Редактор Vim и другие редакторы текста в Linux

А.Е. Шевель *
Петербургский Институт Ядерной Физики
188350, Гатчина, Ленинградской обл.,
Россия

3 апреля 2000 г.

Аннотация

В статье обсуждаются некоторые редакторы текстов в Linux. Бóльшее внимание уделяется редактору **Vim**.

В **Linux** широко используются около несколько десятков редакторов текста. Основным назначением любого редактора является помощь человеку при вводе и измененииии текста. Предполагается, что редактор позволяет минимизировать объём ручного ввода с клавиатуры, который не только утомителен, но, главное, является серьёзным источником ошибок. Редакторы текстов могут быть разделены на группы, или виды, по самым разным признакам.

Редакторы легко разделить на два вида: интерактивные редакторы текста, которыми пользуются при ручном вводе или изменении текста (большинство), и неинтерактивные редакторы текста, которые используются при редактировании текста без ручного вмешательства, например, **sed**. Можно различать редакторы, которые имеют интефейсы к внешним программам, например, **Perl**, **Tck**, **CVS**, **mail** и не имеющие таковых.

Решение, какими редакторами следует пользоваться, обусловливается, в основном, вашими потребностями. Однако, в каждом отдельном случае может оказаться интересен тот или иной редактор, в зависимости от контекста использования, включая психологические предпочтения того, кто пользуется редактором. Если ваши потребности таковы, что вам надо за день ввести и отредактировать пару записок длиной в одну страницу, или набросать пару скриптов по 30 строк, то вы запросто сможете обойтись простыми редакторами, например, **pico**. Если же вам приходится иметь дело с большими программами или описаниями, состоящими из многих модулей по несколько сотен строк, которые предполагают нетривиальное

*e-mail:*Andrei.Chevel@pnpi.spb.ru*

согласованное изменение многих фрагментов текста в разных модулях, то вам потребуются адекватные средства редактирования с элементами гипертекста и возможностями вызова приложений из среды редактора. Иными словами, сложность необходимого редактирования определяет сложность применяемого средства, т.е. редактора.

Часть редакторов могут использовать различные типы терминалов, как простейший **vt-100**, так и все средства **X/Window**. Другие не могут работать вне среды **X/Window**. Автору приходится часто "дрейфовать"с одной операционной платформы на другую, с одного типа терминала на другой, поэтому он предпочитает использовать для редактирования текста необходимый изобразительный минимум, который обычно достигается в окне **X/терминала** с эмуляцией **vt-100**.

Приведём краткие сведения по некоторым распространённым редакторам.

sed

sed строчный неинтерактивный редактор текста, который выполняет всю работу по редактированию в один проход по тексту и является обычным программным фильтром. В частности, благодаря однопроходности, **sed** работает быстрее многих других редакторов. Этот редактор удобно вызывать внутри скриптов. Обычно **sed** является частью любого варианта **Linux**.

pico

pico является простым редактором текста с небольшим набором команд редактирования, которым легко пользоваться на терминалах типа `vt-100`. Этот редактор является частью клиентской подсистемы электронной почты **pine**. Исходные тексты находятся в *ftp://ftp.cac.washington.edu/mail/pine.tar.Z*.

xedit

xedit - простой редактор текста с использованием возможностей **X/Window**. Подробнее можно посмотреть страницы `info xedit`. Имеется почти в каждом варианте **Linux**.

joe

Это довольно мощный текстовый редактор. Фактически **joe** является некоторым сборным редактором, который включает в себя привлекательные

модели работы с текстом заимствованные из различных редакторов: **pico**, **WordStar**, **emacs**.

Редактор **joe** имеется почти в каждом варианте **Linux**. Этот редактор можно найти в *ftp.std.com/src/editors/joe*.tar.Z*.

emacs и xemacs

emacs относительно новый (по сравнению с **vi**) мощный экранный редактор текста с массой встроенных функций по изменению текста, созданию командных последовательностей редактирования текста и взаимодействия с операционной системой. **emacs** можно считать настраиваемой средой для программиста, в которой имеются развитые средства редактирования текста.

xemacs является другим редактором (обсуждение отличий можно увидеть на странице *http://www.xemacs.org/About/XEmacsVsGNUemacs.html*), хотя **xemacs** начинал разрабатываться как ветвь **emacs**. Внешне отличия обусловлены модернизацией **emacs** главным образом в использовании графического интерфейса. Хорошим источником информации по **xemacs** является *http://www.xemacs.org/*. Оба редактора обычно являются частью многих вариантов **Linux**.

nedit

nedit - весьма развитый редактор текста для среды X/Window. Позволяет работать с использованием модели клиент/сервер. Список рассылки по обсуждению вопросов по редактору *mailserv@fnal.gov*. **nedit** фактически является свободно распространяемым продуктом с открытыми кодами. Он доступен по адресу: *ftp://ftp.fnal.gov/pub/nedit/*.

vi/vim

vi - один из старейших и наиболее мощных экранных редакторов (работает на терминалах типа **xterm**, **vt-100**). Обычно **vi** является частью любого варианта **Linux**.

По этому редактору текстов довольно часто рассылаются тексты с **FAQ** (Frequently Asked Questions, или ЧАсто задаваемые ВОпросы - **ЧАВО**) в news группе `comp.answers`. Архив news групп может быть найден в *http://www.faqs.org*.

Как сам **FAQ** по редактору **vi**, так и родственная информация может быть найдена в ряде мест

- *ftp://alf.uib.no/pub/vi*

- *ftp://ftp.uwp.edu/pub/vi*
- *ftp://ftp.uu.net/pub/text-processing/vi*
- *ftp://ftp.cc.monash.edu.au/pub/vi*
- *ftp://ftp.s.u-tokyo.ac.jp/misc/vi-archive*

Имеется более поздний усовершенствованный вариант редактора **vi** под именем **vim** (**V**i **IM**proved – усовершенствованный **vi**). **Vim** настолько улучшен в сравнении с **vi**, что имеет смысл говорить о том, что **Vim** – другой редактор, который правильно выполняет команды старого **vi** и имеет массу своих команд и возможностей. У **Vim** нет проблем с Кириллицей и другими алфавитами.

Редактор Vim

Обсуждаемый редактор отличается высокой скоростью работы, легко редактируются файлы размером в пару сотен тысяч строк. При этом имеется возможность работать как в алфавитно-цифровом режиме, так и в графическом. Во всех режимах поддерживается многооконная работа.

Vim имеет весьма богатый набор команд для просмотра и редактирования текстов любого вида включая двоичные файлы. Редактор удобен при работе с большим количеством текстовых файлов, которые необходимо корректировать согласованно. При этом файлы могут быть расположены как в одном каталоге так и в нескольких каталогах файловой системы. Для реализации этих возможностей используется несколько способов, таких как использование специальных таблиц ТЕГОВ (теговских файлов), а также использование механизма меток, которые могут быть установлены в любом месте любого файла и к которым позже можно просто перейти посредством ввода имени метки. Имеется возможность использовать теговские таблицы построенные для редактора **emacs**. Имеется возможность многоуровневой отмены изменения в тексте (многоуровневый undo).

Vim имеет интерфейсы для обращения к различным внешним программам и системам в том числе **SNiFF+** (*http://www.takefive.co.at/*), **cscope** и другим, а также к интерпретаторам **perl**, **python**, **tcl**.

Редактор имеет встроенный язык для описания поведения редактора в зависимости от различных меняющихся условий. На этом языке можно писать скрипты и позже использовать их в своей работе.

Vim позволяет автоматически настраиваться на заданные вами типы файлов, например, на редактирование исходных текстов различных языков

программирования или автоматически распаковать файл перед чтением и т.д. При этом редактор имеет широкий набор параметров (около 100), которые позволяют значительно изменять возможности по преобразованию текстов, включая автоматический сдвиг строк, форматирование комментариев в программах, нетривиальные преобразования текста как командами редактора, так и путём обращения к внешним программным фильтрам, а также многое другое. Поддерживается специальный режим быстрой отладки **quickfix**.

Поскольку обсуждаемый редактор является одним из наиболее мощных средств редактирования наравне с **emacs**, **xemacs** и другими, то в него встроено полное описание различных команд и возможностей. Страницы описания могут быть получены путём ввода команды редактора **:help**. **Vim** может быть использован для редактирования или первоначального ввода любого текстового файла (включая Кириллицу, фарси и другие языки).

Наконец, редактор постоянно улучшается, поэтому последние изменения полезно смотреть на сайте *http://www.vim.org/*.

Технические особенности редакторования текста в Vim

Редактирование может начинаться вводом команды

```
vim
```

При этом вызывается редактор **Vim** и никакие файлы не открываются. Начиная с этой точки можно посмотреть страницы документации по редактору посредством команды редактора `:help` или начать редактирование файла, что производится командой редактора

```
:edit filename
```

где `filename` есть имя файла, который вы предполагаете редактировать. По этой команде, текст из файла с именем `filename` будет прочитан во внутренний буфер редактора. Если редактируется несколько файлов, то внутренних буферов может быть несколько. Количество и тип буферов используемых в данный момент можно получить командой редактора

```
:buffers
```

Количество редактируемых файлов часто не совпадает с числом буферов. Какие файлы редактировались ранее и какие находятся в работе сейчас легко посмотреть командой редактора

```
:files
```

Кроме этого редактор поддерживает несколько специальных таблиц, которые помогают в быстрой ориентировке в большом количестве больших файлов. К таким таблицам отнесём таблицу скачков (перемещений) курсора. Редактор ведёт протокол перемещений курсора, которые производятся

командами редактора. Позже, вы можете воспользоваться данной информацией, чтобы вернуться в прежнее место. Эту таблицу легко посмотреть командой редактора

`jumps`

Аналогичным образом ведётся таблица меток, которыми вы можете отмечать определённые точки в тексте. Таблица меток просматривается командой

`:marks`

Vim ведёт таблицу (стек) тегов, которые вы использовали в своей работе. Редактор позволяет вам перемещаться по стеку использованных тегов. Стек тегов отображается командой

`:tags`

В редакторе имеются специальные области внутренней памяти, которые называются регистрами. Вы можете использовать регистры как механизм временного хранения частей текста (частей строк или частей статьи) или команд редактора.

Как любой редактор, **Vim** имеет команды как для выполнения простейших операций редактирования, так и более сложные команды преобразования текста. Простейшими командами легко воспользоваться для простых операций редактирования, например, вы начинаете редактировать существующий файл с именем `test`

`vim test`

После прочтения файла во внутренний буфер вы можете перейти в конец файла введя команду

`G`

Чтобы добавить новую строку к файлу, вы можете ввести команду

`o`

тем самым редактор перейдёт в режим ввода и вы можете вводить сколько угодно строк текста. Завершение ввода производится вводом одной из комбинаций:

`<ESC>` или `Ctrl/c` или `Ctrl/[`

Удаление одной строки производится командой

`dd`

Другие команды удаления элементов текста приведены в таблице 1.

Элементы текста, с которыми работает Vim

Редактор **Vim** может оперировать с различными элементами текста: СИМВОЛ, СЛОВО, ПРЕДЛОЖЕНИЕ, ПАРАГРАФ, РАЗДЕЛ. Здесь под термином

оперировать мы будем иметь в виду перемещение курсора пропуская определённое количество заданных объектов текста, удаление объектов и т.д.

СЛОВО представляет собой последовательность символов, которые не являются разделителями слов. Слова могут разделяться пробелами, знаками табуляции `<TAB>`, символами конца строки `<EOL>`.

ПРЕДЛОЖЕНИЕ,или фраза, (sentence) определяется как слово, или группа слов, за которым следует символ `.` (точка) или символ `!` (восклицательный знак) или символ `?` (вопросительный знак), за которыми следует конец строки или пробел. Любое количество закрывающих символов ограничителей `)` , `]`, `"`, `'` могут появиться после `.`, `?` или `!` до конца строки или пробела. Конец параграфа или раздела являются одновременно и концом предложения.

ПАРАГРАФ (paragraph) начинается после каждой пустой строки а также в точках, где имеются пары управляющих символов макроопределений параграфа, которые определяются параметром `paragraphs`

По умолчанию, `paragraphs=IPLPPPQPP LIpplpipbp`. Это соответствует управляющим макросам **nroff** `.IP` и `.LP`, которые должны быть в первой колонке.

РАЗДЕЛ (section) начинается после символа `<FF>` (Form Feed) в первой колонке и в точках макросов разделов, которые определяются парами символов в параметре `sections`. Эти управляющие символы также соответствуют командам **nroff**.

Команды редактора `]` и `[` останавливаются в положениях `{` и `}`, если они находятся в первой колонке. Это удобно для поиска, например, функции, в языке **C**. Более детальная информация может быть получена посредством `:h section`.

Удаление различных объектов текста

Ниже перечислены в качестве примера основные операции удаления с использованием лексических элементов текста. Операции производятся когда **Vim** находится в обычном (нормальном) режиме.

Таблица 1: УДАЛЕНИЕ ОБЪЕКТОВ ТЕКСТА

Удаление объектов текста	
Параметр	Значение
`dl` или `x`	удалить символ, на который указывает курсор.
`diw`	Удалить слово.
`daw`	Удалить слово вместе с последующим пробелом.
Продолжение на следующей странице	

Удаление объектов текста (продолжение)	
Параметр	Значение
`dd`	Удалить строку.
`dis`	Удалить фразу.
`das`	Удалить фразу вместе с обрамлением.
`dib`	Удалить внутренний блок `( )`.
`dab`	Удалить блок `( )` вместе с обрамлением.
`dip`	Удалить внутреннюю часть параграфа.
`dap`	Удалить параграф вместе с обрамлением.
`diB`	Удалить внутренний блок `{ }`.
`daB`	Удалить блок `{ }` вместе с обрамлением.

Некоторые режимы работы редактора

Редактор **Vim** функционирует в нескольких основных режимах: (полезно посмотреть страницы описаний по команде редактора `:help vim-modes`)

- командном, или обычном, (normal) режиме, в котором производится ввод команд редактора. В этом режиме находится редактор сразу после старта.
- В режиме ввода командной строки. В этот режим редактор переводится из обычного режима вводом следующих символов:
  - `:` (двоеточие), после которого может следовать любая команда, например `:help` или `:quit`.
  - `/` (наклонная черта) или `?` (знак вопроса), которые вводят операции поиска регулярных выражений.
  - `:!` (двоеточие и восклицательный знак), после которой следует фильтр, или любая команда **Linux**, например, `date`.
- В режиме ввода текста с клавиатуры вы можете перевести

  редактор из обычного режима несколькими способами, например, введя символ `i` При этом внизу экрана появляется надпись

  `--INSERT --`

  Режим ввода завершается нажатием клавиши `Esc` или вводом одной из комбинаций `Ctrl/[` или `Ctrl/c`. По завершении ввода текста с клавиатуры редактор переходит в командный режим.

  Ввод текста может производиться не только с клавиатуры, но и из файла, как например,

`cat File | vim -`

- Визуальный (visual) режим. Сразу после ввода режима любое перемещение курсора выделяет текст, прошедший под курсором. Когда вы отметили желаемую область текста, вы можете выполнить какие-то команды над выделенным текстом. Этот режим вводится несколькими способами:

  – вводом символа `v`, что означает начало ВИЗУАЛЬНОГО посимвольного режима, т.е. последующие операции выполняются над отмеченными цепочками символов.

  – вводом символа `V`, что означает начало ВИЗУАЛЬНОГО построчного режима, т.е. последующие операции выполняются над отмеченными строками.

  – вводом символа `Ctrl/V`, что означает начало ВИЗУАЛЬНОГО поблочного режима, т.е. последующие операции выполняются над отмеченным блоком.

  Завершение ВИЗУАЛЬНОГО режима производится выполнением команды над выделенной частью текста или отменой ВИЗУАЛЬНОГО, а также вводом символа `Ctrl/c`.

- Режим выбора (select) является вариантом ВИЗУАЛЬНОГО режима. Он вводится вводом символа `Ctrl/g` из ВИЗУАЛЬНОГО режима. При этом внизу экрана появляется надпись

  `--- SELECT ---`

  Полезно взглянуть `:help select-mode`

Имеются дополнительные режимы, которые могут использоваться в частных случаях.

- Режим ЗАМЕНЫ (replace mode) – представляет собой специальный случай режима ВВОДА. Этот режим вводится командой `R` из ОБЫЧНОГО режима. Здесь вы можете делать то же, что и в обычном режиме ввода, но каждый символ, который вы вводите будет замещать один символ в тексте. По умолчанию, внизу экрана появится надпись

  `-- REPLACE --`

- ВВОД КОМАНД В РЕЖИМЕ ВВОДА ДАННЫХ (insert command mode) – когда редактор находится в режиме ВВОДА ДАННЫХ (внизу высвечивается строка `-- INSERT --`), вы можете ввести символ `Ctrl/o`.

Редактор перейдёт в ОБЫЧНЫЙ режим (normal), но только для выполнения одной команды. Как только команда будет выполнена, редактор вернётся в режим ВВОДА ДАННЫХ.

- ВИЗУАЛЬНЫЙ РЕЖИМ НА ВВОДЕ (Insert Visual mode) – вводится когда редактор переходит из режима ВВОДА в ВИЗУАЛЬНЫЙ режим. Когда ВИЗУАЛЬНЫЙ режим будет завершён, то редактор перейдёт в режим ВВОДА ДАННЫХ.
- ВИЗУАЛЬНЫЙ РЕЖИМ ПРИ ВЫБОРЕ (Insert Select mode) – вводится когда происходит переход из режима ВВОДА в режим ВЫБОРА. Когда режим ВЫБОРА завершается, то редактор переходит в режим ВВОДА.

Редактирование текста производится в том месте, где стоит курсор. Перемещение курсора может производится различными клавишами и командами.

Редактор ПОНИМАЕТ сокращения имён команд. Если вы переводите редактор в режим ввода командной строки, то клавишами перемещения курсора можно посмотреть ИСТОРИЮ, т.е. ранее выполнявшиеся команды.

Переключение из режима в режим

Если вы не понимаете по каким-то причинам в каком режиме находится редактор, то вы всегда сможете перевести его в ОБЫЧНЫЙ режим дважды подряд введя комбинацию `<ESC>`. Если вы находитесь в СТРОЧНОМ КОМАНДНОМ режиме (**Ex**), то следует ввести команду редактора

```
:visual
```

Вызов редактора

Формат вызова редактора

vim [*options*] [*file ...*]
vim [*options*] -
vim [*options*] **-t** *tag*
vim [*options*] **-q** *errorfile*
ex
view
gvim | gview
rvim | rview | rgvim | rgview

Редактор **Vim** ведёт себя по разному в зависимости от имени, под которым он был вызван.

- **vim** - обычный режим работы.
- **ex** - работа в режиме редактора **ex**.
- **view** - просмотр файла в режиме ТОЛЬКО ЧТЕНИЕ. Эквивалентно использованию **vim -R**.
- **gvim | gview** - использование версии **GUI** (графического интерфейса). То же самое можно получить с помощью **vim -g**.
- **rvim | rview | rgvim | rgview** - почти то же самое, что предыдущее, с рядом ограничений. Главным образом, нельзя запускать команды оболочки из редактора, а также приостановить редактор. То же самое можно получить с использованием **vim -Z**.

Таблица 2: ПАРАМЕТРЫ ПРИ ВЫЗОВЕ РЕДАКТОРА **Vim**

Параметры при вызове редактора **Vim**	
Параметр	Значение
file...	Список имён файлов. Файл с именем, которое находится в списке первым слева, будет прочитан в буфер редактора **Vim** и станет текущим файлом. Курсор будет указывать на первую строку буфера. Вы сможете перейти к следующему по списку файлу выдав команду **:next**.
- (минус)	Файл для редактирования читается с устройства стандартного ввода. При этом команды считываются с устройства **stderr** (стандартное устройство для вывода диагностики об ошибках), которое должно быть назначено терминалу (tty). Этот параметр позволяет использовать **Vim** в конце цепочек программных каналов, например, `ls -ltaR | grep MyCatalogs | vim -`
Продолжение на следующей странице	

Параметры при вызове редактора **Vim** (продолжение)	
Параметр	Значение
-t *tag*	Файл для редактирования, а также начальная позиция курсора зависят от значения поля *tag*. Значение *tag* ищется в специальном файле ТЕГОВ. Ассоциированный с тегом *tag*, файл станет текущим файлом, а курсор будет установлен в позицию, которая также ассоциирована тегу *tag*. Это свойство удобно для редактирования программ, например, на языке **C**. Тогда в качестве тегов могут использоваться имена функций и других синтаксических элементов. В этом случае, имея заранее заготовленный файл ассоциаций с именем `tags` или `TAGS`, вызывая редактор с именем конкретной функции, например `vim -t Mul`, редактор сразу считает в буфер модуль содержащий функцию с именем `Mul`, а курсор будет установлен на начало функции. Теговский файл готовится с помощью программы **ctags** или **etags**.
+[*num*]	будет прочитан первый файл и курсор будет установлен на строке с номером *num*. Если значение *num* опущено, то курсор будет установлен на последней строке файла.
+/*pat*	будет прочитан первый файл и курсор будет установлен на строке, которая удовлетворяет выражению *pat*. Подробнее можно посмотреть в редакторе `:help search-pattern`.
+*command* **-c** *command*	команда редактора *command* будет выполнена после прочтения первого файла. Выражение *command* интерпретируется как команда редактора **ex**, который является составной частью **Vim**. Если команда *command* содержит пробелы, то она должна быть заключена в кавычки. Например, `+"set number" FileName` **Замечание**. Можно использовать до 10 команд + или **-c**.
-b	будут установлены несколько внутренних параметров, которые позволят редактировать двоичные или исполняемые файлы.
-h	выдать краткие ПОЯСНЕНИЯ к списку параметров, используемых в командной строке, и завершится.
Продолжение на следующей странице	

Параметры при вызове редактора **Vim** (продолжение)	
Параметр	Значение
-N	режим несовместимости с **vi**. Редактор **Vim** начинает вести себя лучше, но несколько иначе, чем старый редактор **vi**.
-o[*N*]	открыть *N* окон. Если *N* опущено, то открыть одно окно для каждого файла.
-r **-r** *filename*	восстановить состояние редактируемого файла после аварийного завершения сессии редактирования. Файл восстанавливается из своп-файла, который имеет имя *filename*.swp, где *filename* есть имя редактируемого файла. Если *filename* опущено, то редактор **Vim** выдаёт информацию о всех свопированных файлах.
-s	запустить редактор **Vim** в МОЛЧАЛИВОМ режиме, т.е. с минимумом диагностических сообщений. Используется для запуска **ex** в пакетном режиме.
-s *scriptin*	читается файл с именем *scriptin*. Символы из этого файла интерпретируются как буд-то они были бы введены с клавиатуры. То же самое можно сделать командой редактора `:source! scriptin`
-u *vimrc*	использовать инициализационный файл с именем *vimrc*. При этом все другие виды инициализации опускаются. Этот же параметр можно использовать, чтобы обойти любые виды инициализации: `-u NONE` Детали можно посмотреть посредством команды редактора `:help initialization`
-V	Разговорчивый режим (VERBOSE). Сообщаются имена файлов, которые были использованы для инициализации и другая информация.
Продолжение на следующей странице	

Параметры при вызове редактора **Vim** (продолжение)	
Параметр	Значение
-w *scriptout*	все что вы вводите с клавиатуры будет копироваться в файл с именем `scriptout` до тех пор пока вы не выйдете из редактора. Основное назначение данной возможности – подготовить текст для того, чтобы использовать его в команде редактора **Vim** `:source!` или вместе с параметром **-s**. Повторное использование того же имени файла приведёт к дописыванию в конец файла. Смотрите также страницы документов по **Vim** `:h complex-repeat`
--version	печатается версия вашего редактора **Vim** и много другой полезной информации.

Специальная возможность quickfix

Редактор **Vim** поддерживает сценарий работы в цикле РЕДАКТИРОВАНИЕ - КОМПИЛЯЦИЯ - РЕДАКТИРОВАНИЕ. Хотя такая мода работы использовалась первоначально для ускорения доводки исходных текстов программ, чтобы они быстрее проходили стадию компиляции, однако этот же сценарий можно использовать и в других случаях, например при использовании системы TeX/LaTeX, а также других.

Чтобы упростить использование такого сценария, **Vim** содержит специальные средства: команду `:make` и параметр `errorformat`, описывающий формат диагностических сообщений программы (компилятора или другой программы). Стиль описания формата сообщений программы напоминает стиль описания формата вводных данных для программы **scanf** в языке **C** или описание вводных данных в другом языке программирования. Параметр `errorformat` содержит обычно, довольно длинную строку описания формата сообщений, например,

```
errorformat=%*[^"]"%f"%*\D%l: %m,"%f"%*\D%l: %m,%f:%l:%m,"%f"\,
line %l%*\D%c%*[^ ] %m,%D%*\a[%*\d]: Entering directory ‘%f’,
%X%*\a[%*\d]: Leaving %directory ‘%f’,%DMaking %*\a in %f
```

Вышеприведённая строка является непрерывной одной строкой, которая разбита в соображениях только типографских.

Редактор позволяет посредством описания формата определять виды сообщений программы и отделять одно сообщение от другого, переходить

от одного сообщения к другому при просмотре, а также выполнять дригие операции. **Vim** поддерживает несколько файлов с диагностическими сообщениями (до 10), что позволяет сравнивать прежнюю диагностику с имеющейся. Ниже приведены команды перехода от одного сообщения к другому, а также переходы от одного файла с диагностическими сообщениями к другому.

Таблица 3: Описание команд просмотра диагностики

Описание команд просмотра диагностики	
Команда	Значение
`:cc [`*nr*`]`	Показать диагностическое сообщение с номером *nr*. Если *nr* опущено, то отображается то же самое сообщение, которое уже было на экране.
`:cn[ext]`	Отобразить следующее диагностическое сообщение из списка, который включает имя файла. Если нет никаких имён файлов, то перейти к следующему по порядку сообщению.
`:cN[ext]` или `:cp[revious]`	Отобразить предыдущее диагностическое сообщение, которое включает имя файла. Если нет никаких имён файлов, то перейти к предыдущему по порядку сообщению.
`:cnf[ile]`	Отобразить первое диагностическое сообщение в следующем файле в списке, который включает имя файла. Если нет никаких имён файлов в списке или нет следующего файла, то перейти к следующему по порядку диагностическому сообщению.
`:col[der]`	Перейти к более старому файлу с диагностическими сообщениями.
`:cnew[er]`	Перейти к более свежему (новому) файлу с диагностическими сообщениями.
`:cr[ewind] [`*nr*`]`	Отобразить диагностическое сообщение номер *nr*. Если *nr* опущено, то отображается ПЕРВОЕ диагностическое сообщение.
Продолжение на следующей странице	

Описание команд просмотра диагностики (продолжение)	
Команда	Значение
`:cla[st]` [*nr*]	Отобразить диагностическое сообщение номер *nr*. Если *nr* опущено, то отображается ПОСЛЕДНЕЕ диагностическое сообщение.
`:cq[uit]`	Завершить **Vim** с ненулевым кодом завершения, таким обарзом компилятор (или другая программа) не будет обрабатывать тот же самый файл снова.
`:cf[ile]` [*errorfile*]	Читать файл, содержащий диагностические сообщения, с именем *errorfile* и перейти к первому диагностическому сообщению. То же самое будет выполнено редактором автоматически, если редактор запускается с параметром **-q**. Вы можете использовать комаду `:cf[ile]`, если редактор продолжает работать, когда вы выполняете компиляцию. Если вы задаёте в команде параметр *errorfile*, то параметр редактора `errorfile` будет установлен в значение *errorfile*.
`:cl[ist]` [*from*] [, [*to*]]	Перечислить все диагностические сообщения, которые включают имя файла. Если числа *from* и/или *to* присутствуют, то отображается соответствующий диапазон диагностических сообщений.
`:cl[ist]!` [*from*] [, [*to*]]	Перечислить все диагностические сообщения в заданном диапазоне.
`:mak[e]` [*arguments*]	Команда выполняется следующим образом.
Продолжение на следующей странице	

Описание команд просмотра диагностики (продолжение)	
Команда	Значение
	1. Если внутренний параметр редактора `autowrite` включён, то записать на диск все изменённые буферы. 2. Стартует программа указанная в параметре редактора `makeprg` (по умолчанию имя программы `make`) с возможными параметрами *arguments*. Вывод программы будет сохранён в файле, имя которого хранится во внутреннем параметре редактора `makeef`. 3. Файл с именем, которое содержится во внутреннем параметре редактора `makeef`, прочитывается редактором и отображается первое диагностическое сообщение.
`:gr[ep]` [*arguments*]	Общий сценарий такой же как у команды `:mak[e]` (см. выше), но имя программы хранится во внутреннем параметре **Vim** `grepprg`. Используются также соответствующий внутренний параметр редактора для описания формата диагностических сообщений – `grepformat`.

Более детальную информацию можно почерпнуть в редакторе на страницах

```
:h quickfix
```

Получение описания команд

Для получения полезной информации из редактора **Vim** следует ввести команду `:help` (ввести двоеточие затем слово **help** или просто **h**). Например, введя `:h ZZ`, вы увидите пояснение:

Записать текущий файл, если он был модифицирован, и выйти из редактора (то же самое, что команда редактора `:x`*). Если для данного файла было открыто несколько окон, то файл, если был модифицирован, записывается на диск и одно окно закрывается.*

Часто описание содержит термины, которые описываются в отдельных главах описаний. Например, вы можете увидеть:

```
1. Vim arguments                    |vim-arguments|
2. Vim on the Amiga                 |starting-amiga|
3. Initialization                   |initialization|
```

Если установить курсор между двумя вертикальными чёрточками, к примеру, на слове INITIALIZATION и ввести `Ctrl/]`, то произойдёт переход к описанию термина INITIALIZATION. Если позже вы захотите вернуться к первоначальному описанию, то следует ввести символ `Ctrl/t`.

Можно использовать автоматическое завершение команд. Например, введя `:h cm`, вы можете нажать `<TAB>` или `Ctrl/d` чтобы увидеть один из вариантов завершений. Введя снова `<TAB>` или `Ctrl/d`, вы увидите очередной вариант завершения. Детали менанизма завершения можно почерпнуть в `:h ins-completion`. Перед началом использования редактора полезно посмотреть

`:help starting`

Метки в тексте

В тексте можно устанавливать метки, которые позже могут быть использованы для перевода курсора на ту или иную метку. Метки не видны. Это лишь запомненные позиции курсора в тексте. Не следует путать метки с регистрами, это совершенно разные сущности.

Имена меток состоят из одной буквы или одной цифры, которой предшествует одиночный апостроф. Различают несколько видов меток:

- `'a - 'z` – строчные метки, устанавливаются и действуют в пределах одного файла;
- `'A - 'Z` – прописные метки, устанавливаются и действуют во всех редактируемых в данный момент файлах;
- `'0 - '9` – нумерованные метки, устанавливаются в файле `.viminfo`.

Метки, обозначаемые строчными буквами, запоминаются и хранятся до тех пор пока редактируемый файл находится в буферах редактора. Если вы удаляете файл из буферов редактора, то метки теряются. Если вы удалите строку в файле, на которую указывала метка, то эта метка теряется. Однако метки могут быть восстановлены, если вы выполнили команды редактора **undo** или **redo**.

Строчные имена меток можно использовать в командах редактора. Например,

`d'm` – означает удалить строки от позиции курсора до метки с именем `m`.

Вы можете, например, установить метки командами `:ma t` для первой строки редактируемого файла и `:ma b` для последней строки редактируемого файла. Позже можно быстро переходить к концу файла вводом комбинации `'b` в обычном режиме редактора. Таким же образом вы сможете перейти к началу текущего файла командой `'t`.

Прописные имена меток `'A - 'Z` запоминаются редактором вместе с именем редактируемого файла. Таким образом вы может перейти из одного файла в другой. В операторах редактирования можно использовать такие метки, если только они указывают на тот же файл. Если параметр **viminfo** не пуст, то метки верхнего регистра хранятся в файле `.viminfo`. Значение метки верхнего регистра будет поддерживаться верным, даже если вы удалите или добавите в файл какие-то строки в файл, но не удалите строку, на которую указывает метка.

Нумерованные метки `'0 - '9` заметно отличаются от остальных. Они не могут быть установлены непосредственно с помощью команд редактора и используются в основном в файле `.viminfo` (когда параметр редактора `viminfo` не пуст). Когда записывается файл `.viminfo` (во время выхода из редактора или по команде `:wviminfo`), то `'0` устанавливается в текущий файл и текущую позицию курсора. При этом старое значение метки `'0` перемещается в метку `'1`, а старое значение `'1` перемещается в `'2` и так далее. Таким образом, `'0` указывает на позицию курсора, которую он имел в момент выхода из редактора. Чтобы вернуться, например, на другой день к прежней точке файла надо выполнить команду оболочки:

```
vim -c "normal '0"
```

Значения меток могут быть установлены рядом команд, которые выполняются когда редактор находится в обычном (командном) режиме. Например,

`mb`

установить одну метку в том месте, где стоит курсор, с именем `b`.

Позже можно перейти к метке с именем `b` по команде

`b

Аналогично используются метки с именами прописных букв.

Регистры

Редактор **Vim** имеет специальный вид внутренних буферов, которые называются РЕГИСТРАМИ. Имеется несколько видов регистров.

1. Безымянный регистр "".
2. Десять нумерованных регистров от "0 до "9.
3. Регистр удаления "-.
4. Именованные регистры от "a до "z (или от "A до "Z).
5. Четыре регистра только для чтения: ", ". % "#.
6. Регистр выражений (expression) "=.
7. Регистр выбора (selection) "*.
8. Регистр чёрная дыра (black hole) "_.

Безымянный регистр

Безымянный регистр заполняется командами удаления текста такими как **d**, **r**, **s**, **x**, **c**, а также **yank**. Причём заполнение данного регистра происходит независимо от того используется или нет в команде именованный регистр. Иными словами, при удалении текст не выбрасывается, а перемещается в безымянный регистр. Исключение составляет регистр ЧЁРНАЯ ДЫРА. Если этот регистр использован в команде удаления текста, то текст выбрасывается безвозвратно.

Вы можете использовать содержимое БЕЗЫМЯННОГО РЕГИСТРА использовав имя ".

Нумерованные регистры

Регистр номер 0 содержит текст, который был удалён наиболее поздней командой удаления текста, если только в самой команде не был явно указан другой регистр. Регистр с номером 1 содержит текст, удалённый предыдущей командой **d**, **r**, **s**, **x**, **c** или **yank**. Регистр с номером 2 содержит текст, удалённый ещё более ранней командой и т.д.

Если удаляемый текст содержит менее одной строки, то при удалении он помещается в РЕГИСТР УДАЛЕНИЯ.

При очередном удалении текста **Vim** помещает в регистр 0 удаляемый сдвинув перед этим содержание остальных регистров, т.е. регистр 8 копируется в регистр 9. регистр 7 копируется в регистр 8 и т.д. Естественно, что прежнее содержание регистра 9 теряется.

Регистр удаления

Этот регистр служит для сохранения удаляемого текста, если он занимает меньше одной строки и в команде удаления не указан явно другой регистр.

Именованные регистры

Vim заполняет эти регистры только в том случае, если вы явно попросили это делать. Если использовано обозначение имени регистра строчной буквой, то содержимое регистра будет замещено новой информацией. Если имя регистра обозначено прописной буквой, то новая информация будет дописана вслед за уже имеющейся в регистре.

Например, команда редактора (в обычном режиме) `2"xdd` приведёт к следующим действиям:

- из текущего файла будут удалены две строки начиная с текущей строки (на которую указывает курсор);
- эти две строки будут помещены в регистр с именем `x`, заместив предыдущее содержание регистра;
- те же две строки будут помещены в БЕЗЫМЯННЫЙ РЕГИСТР и они же будут помещены в регистр с номером 0.

Продолжим пример. Если позже вы захотите дописать в тот же регистр (с именем `x`) новые пять строк, то это можно сделать следующей командой редактора (в обычном режиме) `5"Xyy`. При этом пять строк просто копируются (дописываются в данном случае) в регистр с именем `x`. Позже, содержимое регистра с именем `x` можно записать в тот же самый или другой файл командой `"xp`. Например, вы можете открыть новое окно командой `Ctrl/w` в этом новом окне выполняете команду `"xp`. Далее, содержимое окна легко записать в файл на диск командой `:w file_name`, где `file_name` есть имя файла.

Регистры только для чтения

Эти регистры содержат информацию предназначенную только для чтения.

- ". (двойной апостроф и точка) – содержит информацию, которая была введена в режиме ввода, например по команде `i` или `o`
- "% (двойной апостроф и знак процента) – содержит имя текущего файла.
- "# (двойной апостроф и решётка) – содержит имя альтернативного файла, если таковой имеется.
- ": (двойной апостроф и двоеточие) – содержит последнюю командную строку, которая была введена с использованием двоеточия, например,

  `:help registers`.

  Если вы желаете повторить выполнение последней команды, то можно ввести комбинацию `@:`.

Чтобы лучше понять алгоритм изменения в различных ситуациях, полезно поэкспериментировать с содержанием регистров. Содержание всех регистров можно посмотреть командой `:reg`.

Регистр выражений

Этот регистр не является обычным регистром для запоминания информации, а является способом использования выражений в командах редактора, там где обычно используется регистр. Регистр выражений является регистром только для чтения, т.е. вы не имеете возможности изменить содержимое этого регистра обычными командами, которыми мы изменяли содержимое других регистров в ранее приведённых примерах. Заполнение регистра производится следующим образом. Вы вводите "=. После ввода знака равенства курсор переместится в командную строку, где вы сможете ввести выражение. Ввод выражения завершается возвратом каретки (<CR>). Чтобы аннулировать ввод выражения, введите <ESC>.

В качестве простого примера использования Регистр выражений рассмотрим следующий случай. Пусть нам необходимо определить максимальное количество символов помещающихся в видимой части окна редактора. Тогда мы могли бы предпринять следующую последовательность действий.

Вводим команду "= курсор перейдёт в командную строку. Далее, вводим в командной строке `&winheight*&columns` и курсор возвращается на прежнее место. Теперь если сразу ввести команду `p`, то в текст будет вставлено произведение этих двух параметров, что в моём случае равно 200, т.е. 25*80.

Регистр выбора

Этот регистр используется запоминания и поиска выбранного текста в режиме **GUI**.

Регистр черная дыра

Когда что-то записывается, то ничего не происходит с остальными регистрами, т.е. не оказывается никакого влияния. Если производится попытка чтения из этого регистра, то ничего не считывается.

Примеры использования регистров

Пример переноса текста из одного места (файла) в другое (другой файл).

Для выполнения такого переноса проще всего воспользоваться следующим методом.

- Отметить необходимую часть текста, например, посредством `V` (построчное выделение текста).
- Удалить выделенный текст, например, командой редактора `"ad`, которая удалит выделенный текст и скопирует его в регистр с именем `a`.
- Переключиться на другой файл любым способом, например, командой редактора `:n` или создать новое окно посредством команды `Ctrl/wn` и перейти в него.
- В новом месте выполнить команду `"ap`.

Такой перенос можно, конечно, выполнить и без использования регистров, особенно, если перемещение текста производится в пределах одного файла. Для этого можно использовать команды редактора `:copy`, `:move`,

Программа etags

Программа **etags** предназначена для формирования теговского файла для редактора **emacs** по умолчанию используется имя теговского файла `TAGS`. Обычно теговский файл для **Vim** имеет несколько отличный формат и стандартное имя `tags`. Заметим сразу, что редактор **Vim** имеет возможность использовать оба вида теговских файла (достаточно, чтобы редактор был скомпилирован с установленным парамтром `+emacs_tags`).

Результирующие теговские файлы помещаются в текущий каталог.

В этом разделе мы обсудим часть свойств программы **etags**, которая выводит в ответ на команду системы:

`etags --version`

ответит следующими строками:

```
etags (GNU Emacs 20.4)
Copyright (C) 1996 Free Software Foundation, Inc. and Ken Arnold
This program is distributed under the same terms as Emacs
```

Программа способна строить теговский файл для списка языков, который можно увидеть командой:

`etags --help`

Ниже перечислены часть языков, которые ПОНИМАЕТ программа **etags**.

- **C/C++**;
- **Fortran**;
- **Pascal**;
- **LaTeX**;
- **Scheme**;
- **Emacs Lisp/Common Lisp**;
- **Erlang**;
- **Prolog**;
- а также большинство ассемблеро-подобных языков.

Формат использования программы таков:

```
etags [-aCDRSVh] [-i file] [-l language] [-i regexp]
      [-o tagfile] [--c++] [--no-defines] [--ignore-indentation]
      [--language=language] [--regex=regexp] [--no-regexp]
      [--help] [--version] [--include=file] [--output=tagfile]
      [--append] file ...
```

Имена файлов представленные в относительном виде будут записаны в теговском файле как относительные файлы, а имена представленные в абсолютной форме будут записаны в теговском файле как абсолютные файлы. Язык, на котором написаны исходные тексты определяется автоматически по окончанию имени файла, а аткже по содержанию файла, если не указан иной способ определения языка. Иной язык может быть указан значением параметра `--language`.

Таблица 4: Параметры программы **etags**

Параметры программы **etags**	
Параметр	Значение
`-l`*language*, `--language=`*language*	Проанализировать файлы в соотвествии с синтаксисом указанного языка *language*. Можно использовать более, чем один параметр `--language=` для разных файлов в одной командной строке. Возможные значения величины *language* следует получить посредством команды `etags --help` Значение `auto` для величины *language* устанавливает автоматическое определение типа исходного текста. Значение `none` может быть использовано для отключения языкового анализа, при этом будут действовать параметры `--regex`.
`-o` *tagfile*, `--output=`*tagfile*	определить имя теговского файла, заменив умолчание `TAGS`.
Продолжение на следующей странице	

Параметры программы **etags** (продолжение)	
Параметр	Значение
`-r` *regexp*, `--regex=`*regexp*	Создать теги на основе регулярных выражений *regexp* соответствующих строкам в исходных файлах. Эти теги создаются в дополнение к тегам, которые будут создаваться на основе языкового анализа. Можно использовать более, чем один параметр `--regex=`. Параметры могут быть перемешаны с именами файлов в командной строке. Все параметры `--regex=` кумулятивны, т.е. каждый очередной параметр `--regex=` в командной строке добавляет тегов к предыдущим параметрам `--regex=`. Если регулярное выражение достаточно длинно, то может оказаться удобным использовать имя регулярного выражения вместо самого выражения, например: `--regex='/параметр|значение/nn/'` Здесь `nn` можно далее использовать как имя тега.

Использование таблицы тегов

Часто оказывается, что в текстах программ, которые вы редактируете довольно много одинаковых тегов. Ниже приведены команды **Vim**, которые не изменяют стек тегов, а позволяют перемещаться по стеку.

Таблица 5: Команды для перемещения по тегам

Команды для перемещения по тегам	
Команда	Значение
`:ts[elect][!]` [*ident*]	Перечислить теги, удовлетворяющие *ident*, использовав для этого файл(ы) тегов. Если *ident* опущен, то используется последнее имя тега из стека тегов.
Продолжение на следующей странице	

Команды для перемещения по тегам (продолжение)	
Команда	Значение
`:sts[elect] [!]` [*ident*]	Выполнить `:ts[elect] [!]` [*ident*] и разделить текущее окно для выбранного тега. Иными словами в каждом окне будет своё положение тега, например, в первом – тег из строки 17, а во втором тег с тем же именем из строки 237.
`:tj[ump] [!]` [*ident*]	то же что `:ts[elect] [!]`, но переход к единственному тегу будет выполнен сразу.
`:stj[ump] [!]` [*ident*]	то же что `:tj[ump] [!]` [*ident*], но в дополнение окно разбивается на два окна, в одном из которых курсор указывает на тег с именем *ident*.
:[*count*]`tn[ext] [!]`	Перейти вперёд к *count*-ой встрече тега с текущим именем. По умолчанию к следующей встрече.
:[*count*]`tp[revious] [!]`	Перейти назад к *count*-ой встрече тега с текущим именем. По умолчанию к предыдущей встрече.
:[*count*]`tN[ext] [!]`	то же что :*count*`tp[revious] [!]`
:[*count*]`tr[ewind] [!]`	Перейти к первой встрече тега с текущим именем (если *count* опущено). Если *count* присутствует, то перейти *count*-ой встрече (от начала) тега с текущим именем.
`:tl[ast] [!]`	Перейти к последней встрече тега с текущим именем.

Сценарии использования редактора vim

Поскольку многие возможности редактора не являются тривиальными, то здесь мы перечислим несколько сценариев, где использования данного редактора будет достаточно эффективным. Начнём с простых примеров.

Общие замечания

Этот редактор довольно быстро работает с длинными текстами (например, 100 тысяч строк). что позволяет использовать редактор для просмотра различных логов (т.е. файлов протоколов), почтовых ящиков для поиска каких-либо ключевых слов и выражений, других больших файлов. Например,

для просмотра лога сообщений **Linux** можно использовать редактор **Vim** под кличкой **view**.

Просмотр лога системных сообщений /var/log/messages

`view /var/log/messages`

Положим, вы прочитали файл `/var/log/messages`. Теперь вы сможете получить о нём массу информации:

- `Ctrl/g` покажет имя файла и количество строк;
- `g-Ctrl/-g` покажет положение курсора, а также количество строк и количество символов в файле, т.е. объём файла.

Если вы захотите посмотреть список строк где встречается определённая подстрока, то укажите на неё курсором затем введите:

- `[I` – перечислить встречи подстроки во всём файле или
- `]I` – перечислить встречи подстроки начиная с позиции курсора до конца файла.

Параметры редактора

Перед началом использования редактора **Vim** полезно посмотреть с какими параметрами вызван редактор. Командой

`:set`

вы сможете увидеть все параметры, чьи значения отличаются от встроенных умолчаний. Значения всех остальных переменных можно также увидеть посредством команды редактора `:set`.

Часть команд зависят от параметров, значения которых устанавливаются во время трансляции редактора **Vim**. Значения таких параметров можно получить командой редактора `:version`.

Редактирование нескольких файлов одновременно

В ряде ситуаций, когда вы отлаживаете программу из нескольких подпрограмм или готовите описание из нескольких отдельных глав, при внесении изменений в один модуль, требуется вносить согласованные изменения в другие модули. В такой ситуации удобно сразу указать редактору имена всех модулей. Сделать это очень просто:

`vim mod_1 mod_2 ... mod_N`

При этом можно указать желаемое количество окон, которые организует **Vim**. Можно открыть хоть дюжину окон, но эффективнее использовать 2-3 окна. Тогда команда будет выглядеть так:

```
vim -o3 mod_1 mod_2 ... mod_N
```

В каждом окне появится файл из списка параметров. Поскольку были запрошены три окна, то **Vim** поместит в них первые три файла из списка параметров. Когда редактор уже вызван, список параметров можно посмотреть командой

```
:ar[gs]
```

С помощью этой же команды вы сможете перейти к редактированию других файлов, например:

```
:ar[gs] mod_12 mod_19
```

Переместиться по списку параметров вперёд можно командой:

```
:n[ext]
```

а назад – командой

```
:N[ext] или :prev[ious]
```

Перейти к любому файлу в списке аргументов можно командой:

```
:argu[ment] n
```

здесь `n` означает номер файла в списке параметров слева направо.

Переход к первому файлу в списке выполняется командой

```
:rew[ind]
```

а к последнему – командой

```
:la[st]
```

Часто при внесении изменений требуется смотреть в несколько описаний функций или структур, которые находятся в других местах файла или вообще в разных файлах. Быстрый переход от одной точки в одном файле к другой точке в другом файле проще всего реализовать с использованием механизма меток.

Например, командой редактора

```
:ma[rk] W
```

вы можете отметить вашу рабочую точку меткой `W` (вообще можно использовать любую прописную букву). То же самое вы можете реализовать посредством команд `:k W` или `m-W` Тогда переход из любого файла к рабочей точке, где вы модифицируете текст, мог бы быть выполнен посредством команды: `'W` или `‘W`. Если вы при этом отметили интересующую вас таблицу командой `m-T` или `:ma[rk] T`, то после этого вы можете переходить от таблицы к точке ввода сколько угодно раз. Кстати, точку ввода можно и не отмечать, если вы только хотите взглянуть таблицу и сразу вернуться. Вы можете перейти к таблице по команде `'T`, а возврат произвести по команде

''.

Другой способ быстрого перемещения курсора по файлам проекта можно реализовать с помощью команд редактора, которые используют специальную таблицу синтаксических элементов текста ТЕГОВ. Как мы помним, таблица для программ на языках **C** и **C++** строится программой **tags**. Например, по команде

```
:ta[g] function
```

будет произведено перемещение курсора в тот файл и ту точку, где находится тег с именем `function`. Если вам потребуется получить полный список точек, где встречается в тексте тег с именем `name`, вам следует использовать команду:

```
:ts[elect] name
```

Можно использовать альтернативный вариант тех же команд. Если вы введёте в нормальном режиме `Ctrl-]`, то это будет эквивалентно команде `:ta[g] word`, где `word` есть слово, на которое указывает курсор. С другой стороны, ввод комбинации `g]` в нормальном режиме полностью эквивалентен команде `:ts[elect] word`, где `word` есть слово, на которое указывает курсор.

Вызвать программу формирования таблицы тегов и построить таблицу тегов, т.е. индексную таблицу, можно перед вызовом редактора или во время редактирования. Большая часть доступных программ построения таблицы тегов является ориентированными на популярные языки **C**, **C++**, **Java**, **fortran** и т.д.

Допустим, вы желаете быстро установить в какой программе (файле) встречается определённое регулярное выражение и перейти к редактированию этого файла. Это можно сделать командой:

```
:grep "популярные" *.tex
```

Команда передаст все параметры программе **grep**, которая просмотрит указанные файлы (как видно из примера, файлы не обязаны быть только программами) и загрузит первый файл, в котором найдено слово `популярные`. Далее можно перемещаться по списку файлов с помощью команд просмотра `:cnext`, `:clist` и т.д. Подробности следует смотреть с помощью страниц руководства, которое встроено в **Vim**, например:

```
:h clist
```

Другие способы перемещения по текстам

Пусть вы редактируете (или просматриваете) множетсво файлов, состоящее, например, из нескольких десятков файлов. Тогда, можно быстро перемещаться по файлам и отдельным строкам с помощью таблицы скачков и с помощью таблицы буферов, в которых хранятся файлы.

Так, все перемещения курсора по файлу (файлам) записываются в специальную таблицу скачков, содержание которой можно посмотреть командой редактора `:ju[mps]`. Далее, по таблице можно перемещаться (т.е. перемещаться по файлу или файлам) используя команды редактора `[n]Ctrl/i` – перейти по таблице к более недавним положениям курсора и `[n]Ctrl/o` – перейти к более старым положениям курсора. Здесь `n` есть число строк таблицы скачков, на которое следует переместиться по таблице скачков (умолчание = 1). Этот способ перемещения действует как в пределах одного файла, так и с использованием множества файлов.

Если вы желаете переместиться из одного редактируемого файла в другой, ранее редактируемый файл, то вы можете воспользоваться таблицей файлов, которая может быть отображена командой редактора

```
:files
```

После этого вы сможете ввести номер файла в таблице файлов и ввести комбинацию `Ctrl/^` – редактор немедленно перейдёт к указанному файлу.

Пример использования механизма вызова программ по команде редактора :make

По умолчанию, **Vim** при выполнении команды `:make` выполнит командную строку, которая содержится в параметре `makeprg`. Вы можете использовать в качестве такой строки две команды соединённые программным каналом, как показано ниже:

```
:set makeprg=gmake\ \\\|\ myfilter
```

Таким образом можно использовать программу **latex** вместо программы **make**. При этом резервирование места для параметров (комбинация `$*`) производится так:

```
:set makeprg=latex\ \\\\nonstopmode\ \\\\input\\{$*}
```

После выполнения команды `:make` (в даном случае будет выполнен проход **latex**) вы сможете просматривать диагностические сообщения посредством команд редактора.

Пример более сложной команды

Пусть нам требуется время от времени заменять одно слово в тексте на какое-то другое в соответствии с определённым алгоритмом. Реализовать такое можно посредством следующих шагов.

1. Отметить слово с помощью визуального режима.

2. Выполнить команду редактора K с предварительно установленным внутренним параметром редактора `keywordprg` .

3. Наконец, заменить отмеченное слово в тексте на результат, полученный с помощью команды редактора K.

Чтобы выполнить перечисленные шаги, выполним прежде следующие команды установки:

```
:set keywordprg=ESymb
:map F K<CR>:source /tmp/subst11<CR>
```

В первой установки внутреннему параметру `keywordprg` присваивается имя скрипта, который будет вызываться по команде редактора K. В данном случае скрипт производит требуемую перекодировку, а результат в виде команды редактора выводит в файл с именем `/tmp/subst11`. Вторая строка обеспечивает отображение имени новой команды `F` в командтую последовательность редактора, в которой сначала выполняется команда K, а затем выполняются команды редактора сформированые в файле `/tmp/subst11` командой K.

В результате, при вводе в нормальном (обычном) режиме команды `F` будет выполнена команда K, а затем команды редактора сформированые в файле `/tmp/subst11` командой K. Ниже приведены тексты соотвествующих скриптов.

```
#!/bin/bash
#-------------------------------------------+
# Usage:   ESymb word |
# Creation date:  Thu Feb 3 09:00:03 MSK 2000 |
#-------------------------------------------+

WORD="$1"

echo s/"$WORD"/‘symb $WORD‘@"$WORD"/ > /tmp/subst11

exit
```

```
#!/bin/sh
#  Перекодировать слово, представленное в коде koi-8,
#  в число, которое
#  есть конкатенация номеров позиций первых четыр\"ех
#  букв слова в алфавите.
#

ALPH='АаБбВвГгДдЕеЖжЗзИиЙйКкЛлМмНнОоПп'
ALPH=$ALPH'РрСсТтУуФфХхЦцЧчШшЩщЪъЫыЬьЭэЮюЯя'

echo "$1" | \
   awk --assign Line=$ALPH --file ~/bin/ProgN.awk

exit
```

```
# Awk программа ProgN.awk
{
Len=length($1)

if (Len < 4) Len = 4
if (Len > 4) Len = 4

i=1
while (i <= Len)
{
printf("%02d",index(Line,substr($1,i,1)))
i=i+1
}
printf("\n")
}
```

Перенаправление сообщений редактора

Рассмотрим теперь пример перенаправления сообщений из **Vim** в файл или в регистр.

`redi > Messages`

перенаправить все сообщения редактора в файл с именем `Messages`. Чтобы дописать сообщения в конец файла можно использовать команду

`redi >> Messages`

Чтобы перенаправить все сообщения редактора в регистр, следует использовать команду

`redi @R`

где R может быть любым регистром.

Чтобы завершить вывод сообщений в файл следует использовать команду

`redi END`

Примеры использования регулярных выражений при построении теговской таблицы программой etags

Приведённые примеры опираются на свойства программы **etags**.

`etags --language=none --regex='/параметр\|значение/nn/' *.tex`

В приведённой команде запрещается использовать синтаксис любого известного программе языка программирования, должны строится теги для строк файлов `*.tex`, где находятся подстроки `параметр` или `значение`. После завершения построения теговского файла, имя (`nn`) использованного регулярного выражения `/параметр\|значение/` можно использовать при вызове редактора, например,

`vim -t nn`

Если ничего не указано специально, то регулярное выражение будет проверяться на соотвествие только в начале каждой строки. Если вы желаете получить теговский файл с заданными тегами, которые могут располагаться произвольно по тексту, то для параметра `--regex` следует использовать конструкцию

`--regex='/.*параметр\|.*значение/nn/'`

Если в разных файлах вам надо найти разные подстроки, то можно использовать следующий вариант:

`etags --regex='/.*пара/' a1.tex --regex='/.*знач/' a2.tex`

В данном случае в таблицу тегов попадут строки, в которых содержатся указанные подстроки (`пара` в файле с именем `a1.tex` и подстрока `знач` в файле `a2.tex`). Эти теги будут сформированы в дополнение к тегам, которые программа **etags** формирует по умолчанию для файлов типа `.tex`.

Параметр `-R` отменит все значения параемтра `--regex=`.

`-R, --no-regexp` Отменить все значения (или действие) параметра `--regex=`.

Что делать, если что-то происходит не так

Во время редактирования с использованием редактора **Vim** может произойти что-то непредвиденное, что, впрочем, случается вообще в любой работе, а не только с конкретным редактором. Например, вы по ошибке удалили не тот кусок текста или вставили не тот кусок текста и не туда. Такое могло бы произойти если ошибочно двинуть мышку не в том направлении. В результате у вас появился неверный в целом текущий буфер.

Что можно предпринять, чтобы ликвидировать или уменьшить последствия таких ошибочных действий?

Во первых, следует перевести редактор в обычный режим введя комбинации

`Ctrl/c` или `Ctrl/[` или `<ESC>`

После перехода **Vim** в обычный режим следует один раз ввести команду

`u`

По этой команде происходит возврат к состоянию текста перед последним изменением. Внимательно изучив результат, можно повторить команду

`u`

По умолчанию, вы можете повторять команду `u` 100 раз, т.е. вернуться на 100 шагов назад. Обычно этого вполне достаточно для ликвидации последствий ошибочных действий.