Яловой Илья Викторович

Оригинал: www.ar2.mksat.net

Путь к VIM.

Оглавление

НИКОЛАЕВ 2003

Вступление

Необходимость использовать текстовый редактор (ТР) возникает достаточно часто, особенно у пользователя систем подобных UNIX. Поэтому каждый стремиться подобрать себе редактор по вкусу. Начнем с того, что их огромное количество, и даже под Linux их уже написано достаточно много. Среди них есть очень достойные кандидаты на звание лучшего, но ...

Оставив философский аспект выбора ТР, я просто изложу в краткой форме как выбирал я сам. Прежде всего, мне нужен был быстрый и небольшой редактор, с подсветкой синтаксиса, макросами и возможностью оперативно выполнять команды, не выходя из редактора, ну такие как make и др. Найти такой редактор не так уж просто! Под MSDos я пользовался qedit, который имел только два достоинства - он был очень маленького размера и грузился мгновенно даже на машинах, которые уступают по быстродействию современным калькуляторам. На Amiga я пользовался CED, который умел все, кроме подсветки синтаксиса. Потом, правда, появился ProgEdit, но он был уж очень неповоротлив и работал со скоростью тормознутого текстового процессора. Он у меня не прижился. Когда я вернулся на PC (После Amiga, я более-менее комфортно смог себя чувствовать только на Pentium II) то долго перебирал различные редакторы и остановился на Aditor. На мой взгляд, это лучший ТР на этой платформе. Он вполне может удовлетворить требования широких масс, оставаясь быстрым и малогабаритным. Собственно я еще не встречал ни одного человека, который бы поспорил со мной по этому вопросу. Затем началась гонка производителей железа и программного обеспечения, намек, я думаю, достаточно прозрачный. Меня как-то не слишком привлекала перспектива перманентного апгрейда и изучения новых творений софтверного гиганта. Поэтому я начал искать альтернативы. Мои впечатления о Linux подробно изложены в другом документе и здесь я не буду касаться этой темы. Скажу только, что я стал пытаться переползти на Linux.

Подобная миграция, конечно, вызвала массу проблем. Главная из них - это то обстоятельство, что при всем нежелании существует объективная необходимость работать в обеих операционных системах и, что еще хуже, - держать их обе на одной машине. Вторая проблема - это возрастающее количество программного обеспечения, подлежащего изучению.

О том, что VIM существует на многих платформах, я знал еще со времен Amiga. Более того, я слышал восхищенные отзывы, от людей деяния и авторитет которых вызывает глубокое уважение. Конечно, я решил попробовать.

От первого знакомства осталось только чувство недоумения и полной растерянности. Толи людей я не понял, толь я сам ... То есть попытка запустить и "поредактировать" окончилась сокрушительным фиаско. И если версию с графическим интерфейсом еще можно было кое-как использовать благодаря иконкам и меню, то текстовая версия просто повергала меня в полное уныние.

Прошло некоторое время, и я нашел краткий справочник на русском языке по использованию Vi (прародителя VIM). Почитал и понял что эта программа не для нормального человека, а исключительно для маньяков. Сейчас мне смешно вспоминать - справочник занимал неполных четыре страницы крупным шрифтом! Я решил, что по доброй воле я больше не вернусь к этому редактору.

Изменить своему решению меня заставили два обстоятельства:

первое - я хотел иметь под Linux и Windows один редактор;

второе - меня мучило чувство собственной неполноценности, люди, мол пользуются и довольны - нахвалиться не могут, а я даже сохранить документ толком не смог, о каких-то манипуляциях с текстом я вообще молчу!

Я решительно взялся за документацию, которая поставляется вместе с VIM. Вот тут меня ждал сюрприз - объем этой документации. Он поистине необъятный. То есть посидеть вечерок - разобраться в принципе невозможно!

Последняя моя попытка произошла совершенно случайно - нечем было заняться, и я решил распечатать и внимательно прочитать руководство по VIM. Потратив день на чтение и эксперименты, я понял, что им можно таки редактировать текст!!!

С того времени прошел год, и я, практически, пользуюсь только этим редактором. Должен признать, что он достоин того, чтобы потратить время на его изучение. Скажу честно, что каждый раз, когда я заглядываю в документацию, то нахожу новые полезные функции и возможности.

В чем же заключается проблема, которая мешала мне сразу освоить этот редактор. Проблемы две - одна заключена во мне, а вторая заключается в том, что VIM принципиально отличается по своей идеологии от "традиционных" редакторов.

Я глубоко уважаю и высказываю свое почтение людям, которые самостоятельно его освоили и используют, значительно увеличив свою личную производительность, а тем, кому это не удалось, но и желание не пропало, я рекомендую ознакомиться с этим документом.

Я искренне желаю вам успеха в освоении этого замечательного редактора и постараюсь помочь, дав базовые знания и весь арсенал, необходимый для начала полноценной работы.

Авторские права

1. Достоинства

Прежде всего, отмечу, что этот редактор отвечает всем требованиям, перечисленным ранее. Спорными можно назвать только его размер и скорость работы. Не пугайтесь! Занимает он вполне приемлемо и достаточно быстро работает. Но, честно говоря, эти параметры находятся очень близко от грани. Aditor занимает раза в 3 меньше и намного быстрее. Но я перестал беспокоиться по этому поводу, когда посмотрел файлы конфигурации подсветки синтаксиса! VIM - работает с синтаксисом значительно серьезнее, да и вообще когда начинаешь понимать его реальную мощь, то отпадают всякие сомнения.

Чтобы в полной мере оценить этот редактор с ним надо поработать. И именно здесь закопана собака!

Чтобы в полной мере оценить этот редактор с ним надо поработать ПРОФЕССИОНАЛЬНО. Тем не менее, постараюсь кратко сформулировать основные положительные аспекты:

- мощнейшая система подсветки синтаксиса;
- интеллектуальное форматирование при наборе исходных текстов программ;
- невероятная гибкость (очень похоже на теле рекламу, но с ним действительно можно творить чудеса в плане конфигурации и настройки под конкретного пользователя или конкретные задачи);
- различные возможности автоматизации набора и редактирования текста, включая макросы;
- возможность выполнения команд операционной системы;

- возможность одновременной работы с несколькими документами;
- расширенные возможности по отладке программ (настраиваемая система разбора файла с ошибками компиляции);
- очень удобные функции при редактировании текста;
- наличие нескольких буферов обмена (регистров);
- очень серьезная система поиска (от очень простого и быстрого, до сложного - с использованием регулярных выражений во всей их красе).

Перечислять можно долго, но оценить вы это сможете, только СЕРЬЕЗНО пощупав собственными руками.

2. Основные принципы работы

Как я уже говорил, работать с VIM можно, только понимая его идеологию. Чем же он отличается от "обычных" редакторов?

Далее постараюсь быть лаконичным и исключить лирику для лучшего восприятия материала – уж очень это серьезное дело. Текстовый редактор VIM может находиться в одном из трех состояний (В скобках указаны оригинальные английские названия):

1. Режим ввода команд (**NORMAL**).
2. Режим редактирования (**INSERT**).
3. Режим выделения текста (**VISUAL**).

Существуют еще режим совместимый с VI - он нас не интересует, и режим выделения текста эмулирующий поведение обычных редакторов под Windows - Select, но при этом мы теряем больше, чем находим, и я его рассматривать не буду.

Рассмотрим подробнее каждое состояние редактора. Каждое состояние соответствует определенному режиму работы, предназначенному для выполнения определенных операций.

- NORMAL:

В этом режиме выполняются различные команды, такие как чтение-запись файлов, изменение параметров редактора и др.

- INSERT:

В этом режиме осуществляется вставка текста и непосредственное редактирование.

- VISUAL:

В этом режиме можно выделять текст для последующих манипуляций, таких как удаление, перемещение по листу, копирование в один из регистров изменение форматирования и др.

Имея три основных режима работы, мы должны оперативно между ними переключаться. Основным режимом принято считать режим **NORMAL**. Поэтому все переходы осуществляются через него.

Переход		Команда	Комментарий
Из режима	В режим		
NORMAL	INSERT	i или "Insert"	Кнопка клавиатуры "Insert" кроме того переключает режимы вставки – замены.
	INSERT	a	Вставка текста после символа, подсвеченного курсором

	INSERT	o	Вставка новой строки после курсора с переходом в режим INSERT
	INSERT	O	Вставка новой строки перед курсора с переходом в режим INSERT
	INSERT	cc, S	Очистка строки и переход в режим INSERT
	INSERT	C	Удалить правую часть строки и переход в режим INSERT
	INSERT	s	Удалить подсвеченный символ и переход в режим INSERT
	VISUAL	v	Режим выделения текста (с точностью до знака)
	VISUAL	V	Режим выделения текста (с точностью до строки)
	VISUAL	Ctrl+V (Ctrl+Q)	Режим выделения прямоугольной области текста, следует отметить, сто комбинация клавиш Ctrl+V в Windows используется для вставки текста из буфера, поэтому вместо нее следует использовать альтернативный вариант Ctrl+Q.
	VISUAL	gv	Возврат к области, выделенной в прошлый раз.
NORMAL или INSERT	VISUAL	<LeftMouse>	Прекращает выделение или начинает новое
	VISUAL	<RightMouse>	Продолжает выделять область
INSERT	NORMAL	ESC	Выход в режим NORMAL
	NORMAL	Ctrl+O	Кратковременный выход в режим NORMAL для выполнения одной команды, после чего осуществляется автоматический возврат в режим INSERT
VISUAL	NORMAL	Ctrl+c, v, V,...	Вернуться в NORMAL можно повторив ту комбинацию, с помощью которой вы вошли в VISUAL. При этом не будет выполнено никаких действий над выделенной областью. Любая команда которая совершается над выделенной областью также переводит редактор в режим NORMAL, но подробно эти операции будут рассмотрены позже.

Далее я буду рассматривать как выполняются те операции, к которым мы привыкли, используя "традиционный" TP с графическим интерфейсом (если кто помнит, то были строчные и экранные редакторы, так вот VIM, на мой взгляд, ближе к строчным). VIM имеет множество команд типа "удалить 10 слов от текущего положения курсора" или "перейти на 4 слова влево". Глупо говорить, что такие команды не нужны, просто современный рядовой пользователь не привык к таким командам – его орудие мышка. И вот как раз такому человеку будет интересно (надеюсь) прочитать далее, как можно выполнять привычные для него действия в таком своеобразном редакторе.

3. Чтение файла, сохранение, выход из программы

Итак, самые главные операции, которые необходимы любому ТР. Общая идея заключается в том, что VIM имеет некий набор буферов и экранов. Буфер – это собственно файл (не совсем корректно, но в первом приближении сойдет), а экран это то окно, в котором отображается буфер.

Все операции по работе с файлами, буферами и экранами осуществляются из режима **NORMAL** .

:edit – чтение файла и большинство вариаций на эту тему осуществляется с помощью этой команды. Общий формат команды имеет вид (здесь и далее в квадратные скобки заключаются необязательные элементы):

:e[dit][!] [++opt] [+cmd] [Имя файла]

где

++opt - это параметр позволяющий изменить значения fileformat и fileencoding, персонально для открываемого файла. Могут быть полезны следующие варианты:

++enc=koi8-r устанавливает кодировку открываемого файла

++enc=koi8-u

++enc=cp1251

++enc=latin1

++ff=unix строки заканчиваются символом <NL>

++ff=dos строки заканчиваются символом <CR><NL>

++ff=mac строки заканчиваются символом <CR>

+cmd - параметр позволяющий указать положение курсора для открываемого файла или выполнить другую команду.

Вопрос работы с различными русскими кодировками рассмотрен подробнее в соответствующем разделе.

Возможные варианты сохранения файла показаны в таблице:

Команда	Описание
NORMAL	
:w[rite]	Сохранить текущий буфер
:w!	Сохранить текущий буфер не взирая на атрибут “только для чтения”
:sav[eas][!] {file}	Сохранить буфер под новым именем, имя текущего буфера тоже меняется. Предыдущее имя используется как альтернативное. Флаг “!” необходим для записи поверх существующего файла.

:wa[ll]	Сохранить все содержимое всех буферов, котрые были изменены
:wq[!]	Сохранение файла и выход из программы

Вопрос работы с различными русскими кодировками рассмотрен подробнее в соответствующем разделе.

:q[uit] – обеспечивает выход из программы. Если один из открытых файлов был изменен и не сохранен, то для выхода из программы необходимо использовать **:q!** или предварительно сохранить все файлы.

4. Навигация по тексту

Для эффективного редактирования текста мы должны иметь возможность перемещаться по нему в любом направлении и к любому его фрагменту. VIM имеет для этого потрясающие и очень разноплановые возможности. Начнем с простого. для перемещения курсора традиционно используются следующие клавиши (только режим **NORMAL**): 'h' – влево, 'j' – вниз, 'k' – вверх, 'l' – вправо.

Такой подход обеспечивает максимальную скорость, для людей владеющих "слепой печатью", так как избавляет их от необходимости убирать руки от основной клавиатуры. Для людей владеющих "однопальцевой печатью" в последних версиях VIM добавлена возможность перемещения по файлу с помощью традиционных клавиш как в режиме **NORMAL** , так и в режиме **INSERT** . Но такие манипуляции далеко не предел возможностей VIM. Далее привожу сводную таблицу соответствующих команд. Надо заметить, что VIM поддерживает технологию подобную гиперссылкам.

Команда	Описание
Общие	
<HOME>	Переход в начало строки
<END>	Переход в конец строки
NORMAL	
f{символ}	Поиск указанного символа в текущей строке вправо
F{символ}	Поиск указанного символа в текущей строке влево
%	Переход к парной скобке - очень удобно при написании программ на C/C++, Lisp
{nn}G или	Переход к строке nn, если номер строки не указан, то G - переход в конец, а gg - в начало документа. Эта функция очень помогает при отладке программ, однако VIM имеет специальный инструментарий для

{nn}gg	создания удобной среды разработки. Этот вопрос рассмотрен далее
CTRL+O	Возврат на предыдущую позицию (при переходах по документам с помощью меток, именованных меток, ссылок или с помощью команд gg и G)
CTRL+I	Команда обратная по действию CTRL+O. Подробнее смотрите описание использования меток.
*	Переход на следующее слово, аналогичное подсвеченному курсором. (Поиск слова, на котором в данный момент стоит курсор, далее по тексту)
#	Аналогична предыдущему, но в обратном направлении
)	Переход на начало следующего предложения
(	Переход на начало предыдущего предложения
}	Переход на начало следующего параграфа
{	Переход на начало предыдущего параграфа
[m	Переход назад к началу описания метода (Java)
[M	Переход назад к концу описания метода (Java)
]m	Переход вперед к началу описания метода (Java)
]M	Переход вперед к концу описания метода (Java)
[*	Переход назад к началу блока коментария (/*)
]*	Переход вперед к концу блока коментария (*/)
CTRL+]	Переход по ссылке
INSERT	

Очень полезными могут оказаться именованные метки, особенно, когда приходится вносить изменения в нескольких частях одного документа. Для работы с именованными метками используются следующие команды:

Команда	Описание
NORMAL	
m{имя}	Установить метку с именем {имя}. Именем метки может быть любая (одна) буква

'{имя}	Переход к строке с указанной меткой
`{имя}	Переход в конкретное место (строка и столбец) помеченное меткой
:marks	Показать все определенные метки

Кроме определяемых пользователем, имеются также предопределенные метки:

Метка	Описание
'	Позиция курсора до перемещения (метка, ссылка, поиск ...)
"	Позиция курсора во время последнего редактирования
[	Начало последнего изменения
]	Конец последнего изменения

5. Редактирование

Под редактированием я понимаю различные манипуляции над текстом и его форматированием. Соответственно для каждого из режима доступны различные команды. В режиме **VISUAL** осуществляются изменения выделенного участка текста. В режиме **NORMAL** в качестве обьекта редактирования выступает текущая строка, слово или символ. Корректура собственно текста осуществляется непосредственно в режиме **INSERT**, в котором вы можите добавлять новые символы, удалять или заменять старые. Естественно и ввод текста осуществляется в этом режиме.

Команда	Описание
VISUAL	
~	Изменить регистр выделенного фрагмента текста
c [x]	Удалить выделенный участок в регистр “x” и перейти в режим вставки
>	Сдвинуть выделенный фрагмент вправо
<	Сдвинуть выделенный фрагмент влево
!	Отфильтровать выделенные строки с помощью внешней программы
NORMAL	
<DEL>, x	Удалить символ правее курсора
X	Удалить символ левее курсора
dd	Удалить текущую строку
u	Отмена последнего действия
CTRL-R	Вернуть исправления
D	Удалить до конца строки
.	Повторить последнее действие
r	Заменить символ подсвеченный курсором
~	Поменять регистр символа под курсором
INSERT	
<CTRL> + A	Вставляет текст, который был введен в прошлый insert-сеанс
<CTRL> + W	Удаляет предыдущее слово
<DEL>	Удаляет следующий символ
<CTRL> + N	Автоматическое завершение слова с поиском вперед по тексту
<CTRL> + P	Автоматическое завершение слова с поиском назад по тексту

<CTRL> + T	Вставить табуляцию в начало текущей строки
<CTRL> + D	Удалить табуляцию из начала текущей строки
<CTRL> + Q	Вставляет символ, код которого следует за нажатием комбинации.
<CTRL> + X	Переходит в дополнительный режим, в котором вы можите выполнять автозавершение или другие полезные действия. Подробнее этот режим рассмотрен далее.
<CTRL> + E	Вставляет символ, который находится в той же позиции, что и курсор, но ниже.
<CTRL> + Y	Вставляет символ, который находится в той же позиции, что и курсор, но выше.

Особый интерес представляет дополнительный режим, в который можно перейти из режима **INSERT**, нажав комбинацию клавиш **<CTRL> + X**. Чтобы выполнить требуемое действие вам необходимо выбрать одну из следующих комбинаций:

Комбинация	Описание
<CTRL> + E	Сдвиг окна на одну строку вверх (курсор остается на месте)
<CTRL> + Y	Сдвиг окна на одну строку вниз (курсор остается на месте)
<CTRL> + L	Автоматическое завершение целой строки с поиском в обратном направлении.
<CTRL> + K	Автоматическое завершение слова из файла, указанного в параметре 'dictionary'
<CTRL> + T	Автоматическое завершение слова из файла, указанного в параметре 'thesaurus'. Пример использования данной команды рассмотрен далее.
<CTRL> + I	Автоматическое завершение слова с поиском в текущем файле и всех подключаемых файлах.
<CTRL> +]	Автоматическое завершение тэгов
<CTRL> + F	Автоматическое завершение имени файла

Особое внимание заслуживает комманда **<CTRL> + T**. Она представляет собой поиск в словаре. Рассмотрим пример использования этой команды. Допустим файл словаря содержит следующую строку:

```
великолепный замечательный прекрасный отличный превосходный
```

Если вы наберете великол и нажмете комбинацию **<CTRL> + T** , то будет вставлено слово **великолепный.** Последующие нажатия этой комбинации будут вставлять следующие слова соответственно: **замечательный, прекрасный, отличный, превосходный.**

Способов применения этой команды может быть множество, от подбора синонимов до организации удобной работы со словарем (при переводе).

6. Удаление, копирование, вставка – работа с регистрами

Не смотря на то, что команды автоматического завершения часто избавляют нас от необходимости копировать и вставлять текст, тем не менее необходимость таких операций остается. Копирование и удаление фрагментов текста выполняются в режиме **VISUAL**. Исключение составляю команды удаления целого слова или строки, так как в этом случае границы фрагмента очевидны исходя из положения курсора. Следует отметить, что в отличии от большинства текстовых редакторов, использующих системный буфер, VIM имеет обширный набор собственных буферов (регистров). Другими словами у вас есть возможность иметь одновременно скопированными несколько фрагментов текста. В том числе операции удаления, копируют удаленный фрагмент в специальный регистр.

Команда	Описание
VISUAL	
ab	Выделить фрагмент ограниченный символами “(” и “)” включительно
ib	Выделить фрагмент ограниченный символами “(” и “)”
aB	Выделить фрагмент ограниченный символами “{” и “}” включительно
iB	Выделить фрагмент ограниченный символами “{” и “}”
ap	Выделить параграф
a[	Выделить фрагмент ограниченный символами “[” и “]” включительно
i[	Выделить фрагмент ограниченный символами “[” и “]”
a<	Выделить фрагмент ограниченный символами “<” и “>” включительно
i<	Выделить фрагмент ограниченный символами “<” и “>”
:	Выполнить ex- команду для выделенного фрагмента
["x] D	Удалить выделенные строки в регистр “x”
["x] Y	Скопировать выделенный фрагмент в регистр “x”
["x] p	Заменить выделенный фрагмент содержимым регистра “x”
u	Установить для выделенного фрагмента нижний регистр
U	Установить для выделенного фрагмента верхний регистр
NORMAL	
["x] p	Вставить текст из регистра “x” после курсора
["x] P	Вставить текст из регистра “x” до курсора
["x] y	Копировать текущую строку в регистр “x”
INSERT	

<CTRL> + R	Вставляет содержимое регистра. При нажатии этой комбинации в место вставки впечатывается символ кавычек и ожидается ввод идентификатора регистра. В качестве последнего могут выступать следующие символы: 0-9, a-z, ", %, #, *, +, :, ., -, =

По скольку при копировании и вставке фрагментов текста мы активно используем регистры, то давайте остановимся на них подробней. VIM поддерживает следующие виды регистров:

1. безымянный регистр **"**";
2. 10 нумерованных регистров от **"**0 до **"9**;
3. Регистр малого удаления **"-**;
4. 26 именованных регистров от **"a** до **"z** или от **"A** до **"Z**;
5. 4 регистра, доступных только для чтения: **":**, **".**, **"%**, **"#**;
6. регистр выражения **"=**;
7. регистры выделения **"*** и **"+**:
8. регистр “Черная дыра” – **"_**;
9. последний шаблон поиска **"/**.

Безымянный регистр используется при всех операциях копирования или удаления, если только в качестве целевого регистра не указана “Черная дыра”. В последнем случае удаленный фрагмент не копируется ни в дин из регистров. Если при вставке текста не указан регистр-источник, то в качестве источника используется безымянный регистр. Если в качестве целевого регистра при копировании указать безымянный регистр, то запись осуществляется в регистр **"0**.

Нумерованные регистры заполняются скопированными или удаленными фрагментами текста. При этом регистр **"0** содержит последний скопированный фрагмент, если не было указано целевого регистра. Регистр **"1** содержит последний удаленный фрагмент, если только не было указано другого регистра и если удаленный фрагмент содержит более одной строки текста. При удалении фрагментов размером менее строки, он копируется в регистр **"-**. При каждом успешном удалении фрагмент из регистра **"1** переносится в регистр **"2**, из **"2** в **"3** и так далее. Содержимое регистра **"9** безвозвратно теряется.

Именованные регистры записываются только по прямому указанию пользователя. При этом если указывается строчная буква, то фрагмент будет перезаписан, а если заглавная, то фрагмент будет добавлен.

Назначение остальных регистров очевидно по названию и указано в таблице

Специальные регистры:

Символ	Описание
<">	Безымянный регистр, содержащий последний удаленный или скопированный текст

Символ	Описание
<%>	Имя редактируемого файла
<#>	Альтернативное имя файла
<*>	Содержимое буфера обмена (X11 непосредственное выделение)
<+>	Содержимое буфера обмена
</>	Последний ключ поиска
<:>	Последняя команда
<.>	Последний вставленный текст
<->	Последнее “короткое” удаление (меньше строки)
<=>	Регистр выражения - вас попросят ввести выражение, результат вычислений будет вставлен в текст.

7. Работа с несколькими файлами и "окнами"

Конечно VIM позволяет работать с несколькими файлами одновременно. Я не буду подробно описывать все механизмы связанные с этой темой, а рассмотрю только наиболее полезные и простые операции.

Каждый открытый файл может либо отображаться, либо быть скрытым. Мы пропустим описание скрытых файлов, так как чаще всего встречается ситуация, когда необходимо именно отображать несколько файлов. Для этого мы должны рабочее окно разделить на два. По умолчанию в новое окно грузится тот же файл. Это может быть полезно, если необходимо иметь визуальный доступ к нескольким частям одного и того же документа. Однако вы можите в одно из окон загрузить новый файл. Интересной особенностью является то, что рабочее окно может быть разделено по вертикали или по горизонтали.

Команда	Описание
NORMAL	
:sp <CTRL> + W S <CTRL> + W <CTRL> + S	Разделить текущее окно на два по горизонтали. В результате вы имеете два вида на один файл. Первоначальное окно делится пополам.
:vs <CTRL> + W v <CTRL> + W <CTRL> + V	Разделить текущее окно на два по вертикали. В результате вы имеете два вида на один файл. Первоначальное окно делится пополам.
:new <CTRL> + W n <CTRL> + W <CTRL> + N	Создает новое окно (горизонтальное разделение), в котором начинается редактирование пустого (нового) файла.
:vne	Создает новое окно (вертикальное разделение), в котором начинается редактирование пустого (нового) файла.

8. Поиск, замена

VIM имеет очень мощные средства для поиска и замены текста. Эти механизмы основаны на так называемых регулярных выражениях. Если читатель знаком с этой технологией, то мои объяснения будут излишни, если нет – то это тема для отдельной статьи. Рассмотри наиболее употребимые варианты.

Простой поиск осуществляется с помощью команды "**/**" или "**?**" в нормальном режиме для поиска вперед или назад по тексту соответственно. После символа команды необходимо указать шаблон для поиска. Для повторного поиска необходимо нажать соответствующую команду (без шаблона). Удобной особенностью поиска является автоматическая подсветка найденных слов.

Кроме этого, есть очень удобная функция поиска слова на которое указывает курсор. Для этого необходимо в нормальном режиме установит курсор на требуемое слов и нажать # или * для поиска назад или вперед соответственно.

Для замены дного фрагмента текста (шаблон) на другой (новый текст) необходимо использовать следующую форму записи команды:

[range]s/шаблон/новый текст/[i][g][c]

где [**range**] – указывает где именно необходимо осуществить операцию замены. Возможны следующие варианты этого параметра: **%** – весь текст, **1** – первая строка, **.** – текущая строка, **$** –

последняя строка, **.,$** – от текущей строки и до конца файла, и так далее. Параметры в конце команды определяют ее поведение следующим образом: **i** – игнорировать регистр, **g** – заменять все слова в строке, иначе заменяется только первое найденное слово, **c** – подтверждать каждую замену.

Если вы вызвали команду замены из режима **VISUAL**, и опустили первый параметр, то замена будет осуществляться только в выделенной области.

9. Русские кодировки

Одной из проблем, с которой можно встретиться при редактировании документов, содержащих русский текст, является используемая кодировка. Для русского текста широко используются три основные кодировки (реально их больше, но остальные постепенно уходят в прошлое): Dos (альтернативная кодировка 866), Win (cp1251), Koi8-ru (Основная кодировка, используемая в семействе операционных систем Unix). Даже эти несколько разновидностей могут доставить значительно неудобствое в работе. Как вы уже догадались Vim поддерживает работу с различными кодировками. Более того, он поддерживает двухбайтную кодировку и различные направления письма. Хотя для нас это не играет особой роли.

Для поддержки различных кодировок необходимо добавить библиотеку iconv.dll (Для работы в Windows), которая распространяется отдельно. В современных дистрибутивах Linux все необходимое уже есть изначально.

За формат текстового документа отвечают два основных параметра: **fileformat** и **fileencoding** . Первый параметр определяет способ формирования строк в документе. Он может принимать следующие значения: dos (конец строки обозначается парой кодов <**CR**><**NL**>), unix (<**NL**>), mac (<**CR**>). Второй параметр определяет собственно кодировку. Она может принимать множество значений, но нас интересуют только несколько : **koi8-r**, **koi8-u**, **866**, **cp1251** и др.

Эти параметры можно установить для редактора глобально, при открытии файла, локально для буфера или для уже открытого файла. Как это все работает рассмотрим на примере. Допустим, у нас есть некий файл “dos.txt”. Он набран в MS Dos в соответствующей кодировке. Но мы этого не знаем и пытаемся открыть его на редактирование в ОС Linux. Для этого мы выполняем команду **:e dos.txt**. Файл открывается, и его формат распознан автоматически, о чем свидетельствует пометка [**dos**] в строке статуса, но кодировка к сожалению не распознана, в результате чего мы наблюдаем на экране абракадабру. Однако отобразить этот файл в нормальном виде не составит труда. Необходимо набрать команду **:e ++enc=866**, при этом русские буквы примут свой обычный вид, а строка статуса дополнится пометкой [**converted**]. Теперь мы можем работать с этим файлом без проблем. При сохранении его формат сохранится. Но если мы хотим преобразовать его в более удобный для нас формат, то достаточно набрать **:set fenc=koi8-r** и пометки в строке статуса исчезнут, а сохраненный файл будет в нашем выбранном формате и кодировке. Работа непосредственно с командами может показаться неудобной. И с этим можно согласиться, но если вы планируете часто выполнять подобные операции, то ничего не стоит добавить эти операции в графическое меню или назначить им горячие клавиши.

10. Макросы

VIM имеет очень мощные средства автоматизации работы. Для выполнения рутинных операций в текущем или группе файлов имеется встроенный язык. Этот язык достаточно функционален. Есть готовые решения создающие на базе VIM дневник с календарем или даже игру (Тетрис). Однако, описание встроенного языка выходит далеко за рамки данного труда и уж никак не относится к простым

приемам редактирования. Этот раздел добавлен с единственной целью немного приоткрыть штору **скрывающую** внутреннюю мощь этого текстового редактора.

Есть и более простой способ для повторения простых действий – линейный алгоритм, без ветвлений. Этот так называемые “сложные повторения” или “записи”. Эти макросы записываются непосредственно в именованные регистры. Для начала записи необходимо в режиме **NORMAL** нажать “q” и имя того регистра, в который необходимо осуществить запись. Окончание записи осуществляется повторным нажатием “q”. Для повтора записанных действий в режиме **NORMAL** используется команда “@” + имя регистра, а в режиме **INSERT - <CTRL>+R** + имя регистра.

Есть еще один метод быстрой подстановки фиксированных слов (словосочетаний или даже целых выражений). Этот метод использует так называемые аббревиатуры. Для описания своей аббревиатуры необходимо выполнить команду **:iabbrev** за которой следует аббревиатура и ее расшифровка соответственно. Пример: **:iabbrev M$ Microsoft**. После выполнения этой команды, если вы введете M$ с последующим пробелом, то автоматически будет подставлена соответствующая расшифровка “Microsoft”.

11. Группировка текста

И на последок я оставил самый, на мой взгляд, вкусный кусок. Особенно он понравится программистам. Думаю вам случалось писать объемные программы с множеством функций. Хорошим стилем считается группировка этих функций по отдельным файлам (классам при использовании ООП). Однако и в каждом файле может оказаться значительное количество кода. Навигация по этому файлу может быть затруднена. Не смотря на то, что VIM и без того имеет команды, позволяющие быстро отыскать нужную функцию или ее описание, в него еще встроена возможность временно скрывать часть текста, оставляя тем самым заголовки функций на виду и убирая непосредственно реализацию. При необходимости каждый (или все) фрагмент может быть открыт, отредактирован и закрыт. Такой подход делает программу более обозримой и удобной для восприятия.

Группировка может быть осуществлена одним из следующих способов:

- вручную;
- по левому отступу, больший означает глубину вложенности;
- по вычисляемому выражению, определяющему группировку;
- в соответствии с подсветкой синтаксиса;
- в соответствии с командой **diff** – скрываются одинаковые фрагменты текста;
- По специальным маркерам в тексте.

Рассмотри только первый способ группировки. Группировки, созданные вручную теряются при закрытии файла. Чтобы их сохранить необходимо выполнить команду **:mkview**. Восстановление сохраненных группировок осуществляется с помощью команды **:loadview**. Другие команды для работы со сгруппированным текстом приведены в таблице:

Команда	Описание
VISUAL	
zf	Группирует (скрывает) выделенный фрагмент текста
NORMAL	
zfap	Скрывает параграф, содержащий курсор
zf	Создать маркеры группировки
zd	Удалить маркеры группировки
zE	Удаляет все группировки в текущем окне (действует на группировки, созданные вручную и с помощью маркеров)
zo	Открыть текущую группировку
zc	Скрыть (закрыть) текущую группировку
zO	Открыть текущую группировку (рекурсивно)
zC	Скрыть (закрыть) текущую группировку (рекурсивно)
zM	Скрыть все группировки
zR	Открыть все группировки
INSERT	

12. Режим быстрой отладки

VIM поддерживает специальный режим работы ускоряющий цикл “редактирование – компиляция – редактирование”. Суть его заключается в том, что сообщения (ошибки и предостережения), генерируемые компилятором, сохраняются в файл и используются редактором, для быстрого перехода к месту возникновения ошибки. Используя этот режим вы получаете по сути удобную среду разработки не привязанную к конкретному языку и платформе. Для работы в режиме быстрой отладки могут быть полезны следующие команды:

Команда	Описание
NORMAL	
:make	Запуск сборки проекта.
:cc [номер]	Отображение ошибки с указанным номером
:cn	Перейти к следующей ошибке
:cp	Перейти к предыдущей ошибке
:cr	Перейти к первой ошибке
:cla	Перейти к последней ошибке
:cq	Выйти из программы с возвратом кода ошибки
:cf [имя файла]	Загрузить указанный файл в качестве файла ошибок
:cl	Отобразить все ошибки
:copen	Открыть окно со списком ошибок
:cclose	Закрыть окно со списком ошибок

Важной деталью работы в этом режиме является распознание формата файла ошибок. Очевидно, что различные компиляторы генерируют их поразному. Более того, в некоторых из них, этот формат можно задавать или выбирать. Для работы с конкретным компилятором необходимо настроить систему разбора файла с ошибками. Делается это с помощью команды **`:compiler {название}`**. По сути, при выполнении этой команды запускается скрипт, который настраивает параметры VIM на работу с указанным компилятором. Эти параметры можно легко настроить и вручную или написать свой скрипт.

13. Параметры, которые полезно знать и настраивать

Установка параметров осуществляется командой **`:set`**. Она имеет следующие варианты исполнения:

- **`:set`** – показывает все опции, значения которых отличаются от их стандартных значений;
- **`:set all`** – показывает все опции, кроме терминальных;

- **:set termcap** – показывает только терминальные опции;
- **:set {option}?** - показывает значение опции **{option}**;
- **:set {option}** – если **{option}** – логическая, то устанавливает ее в состояние “on”, иначе показывает значение;
- **:set no{option}** – устанавливает опцию в состояние “off”
- **:set {option}!** – инвестирует значение опции;
- **:set {option}&** – устанавливает значение опции в ее стандартное значение;
- **:set {option}={value}** – устанавливает значение опции равным **{value}**.

Параметр	Сокр.	Тип	Описание
autoindent	ai	триггер	При начале новой строки отступ копируется с предыдущей
autowrite	aw	триггер	Сохраняет содержимое файла, если он был изменен перед выполнением некоторых команд
backup	bk	триггер	Создает резервную копию файла при перезаписи. Оставляет ее после успешного окончания записи. Если вы не хотите оставлять резервную копию после успешного завершения записи, то сбросьте значение этого параметра в “off” и установите параметр writebackup
backupcopy	bkc	строка	Определяет метод создание резервной копии: “yes” - копирует существующий файл и перезаписывает старый; “no” - переименовывает существующий файл и создает новый, “auto” - выбирает наиболее оптимальный вариант.
binary	bin	триггер	Эту опцию необходимо установить перед попыткой редактирования бинарных файлов
dictionary	dict	строка	Список файлов, разделенных запятыми. По этим файла происходит поиск при авто-завершении ключевых слов (<CTRL>-X + <CTRL>-K).
display	dy	строка	Изменяет способ отображение текста. Представляет собой список флагов разделенных запятыми: “lastline” - если последняя строка, отображаемая в окне, не может быть показана полностью, то видна ее часть, иначе она заменяется символами “@” ; “uhex” - непечатные символы отображаются в шестнадцатиричном виде как <xx>, вместо использования ^C и ~C.
errorfile	ef	строка	Имя файла с ошибками в режиме быстрой отладки
errorformat	efm	строка	Описание формата строки файла с ошибками, записанное в стиле scanf
fileencoding	fenc	строка	Кодировку текущего файла
fileformat	ff	строка	Формат текущего файла

Параметр	Сокр.	Тип	Описание
foldclose	fcl	строка	Если равно “all”, то группировки автоматически закрываются, когда курсор выходит за их пределы.
foldmethod	fdm	строка	Определяет метод группировки для текущего окна: “manual” - ручная группировка; “indent” - группируются строки с одинаковым отступом ; “expr” - уровень группировки определяется по вычисленному выражению; “marker” - группировка осуществляется в соответствии с маркерами; “syntax” - группируются элементы с подсветкой синтаксиса; “diff” - группируется текст, не содержащий изменений
ignorecase	ic	триггер	Игнорировать регистр при поиске
makeef	mef	строка	Имя файла ошибок, возникших при выполнении команды :make. Усли не указано, то используется автоматически сгенерированное имя.
makeprg	mp	строка	Программа, используемая при выполнении команды :make.
modeline	ml	логич.	Включает непосредственное включение строки конфигурации в редактируемый файл.
shiftwidth	sw	число	Число пробелов, используемых при автоотступе.
showbreak	sbr	строка	Символы, которыми обозначаются автоматически перенесенные строки
showmatch	sm	триггер	Если установлено, то при вводе закрывающей скобки происходит быстрая подсветка соответствующей ей открывающей скобки.
tabstop	ts	число	Размер табуляции
textwidth	tw	число	Максимальное число символов, которые могут быть введены в одну строчку. Если установить равным 0, то данная функция отключается.
thesaurus	tsr	строка	Список файлов , по которым осуществляется поиск словарных статей. (<CTRL>+X + <CTRL>+T)
undolevels	ul	число	Максимальное число изменений, которые могут быть отменены
visualbell	vb	триггер	Заменить звуковой сигнал визуальным
wrap		триггер	Определяет отображение текста. Непосредственных изменений в открытый файл не вносится. Когда параметр установлен, то строки длиннее окна отображаются в следующих строчках и не обрезаются. Иначе видна только часть строки.
writebackup	wb	триггер	При перезаписи файла, создается его временная копия, таким образом, ни при каких обстоятельствах не могут пострадать ваши данный. После успешной перезаписи резервная копия удаляется.

14. Заключение

Я очень надеюсь, что этот документ оказался вам полезен, и вы узнали из него что-то новое. Для людей, которые только познакомились с VIM хочу заметить, что перечисленные и описанные здесь возможности – это только вершина айсберга. Действительно, VIM таит в себе огромную мощь. На его основе программист может легко организовать удобную среду разработки. Несмотря на то, что в среде профессиональных программистов, особенно *nix – ориентированных, этот редактор высоко оценен, я хотел бы привести еще несколько доводов в его пользу. Дело в том, что общественное признание VIM еще выражается и в том, что такие ведущие, в своих областях, интегрированные среды разработки как MSVisualStudio и KDevelop имеют возможность интеграции с VIM. На мой взгляд, это достаточно серьезный довод!

Тем не менее этот документ далек от идеала, ему нехватает примеров, не освещены очень многие варианты применения VIM. Возможно в нем содержатся некоторые ошибки как грамматические так и смысловые. То это не готовый продукт, а только первая версия. Поэтому я надеюсь, что заинтересованные люди примут участие в редактировании и дополнении данного документа. Ваши замечания и благодарности :-) можите отсылать по адресу i_yalovoy@mail.ru.