `:spellrepa11`.

Рассмотрим текст:

```
Swaroop is a name.
```

В этом случае *Vim* пометит 'Swaroop' как неправильное слово ("bad word"). Мы знаем, что это имя и наша задача добавить это правильное слово в список правильных слов, чтобы *Vim* помечал его постоянно. Для этого мы можем выполнить `zg` и добавить слово в список.

Для изучения работы с файлом орфографии вы можете запустить `:spellinfo`.

Я нахожу проверку орфографии полезной, когда я могу включать её и выключать, чтобы она не мешала нормальному редактированию. Поэтому я добавляю следующие строки в мой `vimrc` файл, после чего я могу использовать `F4` для переключения:

```
" Spell check
function! ToggleSpell()
  if !exists("b:spell")
    setlocal spell spelllang=en_us
    let b:spell = 1
  else
    setlocal nospell
    unlet b:spell
  endif
endfunction

nmap <F4> :call ToggleSpell(<CR>
imap <F4> <Esc>:call ToggleSpell(<CR>a
```

Проверка орфографии - это обширная тема и, если вы интересуетесь реализацией проверки орфографии в *Vim*, и тем, как вы можете добавить орфографию или слова в список для вашего языка, пожалуйста, смотрите [:help spell](#).

Прямоугольное выделение

Когда мы редактируем табличные данные, мы иногда хотим копировать лишь несколько столбцов текста, а не несколько строк. Для этого мы можем использовать режим прямоугольного выделения в *Vim*, нажав `ctrl-v`.

Рассмотрим следующий простой текст:

```
1. Concentrate
2. Understand
3. Think
4. Get Things Done
```

Поместите курсор на заглавную 'C' в первой строке.

Нажмите `ctrl-v`.

Нажмите `3j` для перехода вниз на 3 строки.

Нажмите `$` для перемещения вправо в конец строки.

Нажмите **y** для копирования (yank) текста в регистр по умолчанию.
Запустите **:new** и нажмите **p**, что бы вставить прямоугольное выделение.
Новый файл будет иметь следующее содержание:

```
Concentrate
Understand
Think
Get Things Done
```

Смотри [:help ctrl-v](#) для получения детальной информации.

Удаленное редактирование файлов

Используя *Vim*, Вы можете редактировать файл удаленно, через ftp. Просто запустите команду **vim ftp://ftp.foo.com/bar** или запустите **:Nread ftp://ftp.foo.com/bar** в *Vim*.

Для этого можно использовать встроенный плагин "netrw" в *Vim*, используя который вы можете редактировать удаленно файл через scp, http, webdav и другие протоколы. Смотрите [:help netrw-urls](#) для подробной информации.

Вы можете прописать username и password в вашем файле ~/.netrc, что бы *Vim* мог автоматически логиниться за вас.

Смотрите [:help netrw-start](#) для получения подробной информации.

Итоги

Мы погрузились немного глубже в различные функции редактирования, которые предоставляет *Vim*. Это должно дать вам представление о широком спектре вещей, которые вы можете сделать. Опять же, главное - не вызубрить каждую функцию, а понять, что важно вам прямо сейчас, и сделать это привычкой, а затем изучать другие возможности/параметры/плагины, по мере необходимости.

Хорошей идеей будет просмотреть секцию "Editing Effectively" из [:help user-manual](#) и потом читать только разделы, которые вы находите интересными.

Vim : Множественность (Multiplicity)

Введение

В этой главе давайте посмотрим, как *Vim* поможет вам работать с разными частями файла, несколькими различными окнами ('windows') и различными закладками, помогающими добиться параллельной работы. Это важная часть управления и редактирования файлов.

Несколько разделов с использованием Складок (Folds)

Если вы редактируете большой документ, не будет ли намного проще, если вы сможете «закрыть» все ненужные в данный момент разделы и сфокусироваться только на одном?

Это то, что мы называем "folding" в Vim.

Давайте возьмем пример, где ваш документ структурирован так, что каждому уровню текста соответствует отступ на один уровень, к примеру следующий фрагмент текста:

```
Book I
The Shadow of the Past
    Three Rings for the Elven-kings under the sky,
    Seven for the Dwarf-lords in their halls of stone,
    Nine for Mortal Men doomed to die,
    One for the Dark Lord on his dark throne
    In the Land of Mordor where the Shadows lie.
    One Ring to rule them all, One Ring to find them,
    One Ring to bring them all and in the darkness bind them
    In the Land of Mordor where the Shadows lie.
Three is Company
    The Road goes ever on and on
    Down from the door where it began.
    Now far ahead the Road has gone,
    And I must follow, if I can,
    Pursuing it with weary feet,
    Until it joins some larger way,
    Where many paths and errands meet.
    And whither then? I cannot say.
```

Примечание: Этот текст был взят из [WikiQuote](#)

После набора этого текста выполните `:set foldmethod=indent`, установите курсор на текст, которому вы хотите задать отступ, нажмите `zc` и смотрите, как текст свернется. Нажмите `zo` для открытия свернутого текста.

Лично я предпочитаю использовать пробел для открытия и закрытия складок. Чтобы это работало, добавьте строку в ваш файл `vimrc`:

```
:nnoremap <space> za
```

Основные команды, которыми мы можем открывать и закрывать складки, это, соответственно, - `zo` и `zc`. Вы можете использовать `za` для переключения ('alternate) между открытием и закрытием складки. Вы можете сделать это еще проще, используя пробел в нормальном режиме, чтобы открыть и закрыть складку:

Складки - это сама по себе огромная тема с большим количеством способов складывания (ручной, маркер, выражение) и различными способами открытия и закрытия иерархии складок, и так далее.

Смотрите [:help folding](#) для подробной информации.

Множество буферов (Buffers)

Предположим, что вы хотите редактировать больше одного файла одновременно в *Vim*, что вам делать?

Напомню, что файлы загружаются в буфера в *Vim*. *Vim* также может загрузить несколько буферов одновременно. Итак, вы можете открыть несколько файлов одновременно и вы можете переключаться между ними.

Давайте предположим, что вы имеете два файла, `part1.txt` и `part2.txt`:

part1.txt

```
I have coined a phrase for myself - 'CUT to the G':
```

1. Concentrate
2. Understand
3. Think
4. Get Things Done

part2.txt

```
Step 4 is eventually what gets you moving, but Steps 2 and 3 are equally important. As Abraham Lincoln once said "If I had eight hours to chop down a tree, I'd spend six hours sharpening my axe." And to get to this stage, you need to do Step 1 which boils down to one thing - It's all in the mind. That's why it's so hard.
```

Теперь запустим `:e part1.txt` и потом `:e part2.txt`. Обратите внимание, что у вас теперь есть второй файл, открытый для редактирования. Как же переключиться к первому файлу? В данном случае вы можете запустить `:b 1` для переключения к буферу ('buffer) номер '1'. Вы можете также запустить `:e part1.txt` для открытия и отображения существующего буфера.

Вы можете просмотреть, какие буферы были загружены и, соответственно, какие файлы редактируются, запустив `:buffers` или, в короткой форме, `:ls`, что означает список ('list) буферов.

Буферы будут автоматически удалены, когда вы закроете *Vim*, поэтому вам не нужно делать ничего особенного, кроме как убедиться, что вы сохранили файлы. Однако, если вы действительно хотите удалить буфер, например, для освобождения памяти, вы можете использовать команду `:bd 1` для удаления ('delete) буфера ('buffer) номер '1', и так далее.

Смотрите `:help buffer-list` для изучения всех команд работы с буфером.

Множество окон

Мы увидели, как редактировать несколько файлов одновременно, но что, если мы хотим видеть два разных файла одновременно. Например, вы хотите открыть две разных главы своей книги, так чтобы вы могли писать вторую главу в соответствии с формулировками/описанием, данным в первой главе. Или вы хотите копировать/ вставить некоторые части из первого файла во второй файл.

В предыдущем разделе мы использовали "view", чтобы редактировать несколько буферов. *Vim* называет эти "views" окнами. Это использование термина "window" не нужно путать с окнами десктопных приложений, которые обычно ассоциируются с самими приложениями. Для нас окна ('windows') - это представления ('views') разных файлов.

Давайте возьмем простые файлы part1.txt и part2.txt, которые мы использовали в последнем разделе. Сперва загрузим part1.txt, используя `:e part1.txt`. Теперь давайте откроем новый буфер, разделив окно с помощью команды `:new`. Теперь вам доступно обычное редактирование в новом буфере в новом окне, за исключением того, что вы не можете сохранить текст, поскольку вы не ассоциировали имя файла с буфером. Для того, чтобы вы могли сохранить отредактированное в буфере, используйте `:w test.txt`.

Как нам теперь переключиться между двумя окнами? Используйте `ctrl-w` **<клавиша перемещения>** для переключения между окнами. Перемещайтесь клавишами `h,j,k,l` или клавишами со стрелками (в этом примере, только клавиши вверх и вниз имеют смысл). Запомните, комбинация клавиш `ctrl-w` работает с окнами ('windows').

Вы можете нажать `ctrl-w` дважды, т.е. `ctrl-w ctrl-w` для циклического переключения между открытыми окнами.

Практическая ситуация, в которой применение нескольких окон полезно, - это когда вы хотите посмотреть две разные части одного и того же файла одновременно. Просто запустите `:sp` для создания разделенного ('split) окна, а затем вы можете пролистать каждое окно до нужного места и продолжить редактирование. Так как они оба "окна" одного буфера, изменения в одном окне будут немедленно отражены в другом окне. Вы также можете использовать `ctrl-w s` вместо `:sp`.

Для создания вертикального разделения, используйте `:vsp` или `ctrl-w v`. Для закрытия окна выполните `:q`, как обычно.

Теперь, когда мы увидели, как открывать и использовать несколько окон, давайте посмотрим, как дальше работать с дисплеем.



- Предположим, у вас окно разделено на два, а вы хотите поменять окна местами, чтобы вы могли сконцентрировать свое зрение на нижней или верхней части экрана компьютера, в соответствии с вашими предпочтениями? Нажмите `ctrl-w r` для (ротации) "rotate" окон.
- Вы хотите сделать текущее окно верхним? Нажмите `ctrl-w K`.
- Хотите изменить размер окна, сделав его больше или меньше? Запустите `:resize 10` для задания размера в 10 строк, и т.д.
- Хотите сделать текущее окно максимального размера, что бы вы могли сосредоточиться на нем? Нажмите `ctrl-w _`. Думайте о подчеркивании, как признаке того, что другие окна должны быть как можно меньше.
- Хотите выровнять ('equal') окна снова? Нажмите `ctrl-w =`.

Смотрите `:help windows` для большей информации о работе с окнами.

Множество закладок (Tabs)

Если вы используете Firefox, вы скорее всего уже использовали вкладки, которые позволяют открывать несколько веб-сайтов в одном окне Firefox, и вы можете переключаться между ними без головной боли, связанной с переключением между несколькими окнами. Так вот, вкладки работают аналогично и в Vim. Кроме того, что их называют закладками ("tab pages").



Запустите `:tabnew` для открытия новой закладки в новом буфере (аналогично `:new`). Как теперь переключиться между закладками? Нажмите `gt` для перейти ('g'o) к следующей закладке ('t'ab) и `gT` для перейти ('g'o) в противоположном направлении, т.е. на предыдущую закладку ('t'ab).

Лично я предпочитаю использовать комбинации `alt-j` и `alt-k` для тех же действий, аналогично клавишам с символами `j` и `k`, а также `ctrl-w j` и `ctrl-w k` для работы с окнами (разделенными по горизонтали). Для включения этих комбинаций, добавьте следующие строки в ваш vimrc файл:

```
" Shortcuts for moving between tabs.
" Alt-j to move to the tab to the left
```

```
noremap <A-j> gT
" Alt-k to move to the tab to the right
noremap <A-k> gt
```

Для закрытия ('close) вкладки ('tab'), запустите `:tabc` или `:q`.

Вы даже, вместо этого, можете открыть текст в новом окне или в новой вкладке. Например, `:help tabpage` откроет подсказку в горизонтально разделенном окне. Для просмотра подсказки в новой вкладке используйте `:tab help tabpage`.

Если вы хотите изменить порядок закладок, используйте `:tabmove`. Например, для перемещения текущей закладки на первую позицию, используйте `:tabmove 0` и так далее.

Смотри `:help tabpage` для подробной информации по закладкам и другим операциям, которые вы можете выполнить, таким как `:tabdo` для операции над каждой открытой закладкой, и изменения заглавия закладок (`:help setting-guitablabel`), и т.д.

Итоги

Vim имеет несколько путей для редактирования нескольких файлов одновременно — буферы, окна и закладки. Использование этих возможностей зависит от ваших предпочтений. Например, использование нескольких закладок может заменять использование нескольких окон. Важно использовать то, что для вас является наиболее удобным и комфортным.

Внешние ссылки

http://en.wikiquote.org/wiki/The_Fellowship_of_the_Ring

Vim :Управление персональной информацией

Введение

Глава об «управлении персональной информацией» (PIM) в книге о редакторе кажется странным, не правда ли? Да, есть много "профессионального ПО", которое работает с персональной информацией, так почему мы не можем использовать текстовый редактор *Vim* для этой цели?

Управление персональной информацией — это организация всей вашей «информации», такой как ваш список дел, дневник, справочные материалы (например, важные номера телефонов), блокнот и так далее.

Сложить все это в одном удобном месте, может быть, очень удобно и мы будем реализовывать это с помощью *Vim* и нескольких плагинов.

Я склонен думать, что система PIM организована лучше, чем вики. Вики это быстрый способ связать воедино различные документы, которые взаимосвязаны, но независимы сами по себе. Неудивительно, что слово 'wiki' означает "быстрый" на гавайском языке. Подумайте о сайте - есть домашняя страница, есть взаимосвязанные страницы, на которые вы видите ссылку, и каждая страница будет иметь свой собственный контент (содержание), но могут быть ссылки и на другие страницы. Разве это не легкий способ организации веб-сайтов? А что, если вы могли бы сделать то же самое для своих личных данных? См. статью LifeHack под названием "[Wikify Your Life: How to Organize Everything](#)" с прекрасными примерами того, как вы можете это сделать.

Но значит ли это, что нам потребуется специализированное ПО Wiki? Что если вы могли бы сделать то же самое в простом текстовом редакторе файлов, используя *Vim*? Давайте разберемся.

Установка Viki

Примечание: Каталог `$vimfiles` находится в `~/vim` в Linux/Mac, `C:/Documents and Settings/<your-user-name>/vimfiles` в Windows и `C:/Users/<your-user-name>/vimfiles` в Windows Vista. Смотри `:help vimfiles` для дополнительной информации.

Мы собираемся установить Viki и связанные с ним модули:

1. Скачайте [multvals.vim](#) и сохраните как `$vimfiles/plugin/multvals.vim`.
2. Скачайте [genutils.zip](#) и разархивируйте (unzip) этот файл в `$vimfiles`.
3. Скачайте [tlib.vba.gz](#), откройте его в Vim и запустите `:so %`.
4. Скачайте [Viki.vba](#). Откройте ваш файл `vimrc` и установите следующие инструкции для `Viki.vba` (для

получения более подробной информации о vimrc используйте `:help vimrc-intro`)

```
set nocompatible
filetype plugin indent on
syntax on
```

Откройте `wiki.vba` в Vim и запустите `:so %`.

Начало работы

1. Откроем графическую (GUI) версию *Vim*
2. `:e test.txt`
3. `:set filetype=viki`
4. Наберем следующий текст: `[[http://deplate.sourceforge.net/Markup.html]][Viki syntax]`
5. `:w`
6. Установим курсор на этот текст и нажмем `ctrl+enter`, или можно иначе нажать `\vf`
7. Вы должны увидеть открытый веб-браузер с загруженной страничкой. Аналогично, вы можете вписать любое имя файла (с правильным путем) — это может быть `.doc` файл или `.pdf` файл и затем вы можете нажать `ctrl+enter` для открытия файла в программах Word или в Acrobat Reader, соответственно!

Идея в том, что вы можете использовать простые текстовые файлы для хранения всех ваших мыслей вместе, и вы можете по `ctrl+enter` перейти ко всему этому.

Теперь, обратите внимание, что мы должны были ввести парные квадратные скобки, чтобы определить ссылку и слова, которые описывают связь. Это основной синтаксис языка разметки, который мы будем изучать далее.

Язык разметки

Страница [синтаксиса Viki](#) (только что открытой в веб-браузере) объясняет, как написать текст, чтобы Viki подсвечивала часть вашего текста, а также как сделать связь между страницами 'wiki' и писать Viki - ориентированные комментарии.

Изучение основ синтаксиса подсветки полезно, поскольку вы можете визуально видеть части вашего текстового файла. Например, используйте `'* List of things to do'`, чтобы сделать заголовок, а затем используйте тире, чтобы создать список:

```
* List of things to do
- Finish the blog post on Brahmagiri trek
- Fix footer bug on IONLAB website
- Buy some blank CDs
- Get motorbike serviced
```

Отключение CamelCase

Примечание: *CamelCase* — это написание длинных слов или предложений с чередованием нижнего и верхнего регистров букв.

Написание CamelCase может создавать wiki-ссылки в Viki, но лично мне это не нравится. Я предпочитаю, чтобы допускались только явные ссылки, такие как `[[CamelCase]]`, чтобы избежать ситуаций, когда я действительно использовал имя, которое использует camel case, но я не хочу, чтобы это было ссылкой (например, слово "JavaScript"). Для отключения синтаксиса camel-case добавьте следующую строку в файл `vimrc` (объяснения — в разделе о плагинах):

```
let g:vikiNameTypes = "sSeuix"
```

Getting Things Done

Одной из главных причин создания моей 'wiki' является поддержка системы "Getting Things Done".

Getting Things Done ("GTD") эта система, разрабатываемая Дэвидом Алленом (David Allen), которая помогает управлять вашим хламом (*'stuff'* — материал, хлам, чепуха, штука, штуквина) — это может быть что угодно, от ваших карьерных планов до списка хозяйственных работ, которые вам нужно сделать сегодня. Хорошее введение в GTD может быть найдено на bnet.com.

Из книги David Allen's:

"Бери все из своей головы. Принимай решения о необходимых действиях относительно очередной задачи тогда, когда она появляется, а не тогда, когда она уже готова взорваться. Разбивайте напоминания о ваших проектах и дальнейших действиях по соответствующим категориям. Держите систему актуальной, полной, и достаточно обзримой, доверяйте своему интуитивному выбору в плане того, что вы обычно делаете (и не делаете)".

Система GTD в основном заключается в организации вашей информации в виде определенных страниц/папок:

1. Корзина
2. Список проектов
3. Дальнейшие действия
4. Календарь
5. Когда-нибудь/может быть
6. Справочный материал
7. В ожидании

Я создал wiki, чтобы использовать эту систему, используя следующий метод:

1. Во первых, создайте Начальную страницу (StartPage) которая действительно будет стартовой к вашей личной системе (которую называют просто ваш wiki).
2. Затем создайте список основных разделов вашей wiki:

```
* Getting Things Done
1. [[Collect][In Basket]]
2. [[Project][Projects List]]
3. [[NextActions][Next Actions]]
4. [[Calendar]]
5. [[SomedayMaybe][Someday/Maybe]]
6. [[Reference][Reference Material]]
7. [[Waiting][Waiting For]]
```

4. Аналогично, перейдите глубже, на сколько вам нужно, например, создайте [[Reference.Career]], чтобы записать ваши карьерные планы, и [[Project.TopSecret]], чтобы собраться с мыслями о своем следующем проекте, и так далее.

5. Всегда, когда вы хотите что-то записать, используйте страницу [[Collect]], а затем планируйте, рассматривайте и только после делайте физические усилия.

6. Пройдет некоторое время, пока вы привыкнете использовать эту систему, но после вам будет удобно, вы сможете достичь ясности ума, уверенности, что вы контролируете все аспекты вашей жизни, и, самое главное, чувство направления, понимание того, каковы самые важные вещи в вашей жизни.

Обратите внимание, что мы управляем всей системой, используя только простой текст!

Итоги

Мы увидели, как *Vim* может помочь нам в создании системы управления личной информацией. Это потрясающе, нам не нужно сложных программ для такой системы, только обычные текстовые файлы и *Vim* все сделает.

Смотри статью Abhijit Nadgouda's о том, [как использовать Vim как персональную wiki для альтернативного пути достижения того же с помощью встроенных функций Vim](#).

Внешние ссылки

<http://www.lifehack.org/articles/lifehack/wikify-your-life-how-to-organize-everything.html>

http://www.vim.org/scripts/script.php?script_id=171

http://www.vim.org/scripts/script.php?script_id=197

http://www.vim.org/scripts/script.php?script_id=861

<http://deplate.sourceforge.net/Markup.html>

http://www.bnet.com/2403-13074_23-52958.html

<http://ifacethoughts.net/2008/05/02/vim-as-a-personal-wiki/>

Vim :Сценарии

Введение

Если вы хотите сделать какое-то ПО удобным для себя, скорее всего, вы будете менять различные настройки в программном обеспечении, в соответствии с вашим вкусом и потребностями. Что делать, если вы захотите большего? Например, выполнить какие-то действия в зависимости от условия: «Если такая-то версия GUI, то использовать эту цветовую схему, а если версия другая, то другую цветовую схему»? Для этого нужны "сценарии". Сценарии, в основном, подразумевают использование языка, с помощью которого можно указать условия и действия и связать их в «сценарии».

Есть два метода использования скриптов в *Vim* — использовать встроенный в *Vim* скриптовый язык, или использовать полноценный язык программирования, такой как Python или Perl, которые имеют доступ к внутренним модулям *Vim* (нужно чтобы *Vim* был скомпилирован с этими опциями).

В этой главе потребуется некоторое знание основ программирования. Если у вас нет опыта программирования, вы все равно поймете, хотя изложение покажется слишком кратким. Если вы хотите научиться программированию, пожалуйста, обратитесь к моей другой бесплатной книге - [A Byte of Python](#).

Есть два способа создания многократно функциональности в *Vim* - использование макросов и написание сценариев.

Макросы

Используя макрос, вы можете записать последовательность команд и затем повторять их в различных ситуациях.

Например, пусть вы имеете какой-то текст наподобие этого:

```
tansen is the singer
daswant is the painter
todarmal is the financial wizard
abul fazl is the historian
birbal is the wazir
```

Здесь есть много чего исправлять.

1. Изменить первый символ предложения в верхний регистр.
2. Изменить 'is' на 'was'.
3. Изменить 'the' на 'a'.
4. Завершить предложение с "in Akbar's court."

Один из способов заключается в использовании серии команд замены, типа `:s/^\w/\u\|0/`, но это потребует 4 команды замены и будет не очень хорошо, если команда замены изменит части текста, которые мы не хотим менять.

Эффективней было бы использовать макросы.

1. Установите ваш курсор на первом символе первой строки: `tansen is the singer`
2. Наберите `qa` в нормальном режиме для начала записи макроса с именем `a`.
3. Наберите `gU1` для перевода первого символа в верхний регистр.
4. Наберите `w` для перехода к следующему слову.
5. Наберите `cw` для замены слова.
6. Наберите `was`.
7. Нажмите `<Esc>`.
8. Наберите `w` для перехода к следующему слову.
9. Наберите `cw` для замены слова.
10. Наберите `a`.
11. Нажмите `<Esc>`.
12. Наберите `A` для вставки текста в конец строки.
13. Наберите `in Akbar's court`.
14. Нажмите `<Esc>`.
15. Наберите `q` для завершения записи макроса.

Это выглядит как длительная процедура, но иногда это гораздо легче, чем создать какие-то сложные команды замены!

В конце процедуры, строка должна выглядеть следующим образом:

```
Tansen was a singer in Akbar's court.
```

Великолепно. Теперь, давайте применим это к другим строкам. Просто переместите курсор на первый символ второй строки и нажмите @a. Вуаля, строка должна измениться на следующее:

```
Daswant was a painter in Akbar's court.
```

Это свидетельствует о том, что макросы могут записывать сложные операции и могут легко повторять их. Это помогает пользователю повторять сложные редактирования в нескольких местах. Это один из видов повторяющихся манипуляций, которые можно сделать над текстом. Далее мы увидим более формальные способы управления текстом.

Примечание: Если вы хотите просто повторить последнее действие, а не последовательность действий, нет необходимости использовать макросы, просто нажмите . (клавишу с точкой).

Основы написания скриптов

Vim имеет встроенный скриптовый язык, используя который вы можете написать свои собственные скрипты для принятия решений, исполнять "do", и управлять текстом.

Действия

Как вы измените тему, то есть используемый *Vim* цвет? Просто выполните:

```
:colorscheme desert
```

Здесь я использую цветовую тему 'desert', которая, по случаю, моя любимая. Вы можете увидеть другие доступные темы, набрав `:colorscheme` и затем нажав клавишу <tab> для переключения доступных тем.

Что делать, если вы хотели узнать, сколько символов в текущей строке?

```
:echo strlen(getline("."))
```

Обратите внимание на имена 'strlen' и 'getline'. Это функции ("functions"). Функции - это части скриптов, уже написанные и с именами, чтобы мы могли использовать их снова и снова. Например, функция `getline` извлекает строку, а мы указываем какие строки извлекать, . (точка) означает текущую строку.

Мы передаем результат, возвращаемый функцией `getline`, в функцию `strlen`, которая подсчитывает количество символов в тексте, а затем мы передаем результат, возвращаемый функцией `strlen`, команде `:echo`, которая просто выводит результат. Обратите внимание, как информация передается в этой команде.

Строка `strlen(getline("."))` называется выражением. Мы можем сохранять результаты такого выражения, используя переменные. Переменные делают то, что следует из их названия, — это имена, указывающие на значения, и значением может быть что угодно, то есть оно может меняться. Например, мы можем сохранить значение длины строки в переменной `len`:

```
:let len = strlen(getline("."))  
:echo "We have" len "characters in this line."
```

Когда вы запустите код из второй строки этого текста, вы получите следующий результат:

```
We have 46 characters in this line.
```

Обратите внимание, мы можем использовать переменную в другом выражении. Возможности, которые вы получаете с помощью переменных, выражений и команд, поистине безграничны.

Vim различает разные типы переменных с помощью префиксов, таких как \$ для переменных среды окружения, & для опций, и @ для регистров:

```
:echo $HOME  
:echo &filetype  
:echo @a
```

Смотри `:help function-list`, где указан большой список доступных функций.

Вы сами можете создать функцию следующим образом:

```
:function CurrentLineLength()  
:   let len = strlen(getline("."))  
:   return len
```

```
:endfunction
```

Теперь позиционируйте ваш курсор на любой строке и выполните следующую команду:

```
:echo CurrentLineLength()
```

Вы должны увидеть напечатанную цифру. Имена функций должны начинаться с символов в верхнем регистре (с заглавной буквы). Это чтобы различать встроенные функции, которые начинаются с символа в нижнем регистре, и определяемые пользователем функции, которые начинаются с заглавных букв.

Если вы хотите просто вызвать ("call") функцию для запуска, но не отображать её вывод, вы можете использовать

```
:call CurrentLineLength()
```

Ветвления

Предположим, вы хотите отобразить различные цветовые схемы в зависимости от того работает *Vim* в терминале или работает в GUI, т.е. вам нужен скрипт для принятия решения. Вы можете использовать следующее:

```
:if has("gui_running")
:   colorscheme desert
:else
:   colorscheme darkblue
:endif
```

Как это работает:

- `has()` функция, которая используется для определения, поддерживается ли данная функция в установленном на компьютере *Vim*. Смотри `:help feature-list` для просмотра того, какие возможности доступны в *Vim*.
- Оператор `if` проверяет данное условие. Если условие выполняется, мы предпринимаем определенные действия. Иначе ("else"), мы предпринимаем альтернативные действия.
- Обратите внимание, что объявление `if` должно иметь соответствующий `endif`.
- `elseif` также доступны, если вы хотите построить цепочку с несколькими условиями и действиями.

Циклы, начинающиеся с 'for' и 'while', также доступны в *Vim*:

```
:let i = 0
:while i < 5
:   echo i
:   let i += 1
:endwhile
```

Вывод:

```
0
1
2
3
4
```

Используя встроенные функции *Vim*, то же можно записать так:

```
:for i in range(5)
:   echo i
:endfor
```

- `range()` это встроенная функция, используемая для генерации диапазона чисел. Смотри `:help range()` для детального описания.
- структуры `continue` и `break` также доступны.

Структуры данных

Скрипты *Vim* также поддерживают списки и словари. Используя их, вы можете строить сложные структуры

данных и программы.

```
:let fruits = ['apple', 'mango', 'coconut']
:echo fruits[0]
    " apple
:echo len(fruits)
    " 3
:call remove(fruits, 0)
:echo fruits
    " ['mango', 'coconut']
:call sort(fruits)
:echo fruits
    " ['coconut', 'mango']
:for fruit in fruits
:  echo "I like" fruit
:endfor
    " I like coconut
    " I like mango
```

Есть много доступных функций — смотри разделы 'List manipulation' и 'Dictionary manipulation' в [:help function-list](#).

Создание скриптов в Vim

Сейчас мы напишем сценарий *Vim*, который можно загружать в *Vim* и выполнять по мере необходимости. Это отличается от создания внутренних сценариев, исполняемых сразу после написания.

Давайте решим простую задачу — перевести в верхний регистр первую букву каждого слова в выбранном диапазоне строк? Вариант использования прост - когда я пишу заголовки в тексте документа, они выглядят лучше, если они капитализированы, но я слишком ленив, чтобы делать это самому. Однако, я могу написать текст в нижнем регистре, а затем просто вызвать функцию капитализации.

Мы начнем с простого шаблона сценария. Сохраните следующий скрипт как файл [capitalize.vim](#):

```
" Vim global plugin for capitalizing first letter of each word
" in the current line.
" Last Change: 2008-11-21 Fri 08:23 AM IST
" Maintainer: www.swaroopch.com/contact/
" License: www.opensource.org/licenses/bsd-license.php
if exists("loaded_capitalize")
    finish
endif
let loaded_capitalize = 1
" TODO : The real functionality goes in here.
```

Как это работает:

- Первая строка файла должна быть комментарием, с пояснением, что содержится в файле.
- Есть 2-3 стандартных заголовка описания файла, таких как 'Последнее обновление:', который объясняет, как был изменен 'Разработчиком' старый сценарий, информация, как пользователи могут связаться с разработчиком сценария по поводу каких-либо проблем или, возможно, с благодарностью.
- Заголовок 'Лицензия:' не обязателен, но очень рекомендуется. Скрипты *Vim* или плагины, которые вы напишете, могут использоваться многими людьми, так что укажите лицензию для скрипта. Соответственно, другие люди могут улучшить вашу работу, а это будет в свою очередь, вам на пользу.
- Скрипт может быть загружен неоднократно. Например, если вы открываете два различных файла в одном *Vim* и оба они `.html` файлы, тогда *Vim* использует подсветку HTML-синтаксиса для обеих файлов. Чтобы избежать запуска одного и того же скрипта дважды и повторного рендеринга, мы используется защита, заключающаяся в проверке существования имени 'loaded_capitalize' и закрытия его, если сценарий был уже загружен.

Теперь, давайте приступим к написанию реального функционала.

Мы можем определить функцию, выполняющую преобразование — капитализировать первую букву каждого слова, поэтому мы можем назвать функцию `Capitalize()`. Поскольку функция будет работать в диапазоне строк, мы можем указать этот диапазон.

```

" Vim global plugin for capitalizing first letter of each word
" in the current line
" Last Change: 2008-11-21 Fri 08:23 AM IST
" Maintainer: www.swaroopch.com/contact/
" License: www.opensource.org/licenses/bsd-license.php

" Make sure we run only once
if exists("loaded_capitalize")
    finish
endif
let loaded_capitalize = 1
" Capitalize the first letter of each word
function Capitalize() range
    for line_number in range(a:firstline, a:lastline)
        let line_content = getline(line_number)
        " Luckily, the Vim manual had the solution already!
        " Refer ":help s%" and see 'Examples' section
let line_content = substitute(line_content, "\\w\\+", "\\u\\0", "g")
        call setline(line_number, line_content)
    endfor
endfunction

```

Как это работает:

- `a:firstline` и `a:lastline` представляют собой аргументы функции, соответственно, начала и конца диапазона строк.
- мы используем цикл `for` для обработки каждой строки в диапазоне (выборка производится `getline()`).
- мы используем функцию `substitute()` для обработки регулярных выражений поиска-и-замены в строках. Здесь мы определяем функцию для поиска слов, которая выглядит так: `\\w\\+`, что означает слово (т.е. непрерывный набор символов, которые составляют слово).

После нахождения таких слов они будут конвертированы, как указывают опции `\\u\\0` — где `\\u` означает, что первый символ, следующий за этой последовательностью, должен быть преобразован в верхний регистр, а `\\0` указывает совпадение, найденное функцией поиска `substitute()`, которое соответствует словам. Как результат, первая буква каждого слова будет преобразована в верхний регистр.

- Затем вызывается функция `setline()` для замены строки в *Vim* измененной строкой.

Для выполнения этой команды:

1. Откройте *Vim* и введите какой-то случайный текст, типа `'this is a test'`.
2. Запустите `:source capitalize.vim` - где `'source'` это файл, в котором находятся команды, которые передаются в *Vim*, как мы это делали раньше.
3. Запустите `:call Capitalize()`.
4. Строка должна теперь выглядеть так `'This Is A Test'`.

Запускать `:call Capitalize()` каждый раз может показаться довольно утомительно, так что мы можем назначить короткие клавиши, используя `leaders`:

```

" Vim global plugin for capitalizing first letter of each word
" in the current line
" Last Change: 2008-11-21 Fri 08:23 AM IST
" Maintainer: www.swaroopch.com/contact/
" License: www.opensource.org/licenses/bsd-license.php
" Make sure we run only once
if exists("loaded_capitalize")
    finish
endif
let loaded_capitalize = 1
" Refer ':help using-<Plug>'
if !hasmapto('<Plug>Capitalize')
    map <unique> <Leader>c <Plug>Capitalize
endif
noremap <unique> <script> <Plug>Capitalize <SID>Capitalize

```

```
noremap <SID>Capitalize :call <SID>Capitalize()<CR>
" Capitalize the first letter of each word
function s:Capitalize() range
  for line_number in range(a:firstline, a:lastline)
    let line_content = getline(line_number)
    " Luckily, the Vim manual had the solution already!
    " Refer ":help s%" and see 'Examples' section
    let line_content = substitute(line_content, "\\w\\|+", "\\u\\|0", "g")
    call setline(line_number, line_content)
  endfor
endfunction
```

- Мы должны изменить имя функции с простого 'Capitalize' на 's:Capitalize', чтобы показать, что функция является локальной в скрипте и не будет доступна глобально, потому что мы хотим избежать вмешательства в другие сценарии.
- Мы используем команду `map` для определения комбинации клавиш.
- Символ `<Leader>`, обычно обратная косая черта.
- Теперь мы определим комбинацию клавиш `<Leader>c` (т.е. символ `leader`, а затем символ 'c') для некоторой функциональности.
- Мы используем `<Plug>Capitalize` для описания функции `Capitalize()` внутри плагина, т.е. нашего скрипта.
- Каждый скрипт имеет ID, который показывается `<SID>`-ом. Таким образом, мы можем отобразить команду `<SID>Capitalize` для вызова локальной функции `Capitalize()`.

Итак, теперь можно повторить шаги, описанные ранее, чтобы проверить этот скрипт, но на этот раз вы можете запустить `\\c` для капитализации строк(и) вместо запуска `:call Capitalize()`.

Последний набор шагов может показаться сложным, но это просто говорит о том, что есть много возможностей в создании скриптов *Vim*, и они могут делать сложные вещи.

Если что-то пойдет не так в ваших сценариях, вы можете использовать `v:errmsg` для просмотра последних сообщений о ошибках, которые подскажут, что пошло не так.

Примечание: вы можете использовать `:help` для поиска помощи по всем вопросам, которые мы обсуждали выше, от `:help \\w` до `:help setline()`.

Использование внешних языков программирования

Многие люди не любят тратить время на изучение скриптового языка *Vim* и могут использовать языки программирования, которые они уже знают и создавать плагины для *Vim* на этих языках. Это возможно, поскольку *Vim* поддерживает скрипты, написанные на Python, Perl, Ruby и на многих других языках.

В этой главе мы рассмотрим, как можно просто подключить и использовать плагин, написанный на языке Python, но вы также просто можете использовать любой другой поддерживаемый язык.

Как уже упоминалось ранее, если вы заинтересованы в изучении языка Python, вы можете ознакомиться с моей другой бесплатной книгой «[A Byte of Python](#)», и для начала мы должны проверить, есть ли поддержка языка программирования Python в *Vim*.

```
:echo has("python")
```

Если функция вернула 1, мы можем продолжать, в противном случае вы должны установить Python на вашей машине и проверить возможность еще раз.

Предположим, что вы пишете сообщение в блоге. Блоггер обычно хочет, чтобы как можно больше людей, насколько это возможно, прочитало его/ее блог. Один из способов того, как люди находят такие статьи в блогах, так это с помощью запросов в поисковой системе. Итак, если вы собираетесь писать на какую-то тему (скажем 'C V Raman', известный индийский физик, который получил Нобелевскую премию за свою работу по рассеиванию света), вы будете использовать ключевые фразы, чтобы помочь разным людям найти ваш блог, когда они делают поиск в данном разделе. Например, если люди пытаются найти 'c v raman', они также найдут 'raman effect', так что вы можете захотеть упомянуть это в вашем блоге или статье.

Как мы найдем похожие фразы? Оказывается, что решение довольно простое, благодаря поисковику Yahoo!. Во-первых, давайте поймем, как использовать Python для доступа к текущему тексту, который мы будем использовать для создания связанных фраз.

```
" Vim plugin for looking up popular search queries related
```



```

" to the current line
" Last Updated: 2008-11-21 Fri 08:36 AM IST
" Maintainer: www.swaroopch.com/contact/
" License: www.opensource.org/licenses/bsd-license.php
" Make sure we run only once
if exists("loaded_related")
    finish
endif
let loaded_related = 1
" Look up Yahoo Search and show results to the user
function Related()
python <<EOF
import vim
print 'Length of the current line is', len(vim.current.line)
EOF
endfunction

```

Основной подход к написанию плагинов в интерпретируемых языках такой же, как и во встроенном языке сценариев.

Ключевым отличием является то, что мы должны передать интерпретатору Python код, который мы написали на языке Python. Это достигается с помощью EOF, как показано выше - весь текст от команды `python <<EOF` до команд `EOF` передается в интерпретатор Python.

Вы можете протестировать эту программу, открыв *Vim* отдельно и запустив `:source related.vim`, а затем запустив `:call Related()`. При этом должно быть выдано что-то вроде 'Length of the current line is 54'.

Теперь, давайте покажем реальную функциональность программы. Поиск в Yahoo! имеет так называемый запрос `RelatedSuggestion`, доступ к которому мы можем получить, используя службу `web`. Доступ к службе `web` можно получить, используя Python API, реализуемый Yahoo! Search [pYsearch](#):

```

" Vim plugin for looking up popular search queries related
" to the current line
" Last Updated: 2008-11-21 Fri 08:36 AM IST
" Maintainer: www.swaroopch.com/contact/
" License: www.opensource.org/licenses/bsd-license.php
" Make sure we run only once
if exists("loaded_related")
    finish
endif
let loaded_related = 1
" Look up Yahoo Search and show results to the user
function Related()
python <<EOF
import vim
from yahoo.search.web import RelatedSuggestion
search = RelatedSuggestion(app_id='vimsearch', query=vim.current.line)
results = search.parse_results()
msg = 'Related popular searches are:\n'
for i, result in enumerate(results):
    msg += '%d. %s\n' % (i + 1, result)
print msg
EOF
endfunction

```

Обратите внимание, что мы используем текущую строку в *Vim* как актуальный текст, который нас интересует, вы можете изменить текст на такой, какой вы захотите, например, текущее слово и т. д.

Мы используем класс `yahoo.search.web.RelatedSuggestion` для запроса в поисковике Yahoo! фраз, относящихся к запросу, который мы указали. Мы получим результаты, применив `parse_results()` к полученному объекту. Затем перебираем в цикле результаты и показываем их пользователю.

1. Запустите `:source related.vim`
2. Наберите текст `c v raman`.

3. Запустите `:call Related()`

4. Вывод должен показать нечто такое:

```
Related popular searches are:  
1. raman effect  
2. c v raman india  
3. raman research institute  
4. chandrasekhara venkata raman
```

Итоги

Мы показали работу скриптов, используя внутренний язык скриптов *Vim*, а также используя внешние скриптовые языки/языки программирования. Это важно поскольку функциональность *Vim* может быть расширена до бесконечности.

Смотри `:help eval`, `:help python-commands`, `:help perl-using` и `:help ruby-commands` для получения более полной информации.

Внешние ссылки

<http://developer.yahoo.com/search/web/V1/relatedSuggestion.html>

<http://pysearch.sourceforge.net>

Vim :Плагины

Введение

В предыдущей главе мы рассмотрели написание скриптов, расширяющих существующую функциональность *Vim*. Мы вызываем эти скрипты, которые расширяют и добавляют функции, как "плагины".

Существуют различные виды плагинов:

- vimrc
- глобальные плагины
- плагины filetype
- плагины подсветки синтаксиса
- плагины компиляции

Вы можете не только написать свой плагин, но и [скачать и использовать плагины](#), написанные другими.

Настройка с использованием vimrc

Когда я устанавливаю новый дистрибутив Linux или переустанавливаю Windows, первое, что я делаю после установки *Vim*, - восстанавливаю мой последний vimrc файл из архивной копии, и потом запускаю *Vim*. Почему это важно? Поскольку мой vimrc файл содержит переменные настройки/установки, я люблю подстраивать *Vim* под себя и собственного удобства.

Вы можете создать два файла для настроек *Vim* на свой вкус:

1. [vimrc](#) — для основных настроек
2. [gvimrc](#) — для специфических GUI настроек

Эти файлы хранятся в:

- %HOMEPATH%_vimrc и %HOMEPATH%_gvimrc в Windows
- \$HOME/.vimrc и \$HOME/.gvimrc на Linux/BSD/Mac OS X

Смотри [:help vimrc](#) о точном их местонахождении в вашей системе.

Файлы vimrc и gvimrc могут содержать любые команды *Vim*. В определениях следует использовать только простые установки в файлах vimrc, и расширенные опции из плагинов.

Например, вот содержание моего файла vimrc:

```
" My Vimrc file
" Maintainer: www.swaroopch.com/contact/
" Reference: Initially based on http://dev.gentoo.org/~ciaranm/docs/vim-guide/
" License: www.opensource.org/licenses/bsd-license.php
" Use Vim settings, rather than Vi settings (much better!).
" This must be first, because it changes other options as a side effect.
set nocompatible
" Enable syntax highlighting.
syntax on
" Automatically indent when adding a curly bracket, etc.
set smartindent
" Tabs should be converted to a group of 4 spaces.
" This is the official Python convention
" (http://www.python.org/dev/peps/pep-0008/)
" I didn't find a good reason to not use it everywhere.
set shiftwidth=4
set tabstop=4
set expandtab
set smarttab
" Minimal number of screen lines to keep above and below the cursor.
set scrolloff=999
" Use UTF-8.
set encoding=utf-8
" Set color scheme that I like.
```

```

if has("gui_running")
    colorscheme desert
else
    colorscheme darkblue
endif
" Status line
set laststatus=2
set statusline=
set statusline+=%-3.3n\           " buffer number
set statusline+=%f\             " filename
set statusline+=%h%m%r%w       " status flags
set statusline+=\[%{strlen(&ft)?&ft:'none'}] " file type
set statusline+=%=            " right align remainder
set statusline+=0x%-8B        " character value
set statusline+=%-14(%l,%c%V%) " line, character
set statusline+=%<%P         " file position
" Show line number, cursor position.
set ruler
" Display incomplete commands.
set showcmd
" To insert timestamp, press F3.
nmap <F3> a<C-R>=strftime("%Y-%m-%d %a %I:%M %p")<CR><Esc>
imap <F3> <C-R>=strftime("%Y-%m-%d %a %I:%M %p")<CR>
" To save, press ctrl-s.
nmap <c-s> :w<CR>
imap <c-s> <Esc>:w<CR>a
" Search as you type.
set incsearch
" Ignore case when searching.
set ignorecase
" Show autocomplete menus.
Vim en:Plugins
set wildmenu
" Show editing mode
set showmode
" Error bells are displayed visually.
set visualbell

```

Обратите внимание, что в командах отсутствует префикс в виде двоеточия. Двоеточие не является обязательным при написании скриптов в файлах, поскольку двоеточие предполагает использование команд в нормальном режиме.

Если вы хотите детально изучить использование каждого параметра, изучайте :help.

Часть моего файла gvimrc:

```

" Size of GVim window
set lines=35 columns=99
" Don't display the menu or toolbar. Just the screen.
set guioptions-=m
set guioptions-=T
" Font. Very important.
if has('win32') || has('win64')
" set guifont=Monaco:h16
" http://jeffmilner.com/index.php/2005/07/30/windows-vista-fonts-now-available/
set guifont=Consolas:h12:cANSI
elseif has('unix')
let &guifont="Monospace 10"
endif

```

Существуют различные [примеры файлов vimrc](#), вам обязательно нужно посмотреть их и узнать различные виды настроек, которые можно сделать, а затем выбрать из них те, что вам понравятся, и записать их в

собственных vimrc.

Несколько хороших, которые я нашел в прошлом:

- [vi-improved.org's vimrc](http://vi-improved.org's_vimrc)
- [Amir Salihefendic's vimrc](#)

Известный факт, что персональный файл vimrc, как правило, отражает то, как долго человек использует Vim.

Глобальные плагины

Глобальные плагины используются для реализации глобальных/основных функций.

Глобальные плагины могут быть сохранены с двух мест:

1. `$VIMRUNTIME/plugin/` стандартные плагины идущие с *Vim* при установке
2. Для установки ваших плагинов или плагинов, которые вы где то скачали, вы можете использовать ваш каталог плагинов:
 - `$HOME/.vim/plugin/` на Linux/BSD/Mac OS X
 - `%HOME%/vimfiles/plugin/` на Windows
 - Смотрите `:help runtimepath` для детальной информации о вашем каталоге плагинов.

Давайте посмотрим, как использовать плагины.

Полезно иметь плагин [highlight_current_line.vim](#) от Ansuman Mohanty, из названия которого можно понять, что он делает. Скачайте [highlight_current_line.vim](#) последней версии и положите его в ваш каталог плагинов (как упоминалось выше).

Теперь перезапустите *Vim* и откройте любой файл. Обратите внимание, что текущая строка будет подсвечена по сравнению с другими строками в файле.

Но, что если вам это не понравилось? Просто удалите файл [highlight_current_line.vim](#) и перезапустите *Vim*.

Аналогично, вы можете установить ваши [related.vim](#) или [capitalize.vim](#) из главы Скрипты в ваш каталог плагинов, и это позволит нам избежать трудностей с использованием `:source`. В конечном счете, любой плагин *Vim*, который вы напишете, должен быть в вашем каталоге плагинов `.vim/vimfiles`.

Плагины Filetype

Плагины Filetype написаны для определенных типов файлов. Например, программы, написанные на языке C, могут иметь свой собственный стиль отступов, закладок, подсветки синтаксиса и показываемых событиях о ошибках. Эти плагины не общего назначения, они работают на определенных, специфических типах файлов.

Использование плагинов filetype

Давайте возьмем плагин filetype для XML. XML это декларативный язык, который использует теги для описание структуры самого документа. Например, если у вас есть текст, как этот:

```
Iron Gods
-----
Ashok Banker's next book immediately following the Ramayana is said to
be a novel tentatively titled "Iron Gods" scheduled to be published in
2007. A contemporary novel, it is an epic hard science fiction story
about a war between the gods of different faiths. Weary of the constant
infighting between religious sects and their deities, God (aka Allah,
Yahweh, brahman, or whatever one chooses to call the Supreme Deity)
wishes to destroy creation altogether.
A representation of prophets and holy warriors led by Ganesa, the
elephant-headed Hindu deity, randomly picks a sample of mortals, five
of whom are the main protagonists of the book--an American Catholic, an
Indian Hindu, a Pakistani Muslim, a Japanese Buddhist, and a Japanese
Shinto follower. The mortal sampling, called a 'Palimpsest' is ferried
aboard a vast Dyson's Sphere artifact termed The Jewel, which is built
around the sun itself, contains retransplanted cities and landscapes
brought from multiple parallel Earths and is the size of 12,000 Earths.
```

It is also a spaceship travelling to the end of creation, where the Palimpsest is to present itself before God to plead clemency for all creation.

Meanwhile, it is upto the five protagonists, aided by Ganesa and a few concerned individuals, including Lucifer Morningstar, Ali Abu Tarab, King David and his son Solomon, and others, to bring about peace among the myriad warring faiths. The question is whether or not they can do so before the audience with God, and if they can do so peacefully--for pressure is mounting to wage one final War of Wars to end all war itself.

(Excerpt taken from http://en.wikipedia.org/w/index.php?title=Ashok_Banker&oldid=86219280 under the GNU Free Documentation License)

Он может быть записан в XML так (формат 'DocBook XML'):

```
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.5//EN"
"http://www.oasis-open.org/docbook/xml/4.5/docbookx.dtd">
<article>
  <articleinfo>
    <author><firstname>Wikipedia Contributors</firstname></author>
    <title>Iron Gods</title>
  </articleinfo>
  <para>
    Ashok Banker's next book immediately following the Ramayana is
    said to be a novel tentatively titled "Iron Gods" scheduled to be
    published in 2007. A contemporary novel, it is an epic hard
    science fiction story about a war between the gods of different
    faiths. Weary of the constant infighting between religious sects
    and their deities, God (aka Allah, Yahweh, brahman, or whatever
    one chooses to call the Supreme Deity) wishes to destroy creation
    altogether.
  </para>
  <para>
    A representation of prophets and holy warriors led by Ganesa, the
    elephant-headed Hindu deity, randomly picks a sample of mortals,
    five of whom are the main protagonists of the book--an American
    Catholic, an Indian Hindu, a Pakistani Muslim, a Japanese
    Buddhist, and a Japanese Shinto follower. The mortal sampling,
    called a 'Palimpsest' is ferried aboard a vast Dyson's Sphere
    artifact termed The Jewel, which is built around the sun itself,
    contains retransplanted cities and landscapes brought from
    multiple parallel Earths and is the size of 12,000 Earths. It is
    also a spaceship travelling to the end of creation, where the
    Palimpsest is to present itself before God to plead clemency for
    all creation.
  </para>
  <para>
    Meanwhile, it is upto the five protagonists, aided by Ganesa and a
    few concerned individuals, including Lucifer Morningstar, Ali Abu
    Tarab, King David and his son Solomon, and others, to bring about
    peace among the myriad warring faiths. The question is whether or
    not they can do so before the audience with God, and if they can
    do so peacefully--for pressure is mounting to wage one final War
    of Wars to end all war itself.
  </para>
  <sidebar>
    <para>
      (Excerpt taken from http://en.wikipedia.org/w/index.php?title=Ashok\_Banker&oldid=86219280 under the GNU Free Documentation License)
```



```
</para>
</sidebar>
</article>
```

Обратите внимание, что структура документа в версии XML более четкая. Поэтому он прост для инструментов конвертирования XML в другие типы форматов, включая PDF и печатные версии. Плохо то, что людям писать на XML более трудно. А давайте посмотрим, как `ftplugins` может помочь пользователям *Vim*, которые пишут на XML.

1. Сначала скачаем [xmledit ftplugin](#) и положим его в ваш каталог `~/.vim/ftplugin/`.
2. Добавьте следующую строку в ваш `~/.vimrc`:

```
autocmd BufNewFile,BufRead *.xml source ~/.vim/ftplugin/xml.vim
```

(Проверьте, что вы указали правильно каталог в вашей операционной системе)

Это включает плагин `ftplugin xmledit` на время открытия файла с расширением XML.

3. Откройте *Vim* и перейдите к редактированию файла с именем `test.xml`.

4. Наберите `<article`.

5. Теперь наберите завершающую `>`, и смотрите как плагин `xmledit ftplugin` автоматически добавит завершающий тег для вас. Итак, ваш документ теперь выглядит так:

```
<article></article>
```

6. Теперь наберем другой `>` и посмотрим как тег расширяется и мы можем ввести еще теги. Документ должен выглядеть так:

```
<article>
</article>
```

7. Заметьте, что курсор также переместился на один отступ, так что вы можете писать документ в четко структурированном виде, чтобы отразить структуру документа.

8. Повторяйте процесс пока вы не напишете весь документ.

Обратите внимание, как плагин `ftplugin` для XML делает простым создание XML документов. Это как раз то, для чего предназначены плагины `ftplugins`.

Написание плагина filetype

Давайте напишем свой собственный `ftplugin`.

В нашем предыдущем примере использовался плагин `xmledit.vim ftplugin` и запись в формате XML, и мы видели, что мы должны вставлять стандартный заголовок в верхней части каждого файла DocBook XML (специфичный для формата, который мы использовали). Почему бы не сделать это автоматически в *Vim*, через использование `ftplugin`?

В `xml ftplugin` нужно заложить информацию, которая будет добавляться в начало нового XML-файла:

```
<?xml version="1.0"?>
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.5//EN"
"http://www.oasis-open.org/docbook/xml/4.5/docbookx.dtd">
```

Итак, давайте создадим новый `ftplugin`, назовем его `"xmlheader.vim"` для работы только с событием `'BufNewFile'`. Добавьте следующее в ваш файл `~/.vimrc`:

```
autocmd BufNewFile *.xml source ~/.vim/ftplugin/xmlheader.vim
```

А теперь, все что нам нужно сделать в `xmlheader.vim`, это создать первую и вторую строку в файле:

```
" Vim plugin to add XML header information to a new XML file
call setline(1, '<?xml version="1.0"?>')
call setline(2, '<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML
V4.5//EN" "http://www.oasis-open.org/docbook/xml/4.5/docbookx.dtd">')
```

Теперь перезапустите *Vim*, убедитесь что файл `'test.xml'` еще не существует, и запустите `:e test.xml`.

Вы можете видеть, что заголовок уже записан!

Подсветка синтаксиса

В предыдущем разделе мы создали файл DocBook XML. Было бы полезно, если бы в нем цветом была выделена структура XML-файла для правильных тегов DocBook, чтобы понимать, что мы пишем корректно. Оказывается, что это возможно, если мы просто запустим `:set filetype=docbkxml`. И тогда Vim использует файл синтаксиса, расположенный в `$VIMRUNTIME/syntax/docbkxml.vim`.

Файл синтаксиса определяет, как части файла зависят друг от друга. Например, файл синтаксиса для XML определяет, каким тегам какие цвета соответствуют в зависимости от имени, и тому подобное.

Использование подсветки синтаксиса

Давайте посмотрим файл синтаксиса в действии. Скачаем скрипт [mkd.vim](#), который является [файлом синтаксиса для Markdown](#). Markdown - это основной формат, в котором простой текст может быть записан так, что бы потом его можно было конвертировать в HTML.

1. Откройте новый файл в Vim с именем 'test_markdown.txt'.
2. Запустите `:set syntax=mkd`
3. Наберите следующий текст в файле:

```
# Bengaluru
The name Bangalore is an anglicised version of the city's name in the Kannada language, Bengaluru.
> A popular anecdote (although one contradicted by historical
> evidence) recounts that the 11th-century Hoysala king Veera Ballala
> II, while on a hunting expedition, lost his way in the forest. Tired
> and hungry, he came across a poor old woman who served him boiled
> beans. The grateful king named the place _"benda kaal-ooru"_
> (literally, "town of boiled beans"), which was eventually
> colloquialised to "Bengaluru".
***
(This information has been retrieved from [Wikipedia](http://en.wikipedia.org/wiki/Bangalore) under the
GNU Free Documentation License.)
```

4. Обратите внимание, как различные части файла, например, заголовок и служебные слова, автоматически выделяются. Это должно, мы надеемся, помочь в создании файлов с синтаксисом Markdown.

Написание темы подсветки синтаксиса

Давайте посмотрим как написать собственный файл синтаксиса для текстового формата AmiFormat.

Подсветка синтаксиса основана на двух действиях: первое - это определение типа текстового формата искомого файла, и второе - это описание того, как его показывать.

Например, предположим, что мы хотим найти все вхождения `any word` для показа жирным шрифтом. Сперва, нам нужно сделать такой шаблон в нашем тексте и затем соединить имя этого шаблона с нужным типом показа:

```
:syntax match ourBold /<b>.*</b>/
:highlight default ourBold term=bold cterm=bold gui=bold
```

Первая строка говорит, что мы создаем новый тип синтаксиса на основании шаблона, с именем 'ourBold' и определяем шаблон на основе регулярных выражений.

Вторая строка говорит, что мы подсвечиваем синтаксис 'ourBold'. Будет использована схема по умолчанию, она может быть изменена пользователем или можно использовать другие цветовые схемы. Мы можем определить представление 'ourBold' для трех различных типов дисплеев, в которых будет запускаться Vim: для черно-белых терминалов, цветных терминалов и GUI (графической версии).

Иногда мы хотим определить некоторые задачи в нашем тексте как пункт todo и мы обычно записываем это заглавными 'TODO', но что, если мы хотим делать это постоянно?

```
:syntax keyword ourTodo TODO FIXME XXX
:hi def link ourTodo Todo
```

Сперва мы определяем ключевое слово 'ourTodo', которое нужно подсвечивать, и соединим шаблон 'ourTodo' с уже существующей группой с названием 'Todo' в Vim. Таких, уже существующих групп, которые имеют предопределенные цветовые схемы, в Vim много. Лучше всего привязать наши синтаксические стили к

существующим группам. Смотри [:help group-name](#) для списка доступных групп.

Далее, некоторые блоки кода в группе могут быть заключены в теги `[code]` .. `[/code]`, как мы можем выделить их подсветкой?

```
:syn region amiCode excludenl start=\/[code\]/ end=\/[\/code\]/
:hi def link amiCode Identifier
```

Во-первых, мы указываем, что мы определяем область текста, начальным и конечным шаблоном (который очень прост в нашем случае), а затем связываем его с уже существующим классом "Identifier".

Аналогичным образом, мы можем обработать и определить другие части текста, определенного в [описании формата AmiFormat](#), а окончательный сценарий может выглядеть следующим образом:

```
"Vim syntax file for AmiFormat
"Language: AmiFormat
"Version: 1
>Last Change: 2006-12-28 Thu
"Maintainer: www.swaroopch.com/contact/
"License: www.opensource.org/licenses/bsd-license.php
"Reference: http://orangoo.com/labs/AmiNation/AmiFormat/

"""""""""" Initial Checks """"""""""
" To be compatible with Vim 5.8. See `:help 44.12`
if version < 600
    syntax clear
elseif exists("b:current_syntax")
    " Quit when a (custom) syntax file was already loaded
    finish
endif

"""""""""" Patterns """"""""""
" Emphasis
syn match amiItalic /<i>.\{-}</i>/
syn match amiBold /<b>.\{-}</b>/
" Todo
syn keyword amiTodo TODO FIXME XXX
" Headings
syn match amiHeading /^h[1-6]\.\s\+\.\{-}$/
" Lists
syn match amiList /^\s*\*\s\+/
syn match amiList /^\s*\d\+\.\s\+/
" Classes
syn match amiClass /^\s*%(\\w\+).*/
syn match amiClass /^\s*%{.*}.*%/
" Code
syn region amiCode excludenl start=\/[code\]/ end=\/[\/code\]/
" HTML
syn region amiEscape excludenl start=\/[escape\]/ end=\/[\/escape\]/
" Link
syn match amiLink /".\{-}":(.\{-})/
" Image
syn match amiImage /!\.\{-}(.\{-})!//

"""""""""" Highlighting """"""""""
hi def amiItalic term=italic cterm=italic gui=italic
hi def amiBold term=bold cterm=bold gui=bold
hi def link amiHeading Title
hi def link amiTodo Todo
hi def link amiList PreProc
hi def link amiClass Statement
hi def link amiCode Identifier
hi def link amiEscape Comment
hi def link amiLink String
```

```

hi def link amiImage String

"""""""""" Finish """"""""""
" Set syntax name
let b:current_syntax = "amifmt"

```

Теперь, когда скрипт уже работает, я должен загрузить его в [раздел скриптов Vim](#), как я уже писал об этом! Теперь кто угодно может использовать скрипт подсветки синтаксиса AmiFormat в Vim.

Для изучения деталей о скриптах подсветки синтаксиса в Vim, прочитайте:

- [:help syntax](#)
- [:help usr_44.txt](#)
- [:help group-name](#)
- [:help pattern-overview](#)
- [:help mysyntaxfile](#)
- [:help new-filetype](#)

Примечание: Если вы хотите, перерисовать экран в случае, если файл подсветки синтаксиса неправильно работает, нажмите *Ctrl-L*.

Примечание: Возможно, вы уже догадались, что, когда мы предварительно задали *filetype*, Vim, в свою очередь, автоматически устанавливает подсветку синтаксиса для этого расширения.

Плагины компиляции

Плагины компиляции используются для компиляции программ, написанных на разных языках. Они полезны везде, где требуется преобразование из исходных текстов в другой формат, даже если вы пишете обычный текстовый файл в Markdown и хотите конвертировать текст в HTML с помощью программы преобразования.

Давайте изучим использование плагина компиляции для Python.

1. Скачаем скрипт [compiler/python.vim](#) и положим его в наш каталог `~/vim/compiler/`.
2. Вставим следующую строку в наш `~/vimrc`:

```
autocmd BufNewFile,BufRead *.py compiler python
```

3. Перезапустим Vim, откроем файл Python, скажем `test.py` и введем следующую программу:

```
#!/python
print 'Hello World'
```

4. Запустите `:make` и вы увидите успешную компиляцию.
5. Давайте намеренно введем ошибку в программу, изменив написание `'print'` в `'pritr'`:

```
pritrn 'Hello World'
```

Теперь запустим `:make` и обратим внимание, что Vim выдал ошибку и автоматически перевел курсор на строку с ошибкой!

6. Запустим `:clist` чтобы просмотреть полный список ошибок.
7. После исправления ошибки вы можете выполнить `:cnext` для перехода к следующей ошибке.

Если вы откроете скрипт [compiler/python.vim](#), который мы скачали, вы можете увидеть, что он очень простой - тут только две определенные переменные, одна - это `makeprg`, которая определяет как сделать 'make' файл, т.е. как компилировать это, и вторая - это `errorformat`, которая определяет форму вывода ошибок компилятора.

Я написал [плагин компилятор для Adobe Flex](#) с использованием тех же двух переменных.

Смотри [:help write-compiler-plugin](#) и [:help quickfix](#) для подробной информации о том, как написать свой собственный плагин компиляции.

Домашнее задание: написать глобальный плагин

В целях закрепления полученных навыков создания плагинов выполните следующее упражнение:

Напишите плагин, который удаляет повторяющиеся и избыточные пустые строки в документе.

Вы можете использовать либо язык сценариев *Vim*, или любой другой язык, поддержка которого есть у вас в *Vim*.

Если вам нужно "вдохновение", см. этот [совет Vim](#).

Вот еще одно упражнение:

Напишите скрипт, который выдает значение и похожие по смыслу слова для текущего слова под курсором.

Опять же, если вам нужно "вдохновение", см. мой [плагин lookup.vim](#).

Отключение плагинов

Предположим, что вы сочтете действия плагина *Vim* странными, и у вас появится подозрение, что причиной этого является скрипт. В таком случае вы можете задать в *Vim* избирательную инициализацию скриптов путем задания аргумента `-u` в командной строке.

Например, `vim -u NONE` запустит *Vim* без инициализации скриптов.

То есть, работает только *Vim*. Используйте `vim -u - your-minimal-initialization.vim` для выполнения только конкретных инициализаций. Эта опция полезна при отладке, если нужно выяснить, вызваны проблемы самим *Vim* или запуском плагина, и т.д.

Смотри `:help -u` и `:help starting` для детальной информации.

Итоги

Мы рассмотрели различные типы плагинов, доступные в *Vim*, как использовать такие плагины и как писать такие плагины. Теперь мы понимаем, как расширяется *Vim*, и как мы можем писать плагины для облегчения своей жизни.

Внешние ссылки

<http://www.vim.org/scripts/index.php>

<http://dotfiles.org/.vimrc>

<http://www.vi-improved.org/vimrc.php>

<http://amix.dk/vim/vimrc.html>

http://www.vim.org/scripts/script.php?script_id=1652

http://www.vim.org/scripts/script.php?script_id=301

http://www.vim.org/scripts/script.php?script_id=1242

<http://daringfireball.net/projects/markdown/>

<http://orango.com/labs/AmiNation/AmiFormat/>

<http://orango.com/labs/AmiNation/AmiFormat/online%20reference/>

http://www.vim.org/scripts/script.php?script_id=1745

http://www.vim.org/scripts/script.php?script_id=1439

http://www.vim.org/scripts/script.php?script_id=1746

http://vim.wikia.com/wiki/Remove_unwanted_empty_lines

http://www.vim.org/scripts/script.php?script_id=2001

Vim : Редактор программистов

Введение

Vim, как правило, активно используется программистами. Функциональность, простота использования и гибкость, которые дает *Vim*, делают его хорошим выбором для людей, которые пишут много кода. Это не должно показаться удивительным, так как написание кода включает в себя много редактирования.

Позвольте мне повторить, что навыки набора текста крайне важны для программиста. Если наши предыдущие обсуждения не убедили вас, надеюсь, статья Jeff Atwood с названием '[We Are Typists First, Programmers Second](#)' убедит вас.

Если вы не имеете опыта программирования, вы можете пропустить этот раздел.

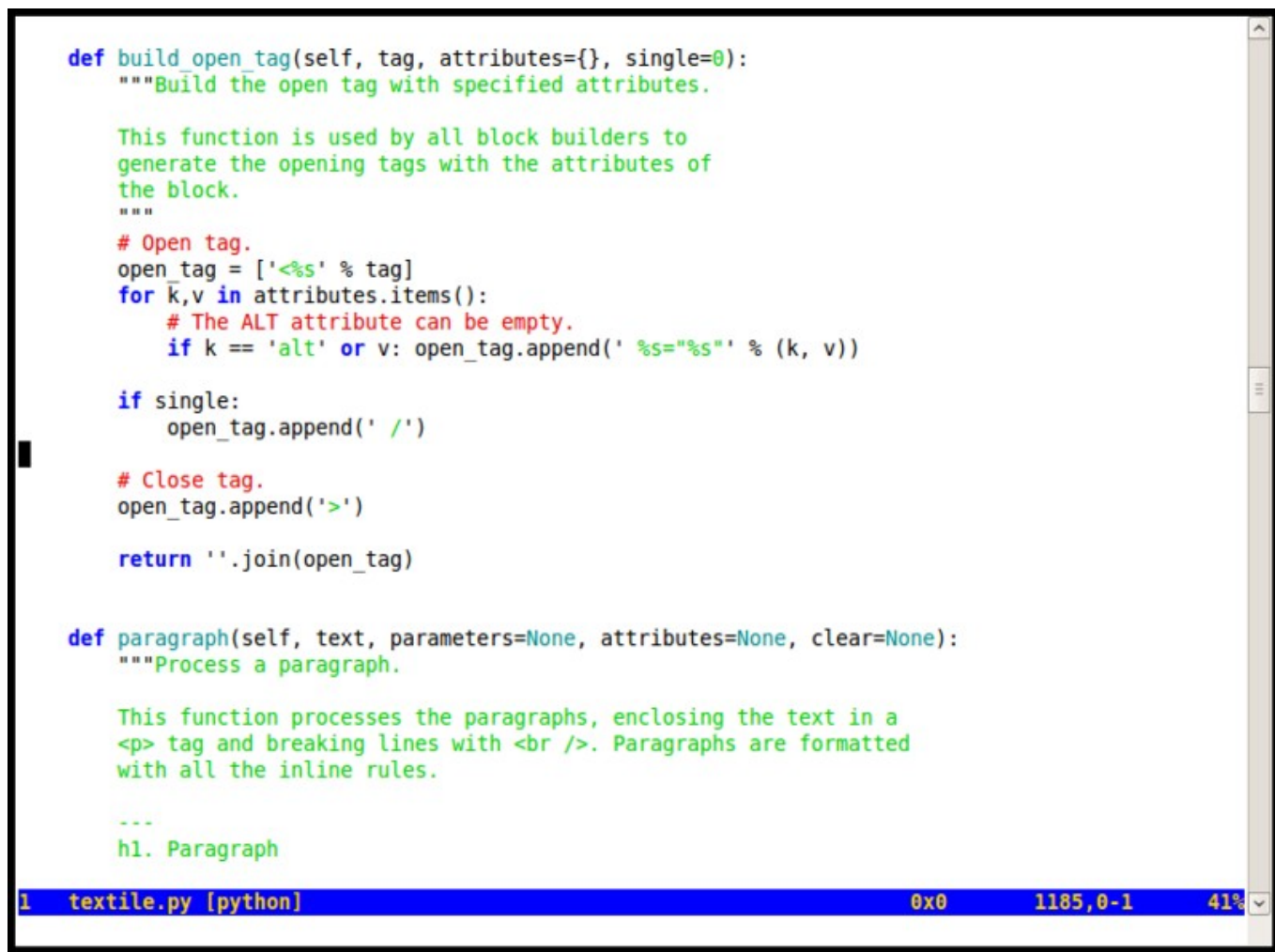
Те же, кто любит программировать, увидят, как *Vim* может помочь в написании кода.

Простые вещи

Самая простая возможность в *Vim*, которую вы можете использовать при написании кода - это использовать подсветку синтаксиса. Это позволяет вам визуализировать, т.е. "видеть" ваш код, подсветка помогает вам в быстром чтении и написании вашего кода, а также помогает избежать очевидных ошибок.

Подсветка синтаксиса

Предположим вы редактируете файл синтаксиса *Vim*, запустите `:set filetype=vim` и посмотрите как *Vim* добавит цвета. Аналогично, если вы редактируете файл на Python, запустите `:set filetype=python`.



```
def build_open_tag(self, tag, attributes={}, single=0):
    """Build the open tag with specified attributes.

    This function is used by all block builders to
    generate the opening tags with the attributes of
    the block.
    """
    # Open tag.
    open_tag = ['<%s' % tag]
    for k,v in attributes.items():
        # The ALT attribute can be empty.
        if k == 'alt' or v: open_tag.append(' %s="%s"' % (k, v))

    if single:
        open_tag.append(' /')

    # Close tag.
    open_tag.append('>')

    return ''.join(open_tag)

def paragraph(self, text, parameters=None, attributes=None, clear=None):
    """Process a paragraph.

    This function processes the paragraphs, enclosing the text in a
    <p> tag and breaking lines with <br />. Paragraphs are formatted
    with all the inline rules.

    ---
    h1. Paragraph
```

1 textile.py [python] 0x0 1185,0-1 41%

Для просмотра списка доступных типов языков, загляните в каталог `$VIMRUNTIME/syntax/`.

На заметку: Если вы хотите включить подсветку синтаксиса для любого вывода Unix оболочки, просто перенаправьте вывод через канал в *Vim*, например, так: `svn diff | vim -R -`. Обратите внимание на тире в конце, которое говорит *Vim*, что он должен читать текст со стандартного ввода.

Умные отступы

Код опытного программиста, как правило, написан с правильным отступом, что делает код информативным и структура кода более очевидна. *Vim* может помочь в создании отступов, чтобы вы могли сосредоточиться на реальном коде.

Если вы отступаете до определенной линии и хотите, чтобы следующие строки были с таким же отступом, то вы можете использовать установку `:set autoindent`.

Если вы начинаете новый блок операторов и хотите, чтобы последующие строки автоматически шли с отступом еще на один уровень, то вы можете использовать `:set smartindent`. Обратите внимание, что поведение этого параметра зависит от использования конкретного языка программирования.

Скачки

Если язык программирования, который вы выбрали, использует фигурные скобки для разграничения блоков операторов, поместите курсор на одну из фигурных скобок и нажмите клавишу % для перехода на парную фигурную скобку. Этот ключ позволяет вам быстро переключаться между началом и концом блока.

Команды оболочки

Вы можете запустить команды оболочки из *Vim*, используя команду '!':

Например, если команда `date` доступна в вашей операционной системе, запустите `!date` и вы увидите строку с текущей датой и временем.

Это удобно в ситуации, когда вы хотите проверить что-то в вашей файловой системе, например, быстро узнать, какие файлы находятся в текущем каталоге: `!ls` или `!dir`, и так далее.

Если вы хотите получить доступ ко всем возможностям оболочки, запустите `:sh`.

Мы можем использовать эти объекты для запуска внешних фильтров для редактируемого текста. Например, если вы имеете набор строк, которые вы хотите отсортировать, вы можете запустить `:%!sort`, эта команда передаст текущий текст команде `sort` в оболочке и затем вывод команды заменит текущее содержание файла.

Переходы

Есть много способов перемещаться по коду.

- Разместите ваш курсор на имени файла в коде и затем нажмите `gf` для открытия файла.
- Разместите ваш курсор на имени переменной и нажмите `gd` для перехода к локальному определению имени переменной. `gD` производит то же для глобального объявления, производя поиск с начала файла.
- Используйте `]]` для перехода к следующему вхождению `{` в первом столбце. Есть много подобных перемещений - смотри [:help object-motions](#) для подробной информации.
- Смотри [:help 29.3](#), [:help 29.4](#), и [:help 29.5](#) по использованию команд. Например, `[I` покажет все строки, содержащие ключевое слово под курсором!

Просмотр части кода

Файловая система

Используйте `:Vex` или `:Sex` для просмотра файловой системы в *Vim* и последующего открытия нужных файлов.

ctags

Мы видели, как производить простые действия в пределах одного файла, но что, если мы хотим перемещаться между различными файлами и иметь перекрестные ссылки между файлами? Тогда мы можем использовать вкладки для достижения этой цели.

Например, для просмотра файла мы можем использовать плагин [taglist.vim](#).

1. Установим программу [Exuberant ctags](#).
2. Установим плагин [taglist.vim](#). Подробное описание установки — на странице скрипта.
3. Запустите `:TlistToggle` для открытия окна `taglist`. Класно, теперь вы можете просматривать части вашей программы, такие как макросы, определения типов, переменных и функций.
4. Вы можете использовать `:tag foo` для перехода к определению `foo`.

```

class
function
ste_new
ste_repr
ste_dealloc
symtable_new
PySymtable_Build
PySymtable_Free
PySymtable_Lookup
PyST_GetScope
analyze_name
analyze_cells
check_unoptimized
update_symbols
analyze_block
symtable_analyze
symtable_warn
symtable_exit_block
symtable_enter_block
symtable_lookup
symtable_add_def
symtable_new_tmpname
symtable_visit_stmt
symtable_visit_expr
symtable_implicit_arg
symtable_visit_params
symtable_visit_arganno
symtable_visit_annotat
symtable_visit_argumen
symtable_visit_except
symtable_visit_alias
symtable_visit_compreh

```

```

if (!PyNumber_InPlaceOr(free, newfree))
    goto error;
Py_DECREF(free);
success = 1;
error:
Py_XDECREF(scopes);
Py_XDECREF(local);
Py_XDECREF(newbound);
Py_XDECREF(newglobal);
Py_XDECREF(newfree);
if (!success)
    assert(PyErr_Occurred());
return success;
}

static int
symtable_analyze(struct symtable *st)
{
    PyObject *free, *global;
    int r;

    free = PySet_New(NULL);
    if (!free)
        return 0;
    global = PySet_New(NULL);
    if (!global) {
        Py_DECREF(free);
        return 0;
    }
    r = analyze_block(st->st_top, NULL, free, global);
    Py_DECREF(free);
    Py_DECREF(global);
    return r;
}

```

Tag List [-][taglist]> 1 symtable.c [c] 0x73 761,1 47%

Taglist в действии.

5. Разместите ваш курсор на любом символе и нажмите `ctrl-]` для перехода к определению символа.
 - Нажмите `ctrl-t` для возврата к предыдущему коду, который вы читали.
6. Используйте `ctrl-w]` для перехода к определению символа в разделенном окне.
7. Используйте `:tnext`, `:tprev`, `:tfirst`, `:tlast` для движения между соответствующими тегами.

Обратите внимание, что Ctags богат поддержкой 33 языков программирования (на момент написания статьи) и может быть легко расширен на другие языки.

Смотри [:help taglist-intro](#) для подробной информации.

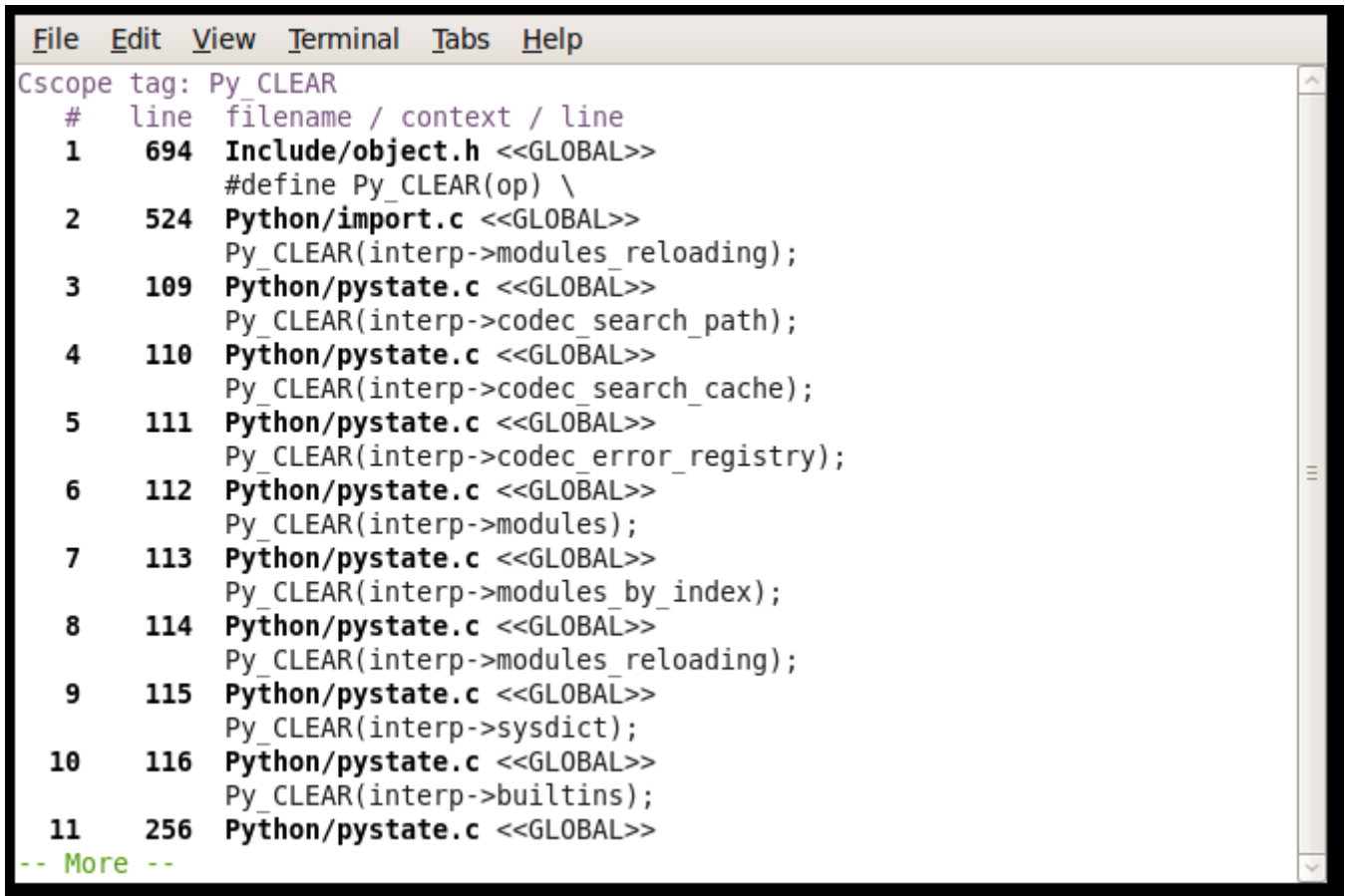
cscope

Для того, чтобы иметь возможность переходить к определениям в файлах, нам необходимо иметь такую программу как cscope. Однако, как следует из названия, эта конкретная программа работает только с программами на языке программирования Си.

1. Установите cscope. Обзор установки в [:help cscope-info](#) и [:help cscope-win32](#).
2. Скопируйте [cscope_maps.vim](#) в ваш каталог `~/vim/plugin/`.
3. Перейдите в ваш каталог исходных кодов и запустите `cscope -R -b` для построения ('b'uild) рекурсивной ('r'eursively) базы данных для всех подкаталогов.
4. Перезапустите Vim и откройте файл с исходным кодом.
5. Запустите `:cscope show` для подтверждения того, что cscope соединение создано.
6. Запустите `:cscope find symbol foo` для поиска символа foo. Вы можете сократить эту команду до `:cs f s foo`.

Вы также можете:

- Найти это определение - `:cs f g`
- Найти функции, вызываемые этой функцией - `:cs f d`
- Найти функции, вызывающие эту функцию - `:cs f c`
- Найти эту строку текста - `:cs f t`
- Найти этот шаблон egrep - `:cs f e`



```
File Edit View Terminal Tabs Help
Cscope tag: Py_CLEAR
# line filename / context / line
1 694 Include/object.h <<GLOBAL>>
  #define Py_CLEAR(op) \
2 524 Python/import.c <<GLOBAL>>
  Py_CLEAR(interp->modules_reloading);
3 109 Python/pystate.c <<GLOBAL>>
  Py_CLEAR(interp->codec_search_path);
4 110 Python/pystate.c <<GLOBAL>>
  Py_CLEAR(interp->codec_search_cache);
5 111 Python/pystate.c <<GLOBAL>>
  Py_CLEAR(interp->codec_error_registry);
6 112 Python/pystate.c <<GLOBAL>>
  Py_CLEAR(interp->modules);
7 113 Python/pystate.c <<GLOBAL>>
  Py_CLEAR(interp->modules_by_index);
8 114 Python/pystate.c <<GLOBAL>>
  Py_CLEAR(interp->modules_reloading);
9 115 Python/pystate.c <<GLOBAL>>
  Py_CLEAR(interp->sysdict);
10 116 Python/pystate.c <<GLOBAL>>
  Py_CLEAR(interp->builtins);
11 256 Python/pystate.c <<GLOBAL>>
-- More --
```

Рис. cscope в действии

Предложения по использованию cscope в Vim вы найдете, запустив [:help cscope-suggestions](#).

Также стоит установить плагин [Source Code Obedience](#), он обеспечивает использование удобных сочетаний клавиш в плагинах cscope/ctags.

Если говорить о языке программирования Си, может быть довольно удобен плагин [c.vim](#).

Компилирование

Мы уже видели в предыдущей главе описание `:make` для программ, которые мы пишем, поэтому мы не будем повторяться.

Легкое написание

Omnicompletion

Одной из наиболее востребованных функций, которая была добавлена в Vim 7, является "omnicompletion", когда текст может быть автоматически завершен в зависимости от текущего контекста. Например, если вы используете длинное имя переменной, причем используете это имя неоднократно, вы можете задать сочетание клавиш в Vim для автодополнения, и Vim будет вводить оставшуюся часть.

Vim решает эту задачу с помощью ftplugins, в частности по имени ftplugin/<language>complete.vim, например, pythoncomplete.vim.

Давайте запустим пример с простой программой на Python:

```
def hello():
    print 'hello world'
def helpme():
    print 'help yourself'
```

После ввода этой программы создайте новую строку в этом файле, наберите 'he' и нажмите `ctrl-x ctrl-o`. Вам будут показаны варианты автозавершения.

```
2 [Scratch] [-][Preview][none] 0x20 1,1 All
def hello():
    print 'hello world'

def helpme():
    print 'help yourself'

helpme(
  helpme()
  hello()

1 test.py [+]python 0x6C 7,3 All
-- Omni completion (^O^N^P) match 1 of 2
```

Omni-completion в действии

Если вы получите ошибку [E764: Option 'omnifunc' is not set](#), запустите `:runtime! autoload/pythoncomplete.vim` для загрузки плагина omnicompletion.

Чтобы не делать это каждый раз, вы можете добавить следующую строку в ваш `~/.vimrc`:

```
autocmd FileType python runtime! autoload/pythoncomplete.vim
```

Vim автоматически использует первый вариант автодополнения, вы можете перейти к следующему или предыдущему с помощью `ctrl-n` и `ctrl-p` соответственно.

Если вы хотите оборвать использование omnicompletion, просто нажмите `esc`.

Изучите `:help new-omni-completion` для подробной информации о поддерживаемых языках (C, HTML, JavaScript, PHP, Python, Ruby, SQL, XML, ...) а также о том, как создавать свои собственные сценарии omnicompletion.

Примечание: Если вам больше нравится использовать кнопки со стрелками для выбора в списке omnicompletion, смотри [Vim Tip 1228](#) как их подключить.

Я предпочитаю использовать просто `ctrl-space` вместо громоздкой комбинации `ctrl-x ctrl-o`. Для реализации этого, вставьте строку в ваш `vimrc`:

```
imap <c-space> <c-x><c-o>
```

И еще, [плагин PySmell](#) может помочь тем пользователям Vim, которые программируют на Python.

Использование фрагментов

Фрагменты кода - это небольшие кусочки кода, которые, как правило, повторяются постоянно. Как и все хорошие ленивые программисты, вы можете использовать плагин, который поможет вам их вставлять. В нашем случае мы используем потрясающий плагин SnippetsEmu.

1. Скачайте плагин [snippetsEmu](#).

2. Создайте ваш каталог `~/.vim/after/`, если он еще не существует.
3. Запустите *Vim*, вставив имя этого плагина в командную строку. Например, запустите *Vim* как `gvim snippy_bundles.vba`
4. Запустите `:source %`. Тем самым 'vimball' будет распакован и большое количество файлов будет сохранено в соответствующие каталоги.
5. Повторите то же для `snippy_plugin.vba`

Теперь, давайте научимся использовать этот плагин.

1. Откройте новый файл, скажем `test.py`.
2. Нажмите клавиши `d`, `e`, `f` и затем `<tab>`.
3. Ура! Смотрите, как `snippetsEmu` создает структуру вашей функции. Вы должны увидеть в вашем файле следующее:

```
def <{fname}>(<{args}>):
    """
    <{>
    <{args}>"""
    <{pass}>
    <{>
```

Примечание: В случае, если вы видите `def<tab>` и ничего больше, то, возможно, плагин фрагментов не загружен. Запустите `:runtime! ftplugin/python_snippets.vim` и посмотрите, поможет ли это.

4. Ваш курсор теперь находится на имени функции, т.е. `fname`.
5. Наберите имя функции, скажем, `test`.
6. Нажмите `<tab>` и курсор автоматически перейдет к аргументам. Нажмите `Tab` снова для перехода к заполнению следующего пункта.
7. Теперь введите комментарий: Скажем `Hi` (привет)
8. Нажмите `Tab` снова и введите `'Hello World'`
9. Нажмите `tab`
10. Ваша программа готова!

Теперь вы должны увидеть вот это:

```
def test():
    """
    Just say Hi
    """
    print 'Hello World'
```

Самое приятное то, что `SnippetsEmu` создает стандартное форматирование, которому нужно следовать, и что ничего в команде не будет "забыто".

Создание фрагментов

Давайте посмотрим, как создать свой собственный фрагмент.

Рассмотрим пример. Предположим, что мне нужно часто писать следующий код в `ActionScript3`:

```
private var _foo:Object;
public function get foo():Object
{
    return _foo;
}
public function set foo(value:Object)
{
    _foo = value;
}
```

Это простая комбинация получения/установки с использованием переменной. Проблема в том, что очень много такого шаблонного кода приходится писать неоднократно. Давайте посмотрим, как автоматизировать это.

Плагин фрагментов языка SnippetsEmu ассоциирует `st` как начальный тег и `et` как закрывающий тег - это направляющие символы, между которыми мы вводим наш код.

Давайте начнем с простого примера.

```
exec "Snippet pubfun public function ".st.et."<CR>{<CR>".st.et."<CR>}<CR>"
```

Добавьте эту строку в ваш `~/vim/after/ftplugin/actionsript_snippets.vim`.

Теперь откройте новый файл, скажем, `test.as`, наберите `pubfun`, нажмите `<tab>` и смотрите как это расширяется:

```
public function <{>>:<{>>
{
}
```

Курсор будет помещен на имени функции, нажмите `tab` чтобы войти в возвращаемый тип функции, введите `tab` снова для ввода тела функции.

Возвращаясь к нашей исходной задаче, вот что я придумал:

```
exec "Snippet getset private var _".st."name".et."<CR><CR>public
functi
```

Примечание: Все фрагменты для этого плагина должны быть введены в одной строке. Это техническое ограничение.

Выполните следующие действия, чтобы можно было использовать новый фрагмент:

1. Добавьте эту строку в ваш `~/vim/after/ftplugin/actionsript_snippets.vim`.
2. Откройте новый файл `test.as`.
3. Наберите `getset` и нажмите `<tab>` и вы должны увидеть следующее:

```
private var _<{name}>;
public function get <{name}>():<{type}>
{
    return _<{name}>;
}
public function set <{name}>(value:<{type}>)
{
    _<{name}> = value;
}
```

4. Наберите `color` и нажмите `<tab>`. Обратите внимание, что переменная с именем `color` везде заменяется.
5. Наберите `Number` и нажмите `<tab>`. Ваш код теперь должен выглядеть так:

```
private var _color;
public function get color():Number
{
    return _color;
}
public function set color(value:Number)
{
    _color = value;
}
```

Посмотрите как в несколько нажатий клавиш мы получили результат! Мы заменили около 11 строк повторяющегося кода одной строкой сценария `Vim`.

Мы можем продолжать добавлять такие фрагменты для упрощения процесса кодирования, что поможет нам сконцентрироваться на реальной работе в программировании.

Смотри `:help snippets_emu.txt` для подробной информации (данный файл помощи будет доступен только после инсталляции плагина).

IDE

`Vim` может использоваться в качестве IDE с помощью нескольких плагинов.

Плагин Project

Плагин Project используется для создания Project-менеджера, используемого в *Vim*.

1. Скачайте плагин [project](#).
2. Разархивируйте его в ваш каталог `~/vim/`.
3. Запустите `:helptags ~/vim/doc/`.
4. Скачайте исходный код Vim из <http://www.vim.org/subversion.php>
5. Запустите `:Project`. Слева откроется боковая панель, которая будет работать как 'project window'.
6. Запустите `\\c` (обратный слэш и 'c')
7. Дайте ответы на следующие параметры
 - введите имя (Name), скажем 'vim7_src'
 - каталог (Directory), скажем `C:\\repo\\vim7\\src\\`
 - опции CD, аналогично как у каталога выше
 - Опции фильтрации (Filter), скажем `*.h *.c`
8. Вы увидите, что боковая панель заполняется списком файлов, которые соответствуют фильтру в указанном каталоге.
9. Используйте клавиши со стрелками или клавиши j/k для перемещения вверх и вниз по списку файлов, а нажатие клавиши Enter откроет файл в главном окне.

Это дает вам знакомый интерфейс в стиле IDE, и хорошо, что нет придуманных файлов конфигурации или задания путей установки, как в интегрированных средах разработки, которые обычно всегда имеют проблемы. Функциональность плагина Project проста и прямолинейна.

Вы можете использовать стандартные команды для открытия и закрытия проектов и их частей.

Вы также можете запускать скрипты в начале и в конце использования проекта, это поможет вам установить переменную PATH или набор опций компиляции и так далее.

Смотрите `:help project.txt` для подробной информации.

Запуск кода из текста

Вы можете запустить код прямо из *Vim* с помощью плагинов, таких как [EvalSelection.vim](#) или простые плагины, такие как [inc-python.vim](#).

Интеграция SCM

Если вы начнете редактировать файл, вы можете сделать автоматическую проверку из Perforce с помощью [плагина perforce](#). Аналогично, есть [плагины для интеграции с CVS/SVN/SVK/Git](#).

Еще

Для обзора остальных плагинов, созданных для реализации IDE в *Vim*, см.:

- [Vim Tip: Using vim as an IDE all in one](#)
- [C++/Python Vim+IDE plugins list](#)

Есть много плагинов, созданных для конкретных языков программирования, которые могут помочь вам изящно работать. Например, для Python могут быть полезны следующие плагины:

- [SuperTab](#) позволяет вам вызывать завершения, просто нажав `tab`, а затем использовать клавиши со стрелками для выбора опции.
- [python_calltips](#) показывает окно в нижней части, в котором дается список возможных завершений. Удобная особенность по сравнению с обычным завершением состоит в том, что вы можете посмотреть в документации информацию по каждому из вариантов.
- [VimPdb](#) поможет вам отладить программы на Python в *Vim*.

Создание своих плагинов

Вы можете написать свои плагины для расширения возможностей *Vim*. Например, вот задача которую вы можете сделать:

Написать плагин для *Vim*, который берет текущее слово и открывает браузер с документацией для этого

конкретного слова (слово может быть именем функции или именем класса, и так далее).

Если вы не понимаете как это сделать, просмотрите ["Online documentation for word under cursor" tip at the Vim Tips wiki](http://vim.wikia.com/wiki/Online_documentation_for_word_under_cursor).

Я расширил пособие и сделал его более общим:

```
" Add the following lines to your ~/.vimrc to enable online documentation
" Inspiration: http://vim.wikia.com/wiki/Online_documentation_for_word_under_cursor
function Browser()
  if has("win32") || has("win64")
    let s:browser = "C:\\Program Files\\Mozilla Firefox\\firefox.exe
    -new-tab"
  elseif has("win32unix") " Cygwin
    let s:browser = "/cygdrive/c/Program Files/Mozilla\
    Firefox/firefox.exe' -new-tab"
  elseif has("mac") || has("macunix") || has("unix")
    let s:browser = "firefox -new-tab"
  endif
  return s:browser
endfunction
function Run(command)
  if has("win32") || has("win64")
    let s:startCommand = "!start"
    let s:endCommand = ""
  elseif has("mac") || has("macunix") " TODO Untested on Mac
    let s:startCommand = "!open -a"
    let s:endCommand = ""
  elseif has("unix") || has("win32unix")
    let s:startCommand = "!"
    let s:endCommand = "&"
  else
    echo "Don't know how to handle this OS!"
    finish
  endif
  let s:cmd = "silent " . s:startCommand . " " . a:command . " " .
  s:endCommand
  " echo s:cmd
  execute s:cmd
endfunction
function OnlineDoc()
  if &filetype == "viki"
    " Dictionary
    let s:urlTemplate = "http://dictionary.reference.com/browse/<name>"
  elseif &filetype == "perl"
    let s:urlTemplate = "http://perldoc.perl.org/functions/<name>.html"
  elseif &filetype == "python"
    let s:urlTemplate = "http://www.google.com/search?q=<name>&domains
    =docs.python.org&sitesearch=docs.python.org"
  elseif &filetype == "ruby"
    let s:urlTemplate = "http://www.ruby-doc.org/core/classes/<name>.html"
  elseif &filetype == "vim"
    let s:urlTemplate = "http://vimdoc.sourceforge.net/search.php?
    search=<name>&docs=help"
  endif
  let s:wordUnderCursor = expand("<cword>")
  let s:url = substitute(s:urlTemplate, '<name>', s:wordUnderCursor, 'g')
  call Run(Browser() . " " . s:url)
endfunction
noremap <silent> <M-d> :call OnlineDoc()<CR>
inoremap <silent> <M-d> <Esc>:call OnlineDoc()<CR>a
```

Доступ к базам данных

Вы можете даже работать с 10 различными базами данных: Oracle, MySQL, PostgreSQL, Sybase, SQLite, и всё из Vim, с помощью [плагина dbext.vim](#). Самое приятное то, что этот плагин поможет вам редактировать SQL, написанный в PHP, Perl, Java и т.д. И вы даже можете напрямую выполнять SQL запрос, даже если он встроен в другой язык программирования, и вас даже спросят о значении переменных.

Итоги

Мы изучили, как Vim можно использовать в программировании с помощью различных плагинов и установок. Если вам нужно больше, вы можете написать свои плагины для Vim (как мы уже видели в главе Scripting).

Хорошим источником дискуссий по теме будет [Stack Overflow](#) и блог [Peteris Krumin's](#).

Внешние ссылки

<http://www.codinghorror.com/blog/archives/001188.html>
<http://ctags.sourceforge.net>
http://www.vim.org/scripts/script.php?script_id=273
http://cscope.sourceforge.net/cscope_maps.vim
http://www.vim.org/scripts/script.php?script_id=1638
http://vim.sourceforge.net/scripts/script.php?script_id=213
http://www.vim.org/tips/tip.php?tip_id=1228
<http://code.google.com/p/pysmell/>
http://www.vim.org/scripts/script.php?script_id=1318
http://www.vim.org/scripts/script.php?script_id=69
http://www.vim.org/scripts/script.php?script_id=889
http://www.vim.org/scripts/script.php?script_id=1941
http://www.vim.org/scripts/script.php?script_id=240
http://www.vim.org/scripts/script.php?script_id=90
http://vim.wikia.com/wiki/Using_vim_as_an_IDE_all_in_one
<http://phraktur.net/vimmity-vim-vim.html>
http://www.vim.org/scripts/script.php?script_id=1643
http://www.vim.org/scripts/script.php?script_id=1074
http://www.vim.org/scripts/script.php?script_id=2043
http://vim.wikia.com/wiki/Online_documentation_for_word_under_cursor
http://www.vim.org/scripts/script.php?script_id=356
<http://stackoverflow.com/questions/tagged/vim>
<http://www.catonmat.net/tag/vim>

Vim :Разное

Введение

Мы рассмотрели много возможностей Vim, но мы до сих пор не охватили всех их, так что давайте сделаем краткий обзор по различным темам, которые являются полезными и интересными.

Строка режима (modeline)

Что, если вы хотите указать, что в определенном файле всегда следует использовать только табуляцию и не использовать пробелов при редактировании. Можем ли мы сделать это в рамках самого файла?

Да, просто поместите [vim: noexpandtab](#) в первые две строки или в две последние строки файла.

Например:

```
# Sample Makefile
.cpp:
    $(CXX) $(CXXFLAGS) $< -o $@
```

```
# vim: noexpandtab
```

Строка, которую мы добавили, называется "modeline" (режимной).

Портируемый Vim

Если вы пользуетесь различными компьютерами, было бы удобно поддерживать одинаковые настройки вашего Vim на каждой машине? Было бы полезно, если бы вы могли бы просто запускать Vim со своей флешки (USB disk)? Для этого есть [портируемый GVim](#).

Просто разархивируйте (unzip) его в каталог на переносимом диске, затем запустите GVimPortable.exe. Вы можете сохранять ваш vimrc и другие нужные файлы на диске и использовать его везде на компьютерах с Microsoft Windows.

Обновление плагинов

Любой достаточно продвинутый пользователь Vim будет использовать кучу плагинов и скриптов к нему, расположенных в каталогах `~/.vim` или `~/vimfiles`. Что делать, если мы захотим обновить их всех до последней версии?

Вы можете посетить странички всех скриптов, скачать и установить их, но есть лучший способ - просто запустите `:GLVS` (что расшифровывается как 'G'et 'L'atest 'V'im 'S'cripts). См. `:help getscript` для подробной информации.

В Vim есть даже [скрипты для twitter](#)!

Плагины Dr.Chip's

"Dr. Chip" написал [несколько удивительных Vim плагинов](#) в течение нескольких лет. Мой любимый [drawit.vim](#), который поможет вам сделать рисунки в тексте, наподобие модных ASCII диаграмм, которые вы видели до этого.

Другой любимый плагин [Align.vim](#), который поможет вам выровнять несколько подобных строк. Например, у вас есть следующий фрагмент кода программы:

```
a = 1
bbbbbb = 2
ccccccccc = 3
```

Просто визуально выберите эти три строки и нажмите `\t=`, и вуаля, теперь вы имеете:

```
a           = 1
bbbbbb     = 2
ccccccccc  = 3
```

Теперь это легко читаемо и ваш код выглядит более профессиональным.

Изучите страничку Dr. Chip's, чтобы узнать о многих других интересных плагинах.

Ведение блога с помощью Vim

Используя плагин [Vimpress](#), вы можете писать блоги в Wordpress прямо из Vim.

Работа Firefox подобно Vim

Используйте дополнение [Vimperator](#), чтобы сделать поведение Firefox аналогичным Vim, включая модальное поведение, горячие клавиши для посещения ссылок, строку состояния, автодополнения и даже поддержку закладок!

Семь привычек Bram-a

Узнайте о семи привычках Bram Moolenaar, создателя Vim, написавшем давно статью под названием ["Семь привычек эффективного редактирования текста"](#), которая описывает, как вы должны использовать хороший редактор (например, Vim).

Bram недавно выступил с докладом под названием [«Семь привычек эффективного редактирования текста. 2.0»](#), где он переходит к описанию новых возможностей в Vim, а также, как эффективно использовать Vim. Этот доклад неплохо бы послушать обычному пользователю Vim.

Помощь Vim

Вы можете внести свой вклад в развитие *Vim* различными способами, такими, как участие в [разработке Vim](#), создание [плагинов и цветовых схем](#), участие в разработке [подсказок](#) и [документации](#).

Если вы хотите помочь в разработке самого *Vim*, смотрите [:help development](#).

Сообщество

Многие пользователи *Vim* заходят в почтовую рассылку vim@vim.org, где спрашивают и отвечают на поставленные вопросы. Лучший способ узнать больше о *Vim* и помочь другим начинающим изучать *Vim* - это частенько читать (и отвечать на) сообщения в этом списке рассылки.

Вы можете также задавать вопросы в [Stack Overflow by tagging the question as 'vim'](#), где вы найдете полезные обсуждения, например, на тему ["What are your favorite vim tricks?"](#)

Смотрите также заметки и обсуждения на [delicious](#) и [reddit](#).

Итоги

Мы рассмотрели широкий спектр связанных с *Vim* тем, и насколько он может быть полезен для нас. Не стесняйтесь исследовать эти и многие другие [сценарии Vim](#), которые облегчат вам редактирование и сделают его еще более удобным.

Внешние ссылки

<http://portablegvim.sourceforge.net>

http://www.vim.org/scripts/script.php?script_id=1853

<http://mysite.verizon.net/astronaut/vim/>

http://www.vim.org/scripts/script.php?script_id=294

http://www.vim.org/scripts/script.php?script_id=1953

<http://vimperator.mozdev.org/>

<http://www.moolenaar.net/habits.html>

<http://video.google.com/videoplay?docid=2538831956647446078&q=%22Google+engEDU%22>

http://groups.google.com/group/vim_dev

<http://www.vim.org/scripts/>

<http://vim.wikia.com>

<http://vimdoc.sourceforge.net>

<http://www.vim.org/maillist.php#vim>

<http://delicious.com/popular/vim>

<http://www.reddit.com/r/vim/>

<http://www.vim.org/scripts/>

Vim :Что дальше

Введение

Мы так много узнали о *Vim*, и что же дальше?

Ну, если вы узнали, поняли и сделали *Vim* привычным, то вы официально Vimmer. Поздравляем!

А сейчас немедленно [отправьте мне письмо](#) с благодарностью за эту книгу;-). Этот шаг является необязательным, но рекомендуемым.

Также, пожалуйста, рассмотрите вопрос о [внесении пожертвований](#), чтобы поддержать продолжение разработки этой книги, или помогите проекту *Vim*, чтобы [помочь детям в Уганде](#).

Далее, я рекомендую регулярно следить за [списком рассылки Vim](#), просматривая вопросы и ответы, которые там имеются. Вы будете удивлены, увидев размах применений и диапазон гибкости *Vim*, который демонстрируется в ходе этих обсуждений. И, может быть, вы тоже сможете ответить на некоторые вопросы!

Два других важных ресурса, которые должны быть хорошими помощниками, страничка ["Best of Vim Tips"](#) и [:help user-manual](#) - индекс всего, что возможно в *Vim*. Убедитесь, что у вас есть руководство по эксплуатации, в которое время от времени придется заглядывать, чтобы найти нужную информацию.

И в завершение, если вы хотите знать о последних интересных функциях в *Vim*, то смотрите [:help new-7](#).

Итоги

Мы просмотрели очень широкий круг возможностей в *Vim*. Самое главное, что мы уже неоднократно подчеркивали, что не надо изучать все подробно все возможности, но нужно освоить самые полезные для вас функции и превратить их использование в привычку. Ну и не забывать использовать `:help` в случае необходимости.

Продвигайтесь таким образом вперед и осваивайте навыки редактирования, чтобы увеличить эффективность и действенность вашей работы до такого уровня, о котором вы даже не подозревали.

Удачного Vimming-a!

Внешние ссылки

<http://www.swaroopch.com/contact/>
<http://www.swaroopch.com/byteofdonate>
<http://www.vim.org/maillist.php#vim>
<http://rayninfo.co.uk/vimtips.html>
http://vimdoc.sourceforge.net/html/doc/usr_toc.html

***Vim* :Обратная связь**

Эта книга находится в процессе создания. Ваша обратная реакция имеет важное значение для улучшения книги. Пожалуйста, [присылайте ваши комментарии](#).

Эта книга доступна на [официальном web-сайте как wiki](#), что означает, что вы можете зайти и исправить то, что вам не нравится!

Известные проблемы:

- Непоследовательность в примерах
- Усиление последних глав для опытных Vimmers в отличие от главы для новичков.

Внешние ссылки

<http://www.swaroopch.com/contact/>
<http://www.swaroopch.com/notes/Vim>

***Vim* :Благотворительность**

Vim занимается благотворительностью. Если вы сочли *Vim* полезным, рекомендуется помочь [фонду ICCF](#) Голландии, в любой удобной для вас форме.

Цитата с их сайта:

The south of Uganda has been suffering from the highest HIV infection rate in the world. Parents die of AIDS, just when their children need them most. An extended family can be their new home. Fortunately there is enough food in this farming district. But who will pay their school fees, provide medical aid and help them grow up? That is where ICCF Holland helps in the hope that they will be able to take care of themselves, and their children, in the long run.

Подробности о ICCF можно найти в *Vim* запустив `:help iccf`.

Чтобы помочь разработке самого *Vim*, вы можете [спонсировать разработку Vim](#), сделав денежное пожертвование, и стать зарегистрированным пользователем *Vim* - вы получите право голоса при выборе того, какие новые функции будут добавлены в *Vim*!

Внешние ссылки

<http://iccf-holland.org>
<http://www.vim.org/sponsor/faq.php>

Vim :Послесловие

Об этой книге

Это альфа-издание книги. Это означает, что она может содержать неполные разделы, очевидные ошибки, вопиющее отсутствие деталей и многое другое. Пожалуйста, не стесняйтесь, <http://www.swaroopch.com/contact/> по исправлению.

Создание этой книги

Эта книга никогда не была бы написана без *Vim* 7. Моя жизнь никогда не была бы полной без *Vim*. Спасибо Bram Moolenaar и мировому сообществу Vimmers.

Оригинальное содержание было написано в формате Viki, а затем были добавлены [команды deplate](#). Особая благодарность Thomas Link, он терпеливо отвечал на все мои вопросы о том, как использовать Viki/Deplate :-)

Книга позже была преобразована в формат wiki. Хотя книга в настоящее время редактируется в Интернет, я все еще использую *Vim* для создания текста и copy/paste в MediaWiki, сохраняя текст.

Вдохновение

Я начал писать эту книгу где-то в 2004. Но от вскоре отложил это дело.

Я продолжил писать книгу через месяц после [foss.in/2006](#) благодаря заразившему меня оптимизму этих людей.

Но я снова не выдержал. Я начал работать более продуктивно, когда я начал следовать принципам David Allen's Getting Things Done. Это вернуло меня к работе над книгой, чтобы завершить её, наконец, в какой-то форме, чтобы выпустить в мир в первый день [foss.in/2008](#).

Об авторе

Swaroop C H — автор этой книги. Ему 26 лет. Он закончил В.Е. (Computer Science) в PESIT, Bangalore, India. Сейчас работает и является соучредителем [IONLAB](#). Прежде работал в Yahoo! и Adobe. Его хобби - чтение и письмо, бег и езда на велосипеде, и немного программирование.

Больше о нем: <http://www.swaroopch.com/about/>

Вы можете контактировать с ним через <http://www.swaroopch.com/contact/>

Внешние ссылки

<http://www.swaroopch.com/contact/>

<http://deplate.sourceforge.net>

http://www.vim.org/account/profile.php?user_id=4037

<http://www.foss.in>

<http://www.swaroopch.com/gtdbook>

<http://www.ionlab.in>

Vim :Переводы

- Если вы хотите узнать как сделать перевод, пожалуйста посмотрите '[A Byte of Python' Translation Howto](#).
- Если вы уже в процессе создания перевода, пожалуйста добавьте ваше описание и язык перевода на этой странице, подобно [списку переводов 'A Byte of Python'](#).

Шведский (Swedish)

Mikael Jacobsson (leochingwake-at-gmail-dot-com) has volunteered to translate the book to Swedish, and the translation is in progress at http://leochingwake.se/wiki/index.php/Byte_of_Vim.

Традиционный китайский (Traditional Chinese)

Yeh, Shin-You (or Yesyo) has volunteered to translate the book to Traditional Chinese. The translation is in progress, and starts with the chapter "[Vim zh-tw](#)".

Русский (Russian)

Vitalij Naumov (or [hbvit7](#)) has volunteered to translate the book to Russian. The translation is in progress, and starts with the chapter "Vim_ru".

Автор данного, фривольного перевода (v 1.0.) Войтенко Андрей Валентинович. vjy.2005@gmail.com.

От переводчика пару слов: Я переводил это творение с нуля и только потом заметил, что человек уже начал переводить данную книгу, но не закончил.

Сама книга может служить прекрасной вводной статьей для начинающих пользователей *Vim*, из которой можно почерпнуть основы работы в *Vim* и ознакомиться с возможностями этого редактора.

В процессе работы системным администратором мне пришлось столкнуться с необходимостью использования *Vim* и, переводя эту книгу, я просто заполнил свой пробел в знаниях по данному вопросу.

Перевод откровенно слаб, но прошу не ругать меня за это, а продолжить перевод или скорректировать его в лучшую сторону! Я не профессиональный переводчик и мое знание английского оставляет желать много лучшего.

При переводе этой книги я активно пользовался google translate и набирал текст в LibreOffice.

В книге есть много ссылок, которые указаны в конце глав, но убраны из основного текста. Я думаю, что для человека, который захочет найти информацию, этого будет достаточно. Все ссылки на оригинал книги и автора оставлены в целях соблюдения лицензии.

Приятного прочтения!