ЭФФЕКТИВНАЯ РАБОТА В СИСТЕМЕ LINUX

Работа с текстами

1998 год

e-mail: Andrei.Chevel@pnpi.spb.ru
Дата формирования текста
16 марта 2000 г.

ЭФФЕКТИВНАЯ РАБОТА В СИСТЕМЕ LINUX
Работа с текстами

Андрей Е. Шевель

Начато 17 декбря 1999

Оглавление

Список иллюстраций

Список таблиц

Список примеров

Замечания по использованию шрифтов в данной книге

В данной книге мы будем стараться придерживаться следующих соглашений об использовании различных шрифтов.

- Имена серверов, ftp и http адреса будут печататься наклонным шрифтом, например, *http://www.pnpi.spb.ru*. Используется псевдофонт `\HSname{}` (Host name).

- Имена переменных окружения будут печататься другим шрифтом, например, $EDITOR. Таким же шрифтом будут обозначаться названия протоколов, например, протокол FTP. Используется псевдофонт `\VSname{}` (Variable name).

- Имена переменных, на месте которых следует использовать подходящие значения мы будем обозначать наклонным стилизованным шрифтом *variable*. Используется псевдофонт `\PSname{}` (Parameter name).

- Имена файлов, конкретных программ, а также сами исходные тексты программ и скриптов будут печататься машинописным шрифтом, например, `cvs` или `modules`. Используется псевдофонт `\FSname{}` (Constant or character name).

- Названия программных систем, имена команд и параметров будем печатать жирным шрифтом, например, система **CVS** или команда **commit**. Используется псевдофонт `\SSname{}` (System name).

- Ещё один фонт – это термины. Используется псевдофонт `\TSname{}` (Term name).

- Наконец, компьютерные сообщения или команды. Используется псевдофонт (как телетайп) `\CSname{}` (Computer line).

Фактически, из-за бедности выбора фонтов на моей установке, изображение части фонтов могут совпадать.

Глава 1

Редактор текста Vim

Vim означает **Vi Improved**, т.е. улучшенный редактор **vi**. **Vi** совместим с **Vim** снизу вверх.

Обсуждаемый редактор отличается весьма 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) (страница 72, полезно посмотреть термин `тег` в предметном указателе), 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 (страница 40). Имеется возможность использовать теговские таблицы построенные для редактора **emacs**.

Vim имеет интерфейсы для обращения к различным внешним программам и системам в том числе **SNiFF+**, **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. Смотрите раздел 1.13.

Vim позволяет автоматически настраиваться на определённые типы файлов, например, на редактирование исходных текстов различных языков программирования или распаковать файл перед чтением и т.д.

При этом редактор имеет широкий набор параметров (около 100), которые позволяют значительно изменять возможности по преобразованию текстов (смотрите таблицу 1.19 на странице 76), 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.

Поскольку обсуждаемый редактор является одним из наиболее мощных средств редактирования наравне с **emacs**, то в него встроено полное описание различных команд и возможностей. Страницы описания могут быть получены путём ввода команды редактора **:help**. Полезно взглянуть таблицу 1.22 на странице 112, которая содержит список файлов со страницами документов доступных в редакторе. **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/*.

1.1 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` (смотрите страницу 101).

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

`:files` (смотрите страницу 102).

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` (смотрите страницу 104).

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

`:marks` (смотрите страницу 39).

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

`:tags` (смотрите страницу 107).

В редакторе имеются специальные области внутренней памяти, которые называются регистрами. Вы можете использовать регистры как механизм временного хранения частей текста или команд редактора, Подробнее концепция регистров обсуждается в разделе 1.12.

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 1.1 на странице 5.

Существует ещё много десятков команд для выполнения простого редактирования и перемещения курсора по тексту. Однако для более сложных преобразований текста требуются и более сложные средства, часть из которых перечислены выше.

1.2 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 пропускаяопределённое 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`.

1.3 Некоторые команды редактирования с использованием элементов текста

1.3.1 Удаление различных объектов текста

Ниже перечислены основные операции удаления текста, которые производятся когда **Vim** находится в обычном (нормальном) режиме.

Таблица 1.1: Удаление

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

1.4 Некоторые режимы работы редактора

Редактор **Vim** функционирует в нескольких основных режимах: (полезно посмотреть `: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`. В таблице 1.2 приведены возможные действия.

- 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.

Таблица 1.2: Таблица перехода в визуальном режиме

Исходное состояние	Новый режим после ввода символа:		
	v	**V**	**Ctrl/v**
Обычный	визуальный посимвольный	визуальный построчный	визуальный поблочный
Визуальный посимвольный	обычный	визуальный построчный	визуальный поблочный
визуальный поблочный	визуальный посимвольный	визуальный построчный	обычный
визуальный построчный	визуальный посимвольный	обычный	визуальный поблочный

Редактирование текста производится в том месте, где стоит курсор. Перемещение курсора может производится различными клавишами и командами как описывается в разделе **??**.

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.

1.4.1 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`

Ниже приведена таблица переходов из одного режима в другой.

Таблица 1.3: Переходы из режима в режим

	В какой режим переходит						
Из какого режима переход	Normal	Visual	Select	Insert	Replace	Cmd-line	Ex
Normal		v V ^V	*4	*1	R	: / ? !	Q
Visual	*2		Ctrl/G	c C	нет	:	нет
Select	*5	^O ^G		*6	нет	:	нет
Insert	<ESC>	нет	нет		<insert>	нет	нет
Replace	<ESC>	нет	нет	<insert>		нет	нет
Cmd-line	*3	нет	нет	:start	нет		нет
Ex	:vi	нет	нет	нет	нет	нет	

Краткие пояснения к таблице 1.3.

1. Переход в режим ВВОДА (Insert) из ОБЫЧНОГО (Normal) режима после ввода команд `i`, `I`, `a`, `A`, `o`, `O`, `c C`, `s`, `S`.

2. Переход из ВИЗУАЛЬНОГО режима (Visual) в ОБЫЧНЫЙ (Normal) режим после ввода команды не связанной с перемещением курсора, а также после ввода `<ESC>`, `v`, `V`,`Ctrl/c` или `Ctrl/V`.

3. Переход из СТРОЧНО-КОМАНДНО режима (Cmd-line) в ОБЫЧНЫЙ режим (Normal) после ввода `<CR>` или `<NL>`, а также `Ctrl/U` или `<ESC>`.

4. Переход из ОБЫЧНОГО режима (Normal) в режим ВЫБОРА (Select).

5. Переход из режима ВЫБОРА (Select) в ОБЫЧНЫЙ режим (Normal) используя команды перемещения курсора.

6. Переход из режима ВЫБОРА (Select) в режим ВВОДА (Insert) посредством ввода обычного изображаемого символа.

1.5 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 1.4: 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 -`
-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** (раздел 1.14) или **etags** (раздел 1.15).
Продолжение на следующей странице	

Параметры при вызове редактора **Vim** (продолжение)	
Параметр	Значение
-q [*errorfile*]	Включить режим быстрой коррекции. Прочитывается файл с именем *errorfile* и отображается первая ошибка. Если файл *errorfile* опущен, то будет использовано имя по умолчанию `errors.vim`. Последующие ошибки могут быть получены посредством команды редактора `:cn`. Детали можно взглянуть посредством `:help quick-fix`.
+[*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.
-C	установить режим совместимости (COMATIBLE) с исходным редактором **vi**.
-e	запустить **vim** в режиме **ex**.
-f	главным образом для версий **GUI**. Редактор не будет порождать новых оболочек для выполнения команд. Этот параметр должен использоваться, если редактор вызывается другой программой (скриптом), которая ожидает завершения сессии редактирования.
Продолжение на следующей странице	

Параметры при вызове редактора **Vim** (продолжение)	
Параметр	Значение
-F	запустить редактор с поддержкой редактирования текстов на ФАРСИ, т.е. справа налево и с раскладкой клавиатуры для ФАРСИ.
-g	если редактор скомпилирован с поддержкой **GUI**, то он будет запущен с поддержкой **GUI**. Если редактор не был скомпилирован с поддержкой **GUI**, то он выдаст диагностику и завершится.
-h	выдать краткие ПОЯСНЕНИЯ к списку параметров, используемых в командной строке, и завершится.
-H	запустить редактор в режиме редактирования справа налево для раскладки клавиатуры HEBREW.
-L	то же самое что параметр **-r**.
-l	устанавливает режим LISP (смотрите также описание параметра LISP на странице 86).
-m	выключается модификация файлов. Иными словами, запись файлов невозможна.
-N	режим несовместимости с **vi**. Редактор **Vim** начинает вести себя лучше, но несколько иначе, чем старый редактор **vi**.
-n	Не используется своп-файл. Может оказаться удобным для работы на очень медленных устройствах вторичной памяти, например, дискетах. Восстановление после сбоя невозможно. Такой режим может быть установлен посредством команды `:set uc=0`. Отменить такой режим можно, например, командой `:set uc=200`
-o[N]	открыть N окон. Если N опущено, то открыть одно окно для каждого файла.
-R	установить режим ТОЛЬКО ЧТЕНИЕ.
Продолжение на следующей странице	

Параметры при вызове редактора **Vim** (продолжение)	
Параметр	Значение
-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`
-T *terminal*	Этот параметр предназначен для сообщения редактору типа терминала, который вы используете. Параметр нужен только тогда, когда тип терминала не установился автоматически. Можно использовать лишь те типы терминалов, которые имеются в файлах `termcap` или/и `terminfo`.
-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`

-U *gvimrc*	использовать инициализационный файл с именем *gvimrc* для инициализации режима **GUI**. При этом все другие виды инициализации режима **GUI** опускаются. Этот же параметр можно использовать, чтобы обойти любые виды инициализации режима **GUI**:
	`-u NONE`
	Детали можно посмотреть командой редактора **Vim**
	`:help gui-init`

Продолжение на следующей странице	

Параметры при вызове редактора **Vim** (продолжение)	
Параметр	Значение
-V	Разговорчивый режим (VERBOSE). Сообщаются имена файлов, которые были использованы для инициализации и другая информация.
-v	Запустить **Vim** в режиме **vi**.
-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`
-W *scriptout*	То же самое, что и параметр **-w**, но при повторном использовании того же имени файла он замещается.
-Z	режим работы с ограничениями (также как кличка редактора, которая начинается с буквы **r**).
`--`	(два минуса подряд) означает, что список параметров завершен; далее могут следовать лишь имена файлов.
`--`**version**	печатается версия вашего редактора **Vim** и много другой полезной информации.

1.6 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.

Одним из способов использования **Vim** в таком сценарии есть запуск редактора, затем запуск компилятора или другой программы, анализирующей текст, а потом повторный запуск редактора как показано ниже.

```
vim source.txt    # редактирование текста
prg source.txt >& FileName    # запуск компилятора
                              # или другой программы
                              # диагностика записывается
                              # в файл FileName
vim -q FileName   # просмотр ошибок
```

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. Ниже приведено краткое описание элементов формата.

Таблица 1.5: Описание формата

Описание формата	
Параметр	Значение
`%f`	имя файла (искать строку).
`%l`	номер строки (искать число).
`%c`	номер колонки (искать число).
`%t`	тип ошибки (один символ).
`%n`	номер ошибки (искать и ввести число).
`%m`	сообщение об ошибке (найти строку).
`%r`	удовлетворяет остатку (**r**est) однострокового файла сообщения `%O/P/Q`.

Продолжение на следующей странице

Описание формата (продолжение)	
Параметр	Значение
`%*conv`	любое не присваиваемое программой **scanf** преобразование.
`%%`	одиночный символ `%` (процент).

Следующие преобразования, обозначенные прописными буквами, определяют типы специальных форматных строк. Максимум одна из них может быть дана как префикс в начале одиночного отделённого запятой форматного образца (описателя). Некоторые компиляторы генерируют диагностические сообщения, которые состоят из имён каталогов, которые должны предшествовать каждому имени файла, который был прочитан посредством формата `%f` (пример: GNU make). Нижеследующие коды могут быть использованы чтобы сканировать эти имена каталогов; они будут запомнены во внутреннем стеке имён каталогов редактора.

- `%D` – форматная строка ВОЙТИ В КАТАЛОГ (enter directory); ожидается, что следующий формат `%f` найдёт имя каталога, в который происходит вход.
- `%X` – форматная строка ВЫЙТИ ИЗ КАТАЛОГА (leave directory); ожидается, что следующий формат `%f` найдёт имя каталога, из которого происходит выход.

Для чтения и анализа многострочных диагностических сообщений имеются специальные средства для описания форматов, которые перечисляются ниже.

- `%A` – Начало многострочного сообщения (неопределённый тип).
- `%E` – начало многострочного сообщения об ошибке.
- `%W` – начало многострочного предупредительного сообщения.
- `%C` – продолжение многострочного сообщения.
- `%Z` – конец многострочного сообщения.
- `%G` – глобальная установка; используется только совместно с знаками `+` (плюс) и `-` (минус).
- `%O` – однострочное сообщение содержащее имя файла: заменить совпадающую часть.

- `%P` – однострочное сообщение содержащее имя файла: поместить имя файла прочитанное с форматом `%f` в стек.

- `%Q` – однострочное сообщение содержащее имя файла: взять последнее имя файла из стека.

Коды `+` и `-` могут быть комбинированы с приведёнными выше кодами прописных букв. В этом случае они должны предшествовать прописным буквам, например, `%+A` или `%-G`. Смысл использования знаков следующий.

- `%-` – не включать подходящие по формату строки ни в какой вывод;

- `%+` – включить полностью подходящие по формату строки в `%m` строку.

Практически все регулярные выражения поддерживаемые в **Vim** могут быть использованы в описаниях формата. Ряд метасимволов, которые могут использоваться как часть имён файлов, должны записываться вместе с предшествующим знаком `%` (процент). Например,

- `%\` – означает одиночный сомвол `\`. Заметим, что в команде редактора `:set errorformat=` этот символ должен быть записан как `%\\`.

- `%.` – одиночная точка.

- `%#` – одиночная звёздочка $*$!!

- `%^` – одиночный символ `^`.

- `%$` – одиночный символ `$`.

- `%[` – одиночный символ `[`.

- `%~` – одиночный символ `~`.

Редактор позволяет посредством описания формата (см. пример на странице 15) определять виды сообщений программы и отделять одно сообщение от другого, переходить от одного сообщения к другому при просмотре, а также выполнять дригие операции. Редактор 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 1.6: 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.
`:cla[st] [`*nr*`]`	Отобразить диагностическое сообщение номер *nr*. Если *nr* опущено, то отображается ПОСЛЕДНЕЕ диагностическое сообщение.
Продолжение на следующей странице	

Описание команд просмотра диагностики (продолжение)	
Команда	Значение
`: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** (см. описание параметров командной строки 11). 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` (см. страницу 76) включён, то записать на диск все изменённые будеры. 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` (стр. 87). 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` (стр. 84). Используются также соответствующий внутренний параметр редактора для описания формата диагностических сообщений – `grepformat` (стр. 84).

Более детальную информацию можно почерпнуть на страницах
`:h quickfix`

1.7 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`

1.8 Инициализация редактора

В данном разделе обсуждается запуск редактора в алфавитно-цифровом режиме (не в **GUI**). Обсуждение режима **GUI** можно посмотреть в

`:h gui`

В момент запуска **Vim** проверяет переменные окружения и инициализационные файлы и устанавливает свои внутренние переменные в соответствии с найденными значениями. Инициализация производится в следующем порядке.

1. Устанавливается значение внутреннего параметра редактора *shell*. Если существует переменная окружения $SHELL, то она используется для установки параметра *shell*.

2. Устанавливается значение параметра *term*. Если существует переменная окружения $TERM, то её значение используется для установки *term*.

3. Выполняются строчные команды редактора, которые могут быть как в переменных окружения, так и в инициализационных файлах. Переменная интерпретируется как одна строка, которая содержит команды редактора. Команда редактора отделяется одна от другой вертикальной чертой `|` или символом `<NL>`.

   Файл, который содержит инициализационные команды, имеет имя `.vimrc`. Каждая строка файла `vimrc` интерпретируется как строковая команда **Ex**. Иногда этот файл называется `.exrc`. Однако, файл `.exrc` всегда используется редактором **vi**, а файл `.vimrc` предназначен только для редактора **Vim**. Обычным местом для нахождения персонального файла `.vimrc` является `$HOME/.vimrc`.

   Если **Vim** запускается с параметром **-u** `filename`, то в качестве инициализационного используется файл с именем `filename`. При этом пропускаются все пункты инициализации с номера 1 по номер 5 включительно. Если вам требуется просто пропустить первые пять пунктов инициализации, то можно использовать параметр

   -u *NONE*.

   Для инициализации используются четыре источника в нижеследующем порядке.

   - Переменная окружения $VIMINIT.
   - Пользовательский файл `$HOME/.vimrc`. Если такой файл не обнаружен, то делается попытка прочесть файл с именем `$HOME/_vimrc`. Если не найдены файлы `$HOME/_vimrc` и `$HOME/.vimrc`, то просматривается переменная $VIM.
   - Переменная окружения $EXINIT.
   - Пользовательский файл `$HOME/.exrc`.
   - Если параметр **exrc** установлен в значение *on* (что не является умолчанием), то будет произведена попытка найти в текущем каталоге в указанном порядке следующие файлы: `.vimrc`, `_vimrc`, `.exrc`, `_exrc`. Будет использован первый найденный файл, остальные будут игнорироваться.

4. Загрузить скрипт определения типа файла по умолчанию. Этот скрипт используется для автоматического выполнения команд при наступлении

события *FileType*. Это выполняется, если **Vim** скомпилирован с возможностью `+autocmd` (смотрите также страницу 109). Если **Vim** запущен с параметром **-u** *filename*, то данный этап инициализации будет опущен. Имя файла для этого скрипта $VIMRUNTIME/filetype.vim.

5. Установить значения параметров **shellpipe** и **shellredir**. Параметры **shellpipe** и **shellredir** устанавливаются автоматически в соответствии со значением параметра **shell**, если они не были ранее установлены вами вручную.

6. Установить **updatecount** в ноль, если использовался параметр **-n**.

7. Установить возможность редактирования двоичных файлов. Если при вызове редактора был использован параметр **-b**, то будет установлена возможность редактирования двоичного файла.

8. Выполнить инициализацию **GUI**, если стартует **gvim**.

9. Прочесть файл с именем `viminfo`. Если параметр `viminfo` не пуст, то будет прочитан файл с именем `viminfo`. По умолчанию значение параметра `viminfo` пусто.

10. Прочесть файл с именем `errorfile` (файл содержащий диагностические сообщения компилятора или другой программы) для возможности QUICKFIX. Это выполняется, если установлен параметр **-q**.

11. Открыть все окна. Если был установлен параметр **-o**, то будут открыты (но не отображены) все окна.

12. Выполнить команды старта редактора.

1.9 Алфавитный список основных команд редактора

Редактор имеет большой список команд (несколько сотен). Поскольку **Vim** фактически состоит из нескольких редакторов: **ex**, **ed** и собственно визуального редактора, то команды также несколько различаются. В начале мы рассмотрим список команд визуального (экранного) редактора. Эти команды выполняются в обычном режиме редактора (не в режиме ввода или других). Полный список команд редактора следует смотреть командой

```
:h index.txt
```

Общий формат команд экранного редактора очень прост. Например, `n-|` – установить курсор в позицию `n` в текущей строке. Здесь `n` – это счётчик числа повторений.

Таблица 1.7: Список команд редактора

Список команд редактора	
Команда	Значение
`Ctrl/a`	добавить число n к числу в тексте, на которое указывает курсор. Десятичным значениям может предшествовать знак. В зависимости от значения параметра **nrformats** *hex* или *octal* можно использовать шестнадцатеричные и восьмеричные числа без знака. Смотрите также `Ctrl/x`
`Ctrl/b`	пропустить (прокрутить) n экранов назад.
`Ctrl/c`	прервать текущую операцию поиска.
`Ctrl/d`	пропустить n строк вниз (по умолчанию половина экрана).
`Ctrl/e`	пропустить n строк вверх.
`Ctrl/f`	пропустить n экранов вперёд.
`Ctrl/g`	показать имя текущего файла и позицию курсора.
`<TAB>`	перейти к позиции курсора, записанной ранее в n–ной строке в таблице переходов. Содержание таблицы переходов можно посмотреть командой `:jumps`
`Ctrl/i`	то же что `<TAB>`
`Ctrl/l`	очистить экран и вывести информацию заново.
n`<CR>`	переместить курсор на n строк и поместить его в первую слева позицию.
`Ctrl/m`	то же что `<CR>`
`Ctrl/n`	то же самое что **j**.
n`Ctrl/o`	установить курсор в позицию, которую он имел n строк ранее в таблице переходов.
n`Ctrl/p`	переместить курсор на n строк вверх.
`Ctrl/q`	используется для управления выводом на экран (возобновить вывод).
`Ctrl/r`	снова внести изменения, которые были отменены командой **u** (отмена).
`Ctrl/s`	используется для управления выводом на экран (остановить вывод).
`Ctrl/u`	переместить курсор на n строк вверх (по умолчанию на половину экрана).
Продолжение на следующей странице	

Список команд редактора (продолжение)	
Команда	Значение
`Ctrl/v`	включить визуальный поблочный режим.
`Ctrl/w`	выбрать команды управления окнами. Смотрите таблицу 1.9.
`Ctrl/x`	вычесть из числа под курсором значение n. Смотрите также `Ctrl/a`.
`Ctrl/y`	переместить курсор на n строк вниз.
`Ctrl/z`	приостановить редактор (или запустить новую оболочку).
`Ctrl/^`	переключиться на редактирование файла с номером n в таблице файлов (предполагается, что вы редактируете сразу несколько файлов). Если n перед символом отсутствует, то производится переключение на редактирование предыдущего файла. Если параметр `autowrite` был включён, то при переключении будут сохранены изменения в файлах, если они имели место.
Продолжение на следующей странице	

Список команд редактора (продолжение)	
Команда	Значение

`!motion`*Fil*	отфильтровать строки начиная с текущей до той, которая указывается курсором. Здесь `motion` означает команду перемещения курсора, а *Fil* – любая команда **Linux**. После символа восклицательный знак должен следовать команда перемещения курсора.
	Команда выполняется следующим порядком. Указанные курсором строки передаются на ввод фильтра и удаляются из файла, а на их место помещается вывод команды *Fil*. Например, если вы введёте
	`! j`
	т.е. восклицательный знак, затем пробел, далее `j` (команда перемещения курсора на одну строку вниз) то внизу появится надпись `:.,.+1!` Если теперь ввести `date`, то вместо двух строк, на которые указывает курсор, вы получите одну с текущей датой, т.е. вывод программы **date**. А если ввести `date>/dev/null`, то будут просто удалены две строки.
	Если требуется отсортировать файл, то следует установить курсор в первую строку командой `:1`. Затем ввести `!G`, после чего внизу появится надпись
	`:.,$!`
	затем ввести команду **sort**.

`!!`*Fil*	Отфильтровать *n* строк. Команда выполняется следующим порядком. Указанное число строк начиная с текущей передаются на ввод фильтра и удаляются из файла, а на их место помещается вывод команды *Fil*. Например, вы вводите `5!!` При этом внизу появится надпись `:.,.+4!`. Введя команду **tac** (утилита в **Linux**) вы измените порядок строк в текущем буфере на обратный.

Продолжение на следующей странице

Список команд редактора (продолжение)

Команда	Значение
`"` (двойные кавычки)	вводят имя регистра для последующей команды удаления (**d**, **x**, **c**, **s** или **y**) или put. В качестве имени регистра используется один символ. Могут использоваться следующие символы `a-zA-Z0-9.%#:-` Регистры с именами `%#:.` могут использоваться только для `put`. Посмотреть содержание регистров можно командой редактора `:reg`. Подробнее смотрите раздел 1.12.
`#`	искать в обратном направлении n–ое включение слова, на которое указывает курсор.
`$`	переместить курсор в конец n–ой строки (отсчитывая от текущей).
`%`	искать в текущей строке любого вида скобки: круглые, квадратные или фигурные. Если курсор уже указывает на скобку, то при вводе `%` производится поиск соответствующей скобки. Например, если курсор указывает на `}`, то будет найдена ближайшая скобка `{` (если она существует) и т.д.
`n%`	установить курсор в позицию, которая соответствует `n%` от начала текста.
`&`	повторить предыдущую команду редактора `:s`.
`'{A-Za-z}` (одиночный апостроф и одна буква)	используется для обозначения перехода к точке текста, маркированной ранее. Например, `'a` – переход к точке, которая маркирована как `a`. Маркировка производится командой редактора `:mark`, например, `:mark a`.
`''` (два одиночных апострофа)	установить курсор в позицию, в которой он был до любой последней команды изменяющей положение курсора.
`'<`	переместить курсор на первый непробельный символ в строке, где начинается (или начиналась ранее) выделенная область в текущем буфере.

Продолжение на следующей странице

Список команд редактора (продолжение)	
Команда	Значение
`'>`	переместить курсор на первый непробельный символ в строке, где заканчивается (или заканчивалась ранее) выделенная область в текущем буфере.
`'[`	переместить курсор на первый непробельный символ в первой строке, где производились последние изменения или начало напечатанного текста.
`']`	переместить курсор в конец последней изменённой строки или в конец введённого текста.
`[n](`	переместить курсор на `n` фраз назад.
`[n])`	переместить курсор на `n` фраз вперёд.
`*`	искать в прямом направлении n-ное включение в тексте слова, на которое указывает курсор.
`+`	переместить курсор на n строк (умолчание 1) вперёд.
`-`	переместить курсор на n строк (умолчание 1) назад.
`.` (точка)	повторить последние изменения n раз (умолчание 1).
`/`*pattern*`<CR>`	искать вперёд n-ое включение *pattern*.
`/<CR>`	повторить предыдущий поиск вперёд.
`0`	установить курсор в первую слева позицию в текущей строке.
`[1-9]`	ввести число (счётчик) перед командой.
`:`	начать ввод команды **Ex**.
n`:`	начать ввод команды **Ex** с числового параметра.
`<motion`	сдвинуть n строк влево на **shiftwidth** позиций.
`<<`	сдвинуть n строк влево на **shiftwidth** позиций.
`>motion`	сдвинуть n строк вправо на **shiftwidth** позиций.
`>>`	сдвинуть n строк вправо на **shiftwidth** позиций.
`?`*pattern*`<CR>`	искать назад n-ое включение *pattern*.
`?<CR>`	повторить поиск в обратном направлении.
`@[a-z]`	выполнить n раз содержимое именованного буфера (регистра) `[a-z]`
`@:`	повторить предыдущую команду редактора n раз.
`@@`	повторить предыдущую команду `@[a-z]` n раз.
`A`	ввести текст после конца строки n раз.
`B`	переместить курсор на n слов назад.

Продолжение на следующей странице

Список команд редактора (продолжение)	
Команда	Значение
`["x]C`	заменить данные начиная с текущей позиции до конца строки и следующие n-1 строку. Одновременно записать удаляемый текст в регистр `x`.
`["x]D`	удалить данные начиная с текущей позиции курсора до конца строки и следующие n-1 строку. Одновременно записать удаляемый текст в регистр `x`.
`E`	переместить курсор в конец n-ого слова.
`Fc`	переместить курсор влево в позицию, в которой символ `c` встречается n-ый раз.
`G`	переместить курсор в строку с номером n (умолчание – последняя строка в файле).
`H`	переместить курсор n-ную строку от вершины экрана. В левый верхний угол по умолчанию).
`I`	вставить текст n раз до первого символа в строке.
`J`	Объединить n строк в одну строку (по умолчанию 2 строки).
`K`	выполнить команду, которая хранится в параметре с именем **keywordprg**. В качестве параметра следует использовать слово, которое находится под курсором. По умолчанию значение **keywordprg**=man.
`L`	переместить курсор на строку n от нижней части экрана. По умолчанию – курсор в левый нижний угол экрана.
`M`	переместить курсор в середину экрана.
`N`	повторить команду поиска `/` или `?` в противоположном направлении.
`O`	начать ввод в новой строке над текущей строкой.
`["x]P`	поместить текст из регистра `x` в строки перед позицией курсора n раз (по умолчанию 1).
`Q`	перейти в режим **Ex**.
`R`	ввести режим замены символов, т.е. при вводе будут замещаться имеющиеся символы.
`["x]S`	удалить n строк из буфера `x` и начать ввод Это синоним для команд `^cc` или `0cc`.
`Tc`	переместить курсор в позицию после n-ой встречи символа `c` при перемещении по строке влево.
Продолжение на следующей странице	

Список команд редактора (продолжение)	
Команда	Значение
`U`	вернуть строку в исходное состояние удалив все изменения, которые имели место (undo).
`V`	начать визуальный построчный режим.
`W`	переместить курсор на n слов вправо.
`["x]X`	удалить n символов до курсора (в регистре `x`).
`["x]Y`	скопировать n строк в буфер `x` (синоним для команды `yy`)
`ZZ`	запомнить текущий файл, если он был модифицирован и выйти из редактора.
`ZQ`	выйти из редактора без всяких действий и условий.
`]c`	группа команд закрывающей квадратной скобки (смотрите таблицу 1.10).
`[c`	группа команд открывающей квадратной скобки (смотрите таблицу 1.10).
`^` (шляпка)	переместить курсор в первую позицию текущей строки.
`_` (подчёркивание)	переместить курсор в первую позицию строки, которая находится ниже текущей на n-1 строк.
`'[a-zA-Z0-9]` (одиночный апостроф)	переместить курсор на метку в тексте с указанным однобуквенным именем. Например, `'k` – перейти к метке `k`, которая была введена ранее командой редактора `:mark k`.
`a`	начать ввод текста сразу после символа, на который указывает курсор.
`b`	переместить курсор на n слов назад.
`["x]cmotion`	удалить текст [и поместить его в регистр `x`] начиная с текущего положения курсора до позиции, куда курсор попадёт после команды перемещения курсора `motion` и начать ввод. Иными словами происходит замена текста в соответствии с основной командой `c`.
`["x]cc`	удалить n строк [и поместить их в регистр `x`] и начать ввод текста. Если параметр `autoindent` включён (on), то не сдвигать первую строку.
Продолжение на следующей странице	

Список команд редактора (продолжение)	
Команда	Значение
`["x]dmotion`	удалить текст начиная с текущей позиции курсора до позиции, в которую курсор установлен командой перемещения курсора `motion` [и поместить удаляемый текст в регистр `x`].
`["x]dd`	удалить n строк начиная с текущей [и поместить их в регистр `x`].
`e`	переместить курсор вперёд до конца слова номер n не обращая внимания на границы строк.
`fc`	переместить курсор вправо до n-ого включения символа `c` в текущей строке.
`gc`	расширенный список команд. Смотрите таблицу 1.8.
`h`	переместить курсор на n позиций влево.
`i`	вставить текст до курсора n раз.
`j`	переместить курсор на n строк вниз.
`k`	переместить на n строк вверх.
`l`	переместить курсор на n символов вправо.
`m{A-Za-z}`	установить метку `{A-Za-z}` на текущей строке.
`n`	повторить последнюю операцию поиска `/` или `?` n раз.
`o`	начать ввод новой строки под текущей строкой и введённый текст поместить n раз.
`["x]p`	поместить текст [из регистра `x`] сразу после курсора n раз.
`q{0-9a-zA-Z"}`	записать вводимые символы в именованный регистр `{0-9a-zA-Z"}`. Обозначение регистра буквами `{A-Z}`) используется если требуется дописать, а не заменить содержимое соответствующего регистра.
`rc`	заменить n символов в тексте на символ `c`.
`["x]s`	удалить n символов [и поместить их в регистр `x`] и начать ввод в соответствии с командой редактора `s` (substitute – заместить). Синоним для команды `cl` (не строковая).
`tc`	переместить курсор вправо до n-ого включения символа `c` в строке.
`u`	убрать все изменения (undo).
Продолжение на следующей странице	

Список команд редактора (продолжение)	
Команда	Значение
`z<CR>`	обновить экран (вывести то же самое заново). Строка, на которую указывает курсор, будет помещёна в верхнюю строку экрана, а курсор – в первую позицию верней строки.
`z.`	перестроить экран. Строка, на которую указывает курсор, помещается в центр экрана, а курсор – в первую позицию этой строки.
`z-`	перестроить экран. Строка, на которую указывает курсор, помещается внизу экрана, а курсор – в первую позицию этой строки.
`zb`	перестроить экран. Строка, на которую указывает курсор, помещается внизу экрана, а курсор – в первую позицию этой строки.
`zt`	перестроить экран. Строка, на которую указывает курсор, помещается верху экрана, а курсор – в первую позицию этой строки.
`zz`	перестроить экран. Строка, на которую указывает курсор, помещается в центр экрана, а курсор – в первую позицию этой строки.
`["x]C`	удалить начиная с позиции курсора до конца текущей строки, а также последующих n-1 строк [и поместить удалённую информацию в регистр `x`] и начать ввод. Синоним для команды `c$` (не строковая).

Таблица 1.8: Список G-команд редактора

Список G-команд редактора	
Команда	Значение
`g-Ctrl/g`	Показать информацию о текущей позиции кусора.
`g-Ctrl/h`	Начать блоковый режим ВЫБОРА.
`g-Ctrl/]`	Эквивалентно `:tjump` с именем тега, на который указывает курсор.
`g#`	Как `#`, только без использования `\<` и `\>`.
`g$`	Перевести кусрсор на наиболее правую позицию видимой на экране части текущей строки экрана.
Продолжение на следующей странице	

Список G-команд редактора (продолжение)	
Команда	Значение
`g*`	То же что `*`, но без использования `\<` и `\>`.
`g0`	Перейти к наиболее левой позиции текущей строки экрана.
`gD`	Перейти к глобальной декларации. Если курсор указывает на глобальную переменную, то перейти к определению этой переменной, если она определена в текущем файле. Поиск начинается с первой строки файла.
`gE`	Перейти назад к концу предыдущего слова.
`gH`	Начать строчный режим ВЫБОРА.
`gI`	Так как `I`, но всегда ввод начинается с колонки 1.
`gJ`	Объединить строки, не вставляя пробела между ними.
`["x]gP`	Поместить текст из регистра `x` после курсора.
`gR`	Ввести режим виртуальной замены на экране. Подробности следует смотреть на страницах `:h Virtual-replace-mode`.
`g]`	то же что `:tselect` с параметром, на который указывает курсор.
`ga`	Напечатать кодовое значение для символа, на который указывает курсор. То же что `:ascii`.
`ge`	То же что `gE`.
`gf`	Начать редактировать файл, чьё имя находится под курсором. При поиске файла используется значение паранетра редактора `path`, где содержится список каталогов, в которых будет произведён поиск заданного файла. Если имя файла под курсором представляет собой гиперлинк типа `type://host/catalog/filename`, то будет использоваться только часть `/catalog/filename`. То же что `[f` или `]f`.
`gg`	Перевести курсор на первую строку текещего файла.
`gm`	Перейти к символу в середине экрана в текущей строке.
`go`	Перейти к указанному символу текущего буфера (если введено число до команды). По умолчанию к началу файла.

Продолжение на следующей странице

Список G-команд редактора (продолжение)	
Команда	Значение
`gq`	Форматировать указанный текст.

1.10 Работа с окнами в редакторе vim

1.10.1 Введение

Окно в редакторе **Vim** есть место откуда вы можете осматривать содержимое внутреннего буфера редактора. В буфер загружен файл, который предназначен для редактирования. При старте **Vim** по умолчанию открывается одно окно. При этом неважно, на каком терминале в данный момент работает **Vim**: алфавитно-цифровой vt100 или X-терминал. На vt100 тоже может быть несколько окон разного размера.

Когда вы изменяете содержимое окна, то вы изменяете тем самым содержимое буфера редактора. Исходный файл на диске остаётся при этом неизменным. Чтобы изменить файл на диске, вы должны записать буфер редактора на диск. Буфер редактора может быть в трёх состояниях:

- АКТИВНЫЙ (active) – буфер отображается на экране и может содержать файл, который может модифицироваться командами редактора.
- НЕВИДИМЫЙ (hidden) – буфер не отображается на экране, но также может модифицироваться командами редактора.
- НЕАКТИВНЫЙ (inactive) – буфер не отображается на экране и ничего не содержит.

1.10.2 Запуск редактора с несколькими окнами

По умолчанию, **Vim** стартует с одним окном. Однако при разработке любого крупного текста: программы, состоящей из множества подпрограмм или крупного отчёта вы будете иметь дело с десятками модулей по несколько тысяч строк. Часто оказывается полезным редактировать сразу несколько модулей, например,

```
vim -o mod1.f mod2.f
```

По этой команде будут прочитаны во внутренние буферы два модуля и будут открыты одновременно два окна: половина экрана для одного файла и половина для другого. Если файлов много больше двух, то полезно использовать другую форму команды вызова редактора:

vim -o2 *.f

По этой команде будут вызваны все файлы типа *.f (т.е. файлы на фортране), но на экране будет организовано только два окна. Здесь имеется в виду, что вы может иметь несколько десятков или сотен модулей на фортране в данном каталоге. Следовательно каждое окно может оказаться размером в одну строку, что не всегда приемлемо.

Чтобы сделать приемлемый размер активного окна вы можете установить значение параметра **winheight** в какое-то разумное значение, например, 25.

:set winheight=25

Когда вы будете переключаться на другое окно, оно автоматически станет размером 25 строк. При этом все остальные окна получат размер, который останется (если это имеет место) на экране.

Переход из одного открытого окна в другое производится командами приведёнными в таблице 1.9.

Таблица 1.9: Команды преобразования окон

Команды преобразования окон **Vim**	
Команда	Значение
Ctrl/w+	увеличить высоту текущего окна на n.
Ctrl/w-	уменьшить высоту текущего окна на n.
Ctrl/w=	сделать все окна одинаковой высоты.
Ctrl/wR	переместить окна редактора циклично, т.е. нижнее окно сделать верхним и т.д.
Ctrl/wS	то же что Ctrl/ws, т.е. разделить текущее окно на два окна.
Ctrl/wW	перейти к очередному n-ому окну.
Ctrl/w]	разделить текущее окно на два и перейти к тегу, на который указывает курсор.
Ctrl/w^	разделить текущее окно на два окна и начать редактировать n–ный файл в списке параметров при вызове редактора **Vim**.
Ctrl/w_	установить высоту текущего окна в значение n.
Ctrl/wb	перейти к ниже лежащему окну.
Ctrl/wc	закрыть текущее окно (то же что команда редактора :close).
Ctrl/wd	разделить текущее окно на два окна и перейти к определению, которое находится под курсором.
Продолжение на следующей странице	

Команды преобразования окон **Vim** (продолжение)	
Команда	Значение
`Ctrl/wf`	разделить текущее окно на два окна и начать редактировать файл, на имя которого указывает курсор.
`Ctrl/wg Ctrl/]`	разделить текущее окно на два окна и выполнить команду перехода к тегу `:tjump` со значением параметра, на который указывает курсор.
`Ctrl/wg]`	разделить текущее окно на два окна и выполнить команду перехода к тегу `:tselect` со значением параметра, на который указывает курсор.
`Ctrl/wg}`	выполнить команду `:ptjump` со значением параметра, на который указывает курсор, разделить текущее окно на два окна и установить курсор на декларацию идентификатора, который был под курсором.
`Ctrl/wi`	разделить текущее окно на два окна и установить курсор на декларацию идентификатора, который был под курсором.
`Ctrl/wj`	перейти к n-ому следующему окну (остановиться на последнемі, если n больше числа окон).
`Ctrl/wk`	перейти к n-ому предыдущему окну (остановиться на первом, если n больше числа окон).
`Ctrl/wn`	открыть новое окно высотой n.
`Ctrl/wo`	закрыть все окна кроме текущего окна (то же что команда **Vim** `:only`).
`Ctrl/wp`	перейти в предыдущее окно (там откуда вы попали в текущее окно).
`Ctrl/wq`	закрыть текущее окно (подобно команде редактора `:quit`).
`Ctrl/wr`	прокрутить набор окон вперёд n раз.
`Ctrl/ws`	разделить текущее окно на два окна; новое окно высотой n строк.
`Ctrl/wt`	перейти к верхнему окну.
`Ctrl/ww`	перейти к следующему окну.
`Ctrl/wx`	сменить текущее окно на окно номер n (по умолчанию следующее окно).
`Ctrl/wz`	закрыть предыдущее окно, т.е. окно из которого вы попали в текущее окно.

Продолжение на следующей странице

Команды преобразования окон **Vim** (продолжение)	
Команда	Значение
`Ctrl/w}`	показать тег, на который указывает курсор в окне просмотра.
`Ctrl/w<Down>`	то же что `Ctrl/wj`, т.е. перейти к n-ому следующему окну (остановиться на последнемі, если n больше числа окон).
`Ctrl/w<Up>`	то же что `Ctrl/wk`, т.е. перейти к n-ому предыдущему окну (остановиться на первом, если n больше числа окон).

Здесь мы приведём только те команды, которые не связаны с синтаксисом конкретных языков программирования. Подробнее можно посмотреть описание, которое вызывается командой редактора `:help index.txt`.

Таблица 1.10: Команды квадратной скобки

Команды квадратной скобки	
Параметр	Значение
`[Ctrl/i`	перейти к началу файла и произвести поиск слова, на который указывает курсор.
`[(`	перейти n раз к непарной скобке `(`.
`[I`	перечислить все включения слова под курсором в текущем файле начиная с начала файла до конца.
`[i`	показать первое включение слова, на которое указывает курсор, при поиске от начала файла.
`[[`	переместить курсор назад на n разделов текста или к символу `{` в первой позиции.
`[]`	переместить курсор назад на один раздел или к предыдущей скобке `}` в первой позиции.
`[{`	переместить курсор назад n раз к незакрытой скобке `{`.
`]Ctrl/i`	переместить курсор к первому включению слова, на которое указывает курсор. Начать поиск с текущей позиции.
`]I`	перечислить все включения слова под курсором в текущем файле начиная с текущей позиции курсора до конца файла.
`][`	переместить курсор вперёд n раз на один раздел.
`]]`	переместить курсор вперёд n раз на один раздел.

Продолжение на следующей странице

Команды квадратной скобки (продолжение)	
Параметр	Значение
`]i`	переместить курсор к первому включению слова, на которое указывает курсор. Начать поиск с текущей позиции.
`]}`	переместить курсор вперёд к незакрытой скобке `}`.

1.11 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 нижнего регистра 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"
```

1.11.1 Установка меток

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

Таблица 1.11: Команды установки меток

Установка меток	
Команда	Значение
`m{a-zA-Z}`	установить одну метку с именем состоящем из одной буквы.
`m'` или `` m` ``	установить предыдущую контекстную метку. В эту точку можно будет вернуться командами `''` или ` `` `.
:[*n*]`ma[rk] {a-zA-Z}`	Установить значение метки с именем a-zA-Z в номер строки, который имеет *n*-ая строка от текущей.
:[*n*]`k{a-zA-Z}`	То же самое, что `mark`, только можно опустить пробел между командой и именем метки.
`'{a-z}`	Перейти на первый символ, не являющийся пробелом, в строке, помеченной меткой `{a-z}`
Продолжение на следующей странице	

Установка меток (продолжение)	
Команда	Значение
’{A-Z0-9}	Перейти на первый символ, не являющийся пробелом, в строке, помеченной меткой {A-Z0-9} в верном файле,
‘{a-z}	Перейти к метке с именем {a-z} (исключительно).
‘A-Z0-9}	Перейти к метке с именем {A-Z0-9}.
:marks	Показать список всех текущих меток.
:marks arg	Показать список всех меток, перечисленных в arg. Например, marks smh – показать значения меток с именами smh, если таковые метки существуют.

1.12 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) "_.

1.12.1 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 `"`.

1.12.2 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.

1.12.3 Регистр удаления

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

1.12.4 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.

1.12.5 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`.

1.12.6 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>`. Синтаксис ВЫРАЖЕНИЙ описан в разделе 1.13.

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.

1.12.7 Регистр выбора

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

1.12.8 Регистр черная дыра

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

1.12.9 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`,

1.13 Элементы внутреннего языка Vim

Чтобы получить дополнительную информацию об элементах внутреннего языка в редакторе, достаточно выполнить команду

```
:help expression
```

1.13.1 Внутренние переменные

Имена внутренних переменных могут состоять из букв, цифр и знака _ (подчёркивания). Имя переменной не может начинаться с цифры.

Внутренние переменные создаются командой `:let`. Можно ликвидировать внутреннюю переменную командой `:unlet`. Использование имени, которое не является именем внутренней переменной приведёт к ошибке.

Имена переменных, которым предшествует комбинация `:b` обозначают локальные переменные для текущего буфера. Имена переменных, которым предшествует комбинация `:w` обозначают локальные переменные для текущего окна. Имена переменных, которым предшествует комбинация `:g` обозначают глобальные переменные при обращении к ним внутри функций.

Имеется несколько предварительно определённых переменных.

1.13.2 Числа

В ВЫРАЖЕНИЯХ могут использоваться числа в следующих представлениях:

- десятичные числа;
- восьмеричные числа, которые начинаются с цифры 0;
- шестнадцатеричные числа, которые должны начинаться как `0x` или `0X`.

Например, 12 – десятичное число; 012 – восьмеричное число; 0x12 – шестнадцатеричное число.

1.13.3 Символьные строки

Для символьных строк используются двойные апострофы, например, `"строка"`. В строке могут использоваться следующие обозначения:

- `\...` – трёхзначное восьмеричное число, например, `\312`.
- `\..` – двузначное восьмеричное число, например, `\12`.
- `\.` – однозначное восьмеричное число, например, `\2`.
- `\x..` – двузначное шестнадцатеричное число, например, `\x31`.
- `\x.` – однозначное шестнадцатеричное число, например, `\x1`.
- `\X..` – то же что `\x..`.
- `\X.` – то же что `\x.`.
- `\b` – `<BS>` – шаг назад.
- `\e` – `<ESC>` – эскейп.
- `\f` – `<FF>` – смена формата.
- `\n` – `<NL>` – новая строка.
- `\r` – `<CR>` – возврат каретки.
- `\t` – `<TAB>` – табулятор.
- `\\` – обратный слеш.
- `\"` – двойной апостроф.
- `\<xxx>` – специальный ключ с именем `xxx`, например, `\<C-W>` для `Ctrl/w`.

Заметим также, что значение `\000` или `\x00` воспринимается как конец строки.

1.13.4 Литеральные строки

Литеральные строки, или литеральные константы, берутся как есть без интерпретации символов, включая обратный слеш. Они обозначаются как `'string'` (используются одиночные апострофы). Внутри строки не может использоваться одиночный апостроф.

1.13.5 Параметр редактора

Значение любого параметра редактора **Vim** в ВЫРАЖЕНИЯХ может быть обозначено как `&parameter_name`, например, `&winheight` (высота окна в строках).

1.13.6 Регистр

Значение любого РЕГИСТРА обозначается как `@r`. На месте такого обозначения появится содержание именованного регистра в виде единой строки. Символы `<NL>` будут добавлены там где необходимо. Для БЕЗЫМЯННОГО регистра следует использовать обозначение `@@`. Регистр выражений `"=` не может быть использован в ВЫРАЖЕНИЯХ.

1.13.7 Переменные окружения

Значение любой переменной окружения может обозначено как `$VAR`, где `VAR` есть имя переменной.

1.13.8 Функции

Обращение к функции производится посредством оператора

```
function(expr1, ...)
```

Редактор **Vim** имеет довольно широкий набор встроенных внутренних функций, которые перечислены в таблице 1.12. Чтобы увидеть в редакторе полный список функций и их описания, достаточно воспользоваться командой редактора

```
:help function
```

Таблица 1.12: ВНУТРЕННИЕ ФУНКЦИИ РЕДАКТОРА **Vim**

Внутренние функции редактора **Vim**		
Функция	Значение	Описание
append(lnum,string)	целое	добавить строку `string` после строки с номером *lnum*
argc()	число	количество файлов в аргументах при вызове редактора
argv(nr)	строка	*nr*–ный аргумент при вызове редактора

Продолжение на следующей странице

Внутренние функции редактора **Vim** (продолжение)		
Функция	Значение	Описание
browse(save,title,initdir,default)	строка	поместить запрос файла (только с **GUI**)
bufexists(expr)	число	ИСТИННО, если буфер с именем *expr* существует.
bufloaded(expr)	число	ИСТИННО, если буфер с именем *expr* загружен (не пуст).
bufname(expr)	строка	имя буфера с идентификатором *expr*, который может означать как число так и строку (подробнее смотрите описание `:h functions`)
bufnr(expr)	число	число буферов
bufwinnr(nr)	число	результат есть номер первого окна связанного с буфером *nr*.
byte2line(byte)	число	номер строки для байта номер *byte*.
char2nr(expr)	число	кодовое значение первого символа из строки *expr*.
col(expr)	число	текущая колонка курсора или колонка метки. Например, `col(".")` – означает текущую колонку курсора, а `col("'t")` есть колонка метки `t`.
confirm(msg,choices[,default[,type]])	число	данная функция предлагает пользователю диалог, в котором пользователь выбирает одно из нескольких значений. Подробнее смотрите `:h functions`.
delete(fname)	число	удалить файл с именем *fname*.
`did_filetype()`	число	ИСТИННО, если используется событие `FileType` в режиме `autocommand`.

Продолжение на следующей странице

Внутренние функции редактора **Vim** (продолжение)		
Функция	Значение	Описание
`escape(string,chars)`	строка	выделить символы в *chars*, которые встречаются в *string* обратной косой чертой. Например, выполнив команду редактора `:echo escape('File_Name','F_')` получим в результате `\File\_Name`.
`exists(Var)`	число	ИСТИННО (1), если переменная с именем `Var` существует. Аргумент может иметь несколько форм: • `&option-name` – имя параметра редактора **Vim**. • `$ENVNAME` – имя переменной окружения. • `*funcname` – имя встроенной функции редактора **Vim**. • `varname` – имя встроенной переменной редактора **Vim**.
`expand(expr)`	строка	раскрыть (развернуть) специальные метасимволы в выражении *expr*. Подробнее смотрите описание `:h expand`.
`filereadable(file)`	число	ИСТИННО (1), если файл с именем `file` разрешён для чтения.
`fnamemodify(fname,mods)`	строка	модифицировать имя файла *fname* в соответствии со значением модификаторов *mods*. Подробности в `:h fnamemodify`.
`getcwd()`	строка	выдать имя текущего рабочего каталога.
Продолжение на следующей странице		

Внутренние функции редактора **Vim** (продолжение)		
Функция	Значение	Описание
`getftime(fname)`	число	сообщить время последней модификации файла с именем `fname` в секундах с 1 января 1970 года.
`getline(lnum)`	строка	выдать строку номер *lnum* из текущего буфера.
`getwinposx()`	число	выдать координату X в пикселах для окна в режиме **GUI**.
`getwinposy()`	число	выдать координату Y в пикселах для окна в режиме **GUI**.
`glob(expr[,flag])`	строка	выполнить подстановку метасимволов в имени файла `expr`. Подробности в `:h glob`.
`has(feature)`	число	ИСТИННО (1), если возможность *feature* поддерживается текущей версией редактора **Vim**.
`histadd(history,item)`	строка	добавить запись к истории выполнения различных операций. Подробности в `:h histadd`.
`histdel(history)`	строка	удалить запись из файла истории выполнения различных операций. Подробности в `:h histdel`.
`histget(history[,index])`	строка	получить запись из файла истории выполнения различных операций. Подробности в `:h histdel`.
`hlexists(name)`	строка	ИСТИННО (1), если выделенное групповое имя *name* существует.
`hlID(name)`	число	синтаксис `ID` для выделенного группового имени *name*
`hostname()`	строка	имя хоста, на котором выполняется редактор **Vim**.
`input(prompt)`	строка	ввести строку, которую напечатает пользователь

Продолжение на следующей странице

Внутренние функции редактора **Vim** (продолжение)		
Функция	Значение	Описание
`isdirectory(directory)`	число	ИСТИННО (1), если имя *directory* есть имя каталога.
`line(expr)`	число	номер строки, которая задаётся параметром *expr*. Например, `line(".")` – номер строки, где находится курсор; `line("'t")` – номер строки, где находится метка `'t` и `line('$')` – номер последней строки в текущем буфере.
`match(expr,pat)`	число	индекс (смещение) подстроки *pat* в строке *expr*.
`matchend(expr,pat)`	число	то же самое, что `match(expr,pat)`, но возвращается индекс символа, который следует сразу за найденной подстрокой.
`nr2char(expr)`	строка	вернуть символ, который соответствует значению *expr*. Например, `nr2char(64)` вернёт `@`
`setline(lnum,line)`	число	установить строку номер *lnum* текущего буфера в значение *line*. Если всё нормально, то возвращается 0, в противном случае – 1.
`strftime(format)`	строка	возвращаемое значение есть строка показывающая текущее время в соответствии с заданным форматом. Например, результатом команды редактора `:echo strftime("%c")`, которую я выполнил было `Чтв 20 Янв  2000 17:21:33`.
`strlen(expr)`	число	длина строки *expr* в байтах.
Продолжение на следующей странице		

Внутренние функции редактора **Vim** (продолжение)		
Функция	Значение	Описание
`strpart(src,start,len)`	строка	возвращается часть строки *src* начиная с символа номер *start* длиной *len*.
`synID(line,col,trans)`	число	синтаксис *ID*, который имеет колонку *col* и строку *line* в текущем окне.
`substitute(expr,pat,sub,flags)`	строка	копия строки *expr*, в которой первое вхождение подстроки *pat* заменено на подстроку *sub*.
`tempname()`	строка	возвращается имя временного файла. Как минимум 26 последовательных обращений дадут разные имена временных файлов.
`virtcol(expr)`	число	выводит колонку символа на экране терминала для текущего файла, в котором указана позиция *expr*.
`winbufnr(nr)`	число	номер буфера связанного с окном номер *nr*.
`winheight(nr)`	число	высота окна номер *nr*.
`winnr()`	число	номер текущего окна.

1.13.9 Группирование

Любые операторы в ВЫРАЖЕНИЯХ могут быть сгруппированы посредством круглых скобок, например,
(ВЫРАЖЕНИЕ).

1.13.10 Команды, используемые в выражениях

В ВЫРАЖЕНИЯХ могут использоваться команды, которые описываются в таблице 1.13.

Таблица 1.13: Команды в выражениях

Команды в выражениях	
Команда	Описание
`:let` *var-name=expr*	присвоить внутренней переменной с именем *var-name* значение *expr*. Если внутренняя переменная с именем *var-name* не существует, то она будет создана. Переменная получит тип выражения *expr*.
`:let` *$env-name=expr*	присвоить переменной окружения с именем *$env-name* значение *expr*. Переменная получит тип СТРОКА.
`:let` *@reg-name=expr*	записать значение выражения *expr* в регистр с именем *reg-name*.
`:let` *&option=expr*	установить значение параметра *option* равным результату выражения *expr*.
`:unl[et][!]` *var-name...*	удалить внутреннюю переменную с именем *var-name*. Если использован восклицательный знак, то не будет выдаваться сообщение об ошибке при попытке удаления несуществующей переменной.
`:if` *expr1* `:en[dif]`	Выполнить последовательность команд редактора, которые находятся до соответствующей команды `else` или `endif`, если значение выражения *expr1* не равно нулю.
`:el[se]`	Выполнить последовательность команд редактора, которые находятся до соответствующей команды `:else` или `:endif`, если они до этого не выполнялись.
`:elsei[f]` *expr1*	Краткая запись для последовательности `:else` `:if` с дополнением, которое не содержит `:endif`.

Продолжение на следующей странице

Команды в выражениях (продолжение)	
Команда	Описание
`:wh[ile]` *expr1* `:endw[hile]`	Повторять выполнение последовательности команд редактора, находящихся между `:wh[ile]` и `:endw[hile]` до тех пор пока выражение *expr1* не равно нулю. Если возникнет ошибка при выполнении команды внутри цикла `:wh[ile]` и `:endw[hile]`, то после ошибки выполнение продолжится начиная с оператора после `:endw[hile]`.
`:con[tinue]`	Когда оператор используется внутри цикла `:wh[ile]` и `:endw[hile]`, то произойдёт переход к первому оператору цикла.
`:brea[k]`	Когда оператор используется внутри цикла `:wh[ile]` и `:endw[hile]`, то произойдёт переход к первому за командой `:endw[hile]`.
`:ec[ho]` *expr1 ...*	Вывести значение каждого из выражений *expr1 ...* на экран, завершив вывод символом КОНЕЦ СТРОКИ. Например. `:echo "the value of 'shell' is" &shell`
`:ec[hon]` *expr1 ...*	Вывести значение каждого из выражений *expr1 ...* на экран. При этом символ КОНЕЦ СТРОКИ не выводится.
`:echoh[l]` *name*	Установить выделенное групповое имя *name* для последующих команд `:echo`. Важно не забыть установить `echohl None`, иначе все последующие команды `echo` будут выделенными.
`:exe[cute]` *expr1 ...*	Выполнить строку, которая является результатом выражений *expr1 ...* как команду редактора **Ex**. Если имеется несколько строк объединяются в одну с одним пробелом между строками. Эта команда позволяет написать цикл `while` в одной строке (смотрите примеры на странице 122).

Простые примеры использования внутреннего языка **Vim** приведены на странице 122.

1.14 Программа ctags

Программа **ctags** является одной из программ генерации таблицы тегов (или индексной таблицы) для набора файлов содержащих исходные тексты программ.

Вызов программы:

`ctags [options] [file(s)]`

Программа **ctags** и родственная ей программа **etags** будут именоваться далее как программа **ctags**, если это не приведёт к путанице. Программа строит во время работы таблицу индексов или тегов анализируя исходные тексты программ на языках **C**, **C++**, **Eiffel**, **Fortran** и **Java**. Наличие такой таблицы позволяет легко и быстро переходить от одного объекта в программе к другому независимо от того, в каком файле располагается искомый объект. Такую таблицу удобно использовать в редакторе текста или какой-либо утилите. Такие теговские (индексные) файлы поддерживаются рядом редакторов: **Vim**, **emacs**, **nedit** и другими, а также такими программами просмотра текста, как **less**.

Программа **ctags** способна генерировать теговские файлы с использованием определённых языковых конструкций, которые перечисляются ниже для каждого языка отдельно.

1.14.1 C/C++

Программа принимает во внимание следующие языковые конструкции:

- имена макроопределений (из операторов `#define/#undef`);
- перечисления (enumerators);
- определения функций;
- прототипы;
- декларации классов, перечислений (enum), структур и объединений (union);
- пространства имён (namespaces);
- определения типов (typedefs);
- переменные (определения и декларации);
- члены классов, структур и объединений.

1.14.2 Fortran

- инициализация общего блока (block data);
- общая память (common block);
- точки входа;
- функции;
- интерфейсы;
- метки;
- модули;
- именованные списки ввода/вывода (namelist);
- программы;
- подпрограммы;
- производные типы.

1.14.3 Java

- классы;
- поля;
- интерфейсы;
- методы;
- пакеты.

1.14.4 Типы файлов с исходными текстами

Типы файлов с исходными текстами определяются автоматически на основании имеющегося расширения, например, `*.c` – для языка **C** и т.д. если не указан параметр `--lang`. По умолчанию, все другие файлы с другими расширениями игнорируются. Таким образом, имеется возможность использовать программу **ctags** следующим образом:

```
ctags *
```

если вас интересует лишь текущий каталог или

```
ctags -R *
```

в случае когда вам нужен теговский файл по всему нижележащему дереву каталогов. При этом в файле будут приняты во внимание все типы исходных текстов, которые известны программе **ctags**. Обратим также внимание, что в одном теговском файле может быть сгруппирована информация об элементах сотен программ, расположенных в десятках каталогов.

Замечательно, что построенный теговский файл с именем (по умолчанию) *tags* может оказаться полезен также с программой **less**, например, по команде

`less -t tag-name`

будет отображён файл, который содержит тег с именем `tag-name`. Подробности можно увидеть на страницах руководства:

`man less` или `info less`

1.14.5 Параметры программы ctags

Следует отметить, что существует несколько различных версий этой программы. Здесь рассматривается версия, которая отвечает на команду

`ctags --version`

строкой

`Exuberant Ctags 3.4, by Darren Hiebert <darren@hiebert.com>`

Параметры, которые воспринимает данная программа сгруппированы в таблице 1.14.

Таблица 1.14: Параметры программы **ctags**

Параметры программы **ctags**	
Параметр	Значение
`-a --append`	дописать имеющийся теговский файл.
`-B`	использовать команды поиска в обратном направлении для определения положения тега в файле, т.е. поиск вида `?expr?`. Игнорируется, если использован параметр `-e`.
`-e`	построить теговский файл для использования с программой **emacs** (он несколько отличается по формату от теговского файла для **Vim**).
Продолжение на следующей странице	

Параметры программы **ctags** (продолжение)	
Параметр	Значение
`-f` *filename*	использовать имя *filename* для теговского файла, по умолчанию используется имя `tags` или `TAGS` для параметра `-e`. Для выдачи информации на стандактное устройство вывода следует использовать знак - (минус) на месте имени файла.
`-F`	использовать команды поиска в прямом направлении для определения положения ега в файле, т.е. поиск вида `/expr/`. Игнорируется, если использован параметр `-e`.
`-h` *list*	определяет список окончаний имён файлов, разделённых точками, который интепретируется как окончания имён файлов заголовков или файлов типа `include`. Этот параметр воздействует на область видимости различных тегов. Теговский тип, который не находится в файле типа `include` рассматривается как видимый только в том файле, где он определён. Значение параметра по умолчанию = `.h.H.hh.hpp.hxx.h++.inc.def`. Следует обратить внимание на параметр `--file-scope`.
Продолжение на следующей странице	

Параметры программы **ctags** (продолжение)	
Параметр	Значение
`-I` *tokenlist*	Список кусочков, которые будут обрабатываться специальным образом при анализе текста на **C/C++**. Параметр предназначен для учёта особенностей, связанных с использованием препроцессорных макроопределений. Если перечисленные в списке кусочки являются простыми подстроками (без специальных знаков), то они прото игнорируются во время анализа иснодного текста. Если сразу за кусочком следует знак + (плюс) , то **ctags** будет также игнорировать любые выражения в скобках, которые могут следовать в исходном тексте за обозначенным кусочком. Если два кусочка разделены знаком = (равно), тогда первый кусочек заменяется вторым кусочком для правильного анализа текста. Список кусочков может быть представлен как в командной строке, так и находиться в файле. Если список кусочков начинается символом . (точка) или слешем, то это рассматривается как имя файла, в котором нранится список кусочков, разделённых пробелами. Например, для текущего каталога можно использовать имя `./tokens`. Можно использовать несколько параметров `-I`.
`-L` *file*	читать из файла с именем *file* список имён файлов, которые должны быть учтены при построении результирующего теговского файла. Если на месте имени файла стоит знак - (минус), то список имён файлов будет считываться со стандартного устройства ввода.
`-n`	то же самое, что `--excmd=`*number*.
`-N`	то же самое, что `--excmd=`*pattern* (регулярное выражение).
Продолжение на следующей странице	

Параметры программы **ctags** (продолжение)	
Параметр	Значение
`-p` *path*	использовать каталог с именем *path* как главный каталог для всех исходных текстов, т.е. остальные имена файлов будут относительны к главному каталогу, если не указано абсолютное имя файла.
`-R`	то же что `--recurse`.
`-u`	то же что `--sort=no`, т.е. не сортировать.
`-V`	то же что `--verbose`, т.е. информировать о происходящем ПОДРОБНО.
`-x`	вывести человеческий табличный вид перекрёстных ссылок на экран вместо построения теговского файла. Выводимая информация содержит: • имя тега; • тип тега; • номер строки, где нйден тег; • имя файла, в котором найден тег; • исходная строка текста, которая содержит тег.
`--append=`*yes*`|`*no*	дописывать (*yes*) или нет (*no*) уже существующий теговский файл.

Продолжение на следующей странице

Параметры программы **ctags** (продолжение)	
Параметр	Значение
`--c-types=`*types*	Указать типы тегов, которые следует обрабатывать для программ на языке **C/C++**. Каждый тип обозначается одной буквой. Если букве предшествует знак - (минус), то данный тип исключается из списка по умолчанию. Если типу предшествует знак + (плюс), то тип добавляется к списку по умолчанию. Отсутствие знаков - (минус) или + (плюс) интерпретируется как включение только тех типов, что указаны явно, т.е. не рассматировается умолчание. Обозначения типов приведено ниже.
Продолжение на следующей странице	

Параметры программы **ctags** (продолжение)	
Параметр	Значение
	• c - классы; • d - макроопределения; • e - перечисления; • f - определения функций; • g - имена перечислений; • m - классы, структуры, имена объединений; • n - пространство имён; • p - прототипы функций и декларации; • s - имена структур; • t - определения типов (typedefs); • u - имена объединений; • v - определения имён переменных; • x - внешние и (forward) переменные
Продолжение на следующей странице	

<table>
<tr><td colspan="2">Параметры программы ctags (продолжение)</td></tr>
<tr><td>Параметр</td><td>Значение</td></tr>
<tr><td></td><td>В дополнение к вышеприведённому, используются следующие модификаторы.
A – записать доступ каждого члена в теговский файл. Эта информация записывается с расширением <code>access</code>;
C – включить описатель класса для каждого члена в форме <code>class::member</code>. Это позволяет определить положение тега с использованием названия класса, например, <code>:tag class::member</code>. Такая возможность увеличивает казмер теговского файла примерно вдвое, поэтому выключена по умолчанию.</td></tr>
<tr><td><code>--eiffel-types=</code>types</td><td>Определяет список типов тегов для языка Eiffel с использованием тех же соглашений о формате, что приведены выше для языка C/C++ (смотрите описание параметра <code>--c-types</code>).. Следующие типы тегов могут быть рассмотрены в языке Eiffel.
c – классы;
f – особенности (features);
l – локальные сущности (local entities).</td></tr>
<tr><td colspan="2">Продолжение на следующей странице</td></tr>
</table>

Параметры программы **ctags** (продолжение)	
Параметр	Значение
	В дополнение может использоваться модификатор C. Если он используется, то в теговский файл включается дополнительная информация о принадлежности каждого члена к определённому классу, с тем, чтобы можно было позже воспользоваться командой редактора `:tag class.member`. По умолчанию эта возможность выключена, поскольку приводит к увеличению размера теговского файла.
`--etags-include=`*file*	включает ссылку на файл с именем *file* в теговский файл. Этот параметр может использоваться много раз. Это может использоваться только в редакторе **emacs**.
`--excmd=`*type*	определить тип команд редактора **Ex**, который является составной частью **Vim**, с помощью которых будет определяться положение тегов в файле. Допустимыми значениями являются: • `n[umber]` – для определения положения тегов использовать только числовые значения (номер строки, символа); • `p[attern]` – для определения положения тегов использовать только поиск регулярных выражений; • `m[ixed]` – используется в основном тип `p[attern]` с некоторыми исключениями. Детали следует смотреть в `info ctags`.
`--file-scope=`*yes*`|`*no*	определяет, как рассматриваются теги: только в пределах файла в котором определёны или являются видимыми извне. По умолчанию, используется значение *yes*. Смотрите также описание параметра `-h`.
Продолжение на следующей странице	

Параметры программы **ctags** (продолжение)	
Параметр	Значение
`--file-tags=yes|no`	Должны ли генерироваться теги для имён файлов с исходными текстами. По умолчанию установлено `no`.
`--format=`*level*	Определить формат выводного теговского файла. Сейчас принимаются значения *level* 1 и 2. Значение *level*=1 означает обычный формат. Значение *level*=2 означает расширенный формат, который содержит дополнительные флаги. По умолчанию установлено *level*=2. Параметр игнорируется, если использован `-e`.
`--fortran-types=`*types*	определить типы тегов для языка **фортран**, которые будут фигурировать в результирующем теговском файле. Смотрите описание параметра `--c-types` для определения формата строки *types*. Для языка **фортран** поддерживаются следующие типы тегов.

Продолжение на следующей странице

Параметры программы **ctags** (продолжение)	
Параметр	Значение
	• `b` – block data; • `c` – common blocks; • `e` – entry points; • `f` – определение функции; • `i` – интерфейсы; • `l` – метки; • `m` – модули; • `n` – списки ввода/вывода (namelists); • `p` – программы; • `s` – подпрограммы (subroutines); • `t` – производные типы.
`--help`	вывод справочной информации на устройство стандартного вывода.
`--java-types=`*types*	определить список типов тегов для языка **Java**, которые будут фигурировать в результирующем теговском файле. Смотрите описание параметра `--c-types` для определения формата строки *types*. Для языка **Java** поддерживаются следующие типы тегов.
Продолжение на следующей странице	

Параметры программы **ctags** (продолжение)	
Параметр	Значение
	• `c` – классы; • `f` – поля; • `i` – интерфейсы; • `m` – методы; • `p` – пакеты. В дополнение используются модификаторы. • `A` – записать доступ каждого поля в теговский файл. Эта информация записывается с использованием дополнительного флага с именем `access`. • `C` – включить дополнение, т.е. тег, который описывает каждый член класса в форме `class.member`. Это позволяет позже определить положение тега посредством команды редактора `:tag class.member`. По умолчанию эта возможность выключена, поскольку может привести к удвоению размеров генерируемого теговского файла.
`--kind-long=yes|no`	определить как будут описываться дополнительные теги в теговском файле (полными словами или одной буквой).
`--lang=c|c++|eiffel` или `--lang=fortran|java`	по умолчанию программа **ctags** автоматически определяет язык исходного текста на основании окончаний имён файлов, игнорируя файлы с нераспознанными окончаниями. Этот параметр заставляет рассматривать все файлы как тексты заданного языка.

Продолжение на следующей странице

Параметры программы **ctags** (продолжение)	
Параметр	Значение
`--langmap=map`	определить связь между окончанием имени файла и языком. Например, строка `--langmap=c:.c.ec.xs,java:+.j` определяет, что файлы, имена которых заканчиваются на `.c`, `.ec`, `.xc` рассматриваются как программы на языке **C**. В то же время, файлы, имена которых заканчиваются на `.j` и другие ранее определённые окончания являются программами на языке **Java**.
`--line-directives=yes` или `--line-directives=no`	определить будет ли распознаваться директива `#line`.
`--links=yes|no`	определить будут ли использоваться символические линки. По умолчанию символические линки игнорируются.
`--recurse=yes|no`	просматривать рекурсивно указанные каталоги или текущий каталог, если ничего не указано, а также следовать всем символическим линкам. Следует понимать, что вы собираетесь делать используя данный параметр. Может оказаться, что вы имеете массу каталогов, имена которых начинаются с точки.

1.15 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 1.15: Параметры программы **etags**

Параметры программы **etags**	
Параметр	Значение
`-a, --append`	Дописать существующий теговский файл.
`-C, --c++`	Рассамтривать имена файлов с окончаниями `.c`, `.h` как тексты на языке **C++**, а не **C**. Имена файлов с окончаниями `.C`, `.H`, `.cxx`, `.hxx`, `.cc` всегда рассматриваются как тексты на языке **C++**.
`-D, --no-defines`	Только для **etags**. Не создавать теги для препроцессорных определений **C**.
`-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 (смотрите пример на странице 123).
`-H, --help`	Выдать на экран краткую информацию о параметрах программы.
`-V, --version`	Напечатаь версию используемой программы.

1.16 Использование таблицы тегов

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

Таблица 1.16: Команды для перемещения по тегам

Команды для перемещения по тегам	
Команда	Значение

Продолжение на следующей странице

Команды для перемещения по тегам (продолжение)	
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.
`: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.

1.17 Получение информации о положении курсора

Таблица 1.17: Информация о положении курсора

Информация о положении курсора	
Команда	Значение
`Ctrl/g` или `:f[ile]`	Напечатать текущее имя файла и позицию курсора.
count `Ctrl/g`	То же, что `Ctrl/g`, но дополнительно печатается абсолютное имя файла. Если значение *count* больше 1, то печатается и имя буфера.
`g Ctrl/g`	Выводит текущую позицию курсора в следующем виде: Колонка, Строка, Символ. Если часть символов занимает более одной колонки, например, символ `<TAB>`, то они указываются через чёрточку.

1.18 Внутренние параметры Vim

Как уже упоминалось, **Vim** имеет набор внутренних параметров, которые могут быть установлены или изменены командой `:set`. Эти параметры оказывают влияние на особенности поведения редактора в различных ситуациях. Многие параметры имеют сокращённые имена, которые могут употребляться наравне с полными именами внутренних параметров.

Часть внутренних параметров являются булевскими, т.е. имеют два значения: установлено и не установлено. Так, установка параметра `option` выполняется командой

`:set option`

А состояние не установлено устаналивается командой

`:set nooption`

Более детально особенности команды `set` приведены в таблице 1.18 на странице pagerefsettable.

Таблица 1.18: Команда **set**

Команда **set**	
Форма команды	Значение
`se[t]`	показать или установить параметры.
`:se[t] all`	показать все параметры, кроме параметров терминала.
Продолжение на следующей странице	

Команда **set** (продолжение)	
Форма команды	Описание
`:se[t] termcap`	показать все параметры терминала.
`:se[t]` *option*?	показать значение параметра с именем *option*.
`:se[t]` *option*	установить булевкий параметр с именем *option* в значение ВКЛЮЧЕНО, или показать значение, если тип параметра есть число или строка.
`:se[t] no`*option*	установить булевкий параметр с имеем *option* в значение ВЫКЛЮЧЕНО.
`:se[t]` *option*! или `se[t] inv`*option*	инвертировать значение параметра с именем *option*, например, из ВЫКЛЮЧЕНО перевести в значение ВКЛЮЧЕНО.
`:se[t]` *option=value*	установить параметр с именем *option* в значение *value*.
`:se[t]` *option+=value*	Увеличить числовой параметр *option* на величину *value* или дописать строку *value* к имеющейся строке *option*, если *option* есть строка. Если параметр *option* есть разделённый запятыми список, то перед *value* добавляется запятая. Так, если параметр *option* содержал значение `abc`, то после выполнения команды `:se option+=de` параметр *option* будет содержать новое значение `abc,de`.
`:se[t]` *option^=value*	Умножить числовой параметр *option* на величину *value* или, если *option* есть строка, поместить строку *value* перед тем значением, что содержалось в *option*. Так, если параметр *option* содержал значение `abc`, то после выполнения команды `:se option^=de` параметр *option* будет содержать новое значение `de,abc`.
Продолжение на следующей странице	

Команда **set** (продолжение)	
Форма команды	Описание
`:se[t]` *option-=value*	Вычесть величину *value* из числового параметра *option* или, если *option* есть строка, убрать из строки *option* отделённую запятой подстроку *value*. Так, если параметр *option* содержал значение `de,abc`, то после выполнения команды `:se option-=de` параметр *option* будет содержать новое значение `abc`.
`:se[t] option&`	Вернуть значение параметра `option` в значение по умолчанию.
`:se[t] all&`	Вернуть все параметры в значение по умолчанию кроме параметра `term` и `line`, которые остаются без изменений.

Большая часть внутренних параметров действуют всегда для всех окон и буферов. Однако небольшая часть может быть локальной для конкретного файла или окна, например, размер окна. Другая часть внутренних параметров действуют, если они были включены во время трансляции **Vim**. Чтобы установить, какие внутренние параметры были включены во время трансляции, следует использовать команду `:version`.

Большая часть внутренних параметров приведена в таблице 1.19 на странице 76. Все параметры описаны на страницах, которые доступны в редакторе посредством команды

```
:help option-list
```

Таблица 1.19: Внутренние параметры редактора **Vim**

Внутренние параметры редактора **Vim**	
Параметр	Значение

Продолжение на следующей странице

<table>
<tr><th colspan="2">Внутренние параметры редактора Vim (продолжение)</th></tr>
<tr><td>Параметр</td><td>Значение</td></tr>
<tr><td><code>autoindent (ai)</code></td><td>Тип: булевский; значение по умолчанию: 0; локально для данного буфера. Этот параметр, будучи установленным, приводит к следующему. Копируется смещение текста от начала в текущей строке и используется при вводе следующей строки, когда редактор находится во режиме ввода (Insert). Форматирование абзацев (<code>gq</code>) использует смещение первой строки, если параметр <code>autoindent</code> включён. Если установлены параметры <code>smartindent</code> или <code>cindent</code>, то форматирование будет изменено в соответствующих специфических случаях.</td></tr>
<tr><td><code>autowrite (aw)</code></td><td>Булевский; (умолчание: 0). Записать содержимое текущего файла на диск если он был изменён и выполняется одна из следующих команд:
<code>:next</code>
<code>:rewind</code>
<code>:previous</code>
<code>:stop</code>
<code>:suspend</code>
<code>:tag</code>
<code>:!</code>
<code>:make</code>
<code>Ctrl/]</code>
<code>Ctrl/^</code></td></tr>
<tr><td colspan="2">Продолжение на следующей странице</td></tr>
</table>

Внутренние параметры редактора **Vim** (продолжение)	
Параметр	Значение
	Кроме того, содержимое изменённого файла будет записываться при переходе к другому файлу по командам: • `Ctrl/O` • `Ctrl/I` • `'{A-Z0-9}` • `` `{A-Z0-9} ``
`background (bg)`	Тип: строка; значения: dark, light. Устанавливает умалчиваемую цветовую гамму текста на терминале. Более детальное обсуждение можно найти использовав `:h option-list`.
`backspace (bs)`	Тип: число; значения: 0,1,2. 0 – символ `<BS>` используется также как в старом **vi**; 1 – символ `<BS>` может стирать символ конца строки; 2 – символ `<BS>` может использоваться до места, где начался ввод.
`backup (bk)`	Тип: булевский; значения: 0,1. Использовать (1) или не использовать резервную копию редактируемого файла.
`backupdir (bdir)`	Тип: строка; значение по умолчанию: `.,~/tmp,~/`. Список каталогов для создания резервных копий редактируемого файла.
`backupext (bex)`	Тип: строка; значение по умолчанию: `~`. Строка, которая добавляется в конец имени файла, чтобы образовать имя резервной копии редактируемого файла.
`binary (bin)`	Тип: булевский; значения: 0,1. Устанавливается перед редактированием двоичных (не текстовых) файлов. То же что параметр **-b** при вызове реактора (страница 11).
Продолжение на следующей странице	

Внутренние параметры редактора **Vim** (продолжение)	
Параметр	Значение
`breakat (brk)`	Тип: строка; значение по умолчанию: `^I!@*-+_`:,./?;. Этот параметр определяет какие символы могут вызвать разрыв строки, если включён параметр `linebreak`.
`browsedir (bsdir)`	Тип: строка; значение по умолчанию: `last`. Какой каталог используется для просмотра файлов. • `last` – используется тот же каталог, что использовался с последним просмотром; • `buffer` – использовать каталог родственного (related) буфера; • `current` – использовать текущий каталог.
`cindent (cin)`	Тип: булевский; значение по умолчанию: 0; локально для буфера. Разрешает автоматический сдвиг строк при вводе программ на языке **C**. Смотрите параметры `cinoptions` и `cinkeys`. Если вы не удовлетворены алгоритмом работы, порождаемым редактором под воздействием упомянутых параметров, то смотрите описание параметра `smartindent`.
`cinkeys (cink)`	Параметр содержит строку символов, которая управляет сдвигом строки в зависимости от того, какие символы или команды вводятся и в каком контексте. Значение по умолчанию: `cinkeys=0{,0},:,0#,!^F,o,O,e`.
`cinoptions (cino)`	Строка (умолчание: `""`). Определяется алгоритм каким образом `cindent` перемещает строки в программах на **C**.
Продолжение на следующей странице	

Внутренние параметры редактора **Vim** (продолжение)	
Параметр	Значение
`cinwords (cinw)`	Строка (умолчание: `if,else,while,do,for,switch`) означают ключевые слова которые начинают дополнительный сдвиг последующих строк, если установлены `smartindent` или `cindent`.
`cmdheight (ch)`	Число (умолчание=1) означает количество строк, которые отводятся на экране для командной строки. Увеличение значения может помочь избежать в ряде случаев задержек в работе из-за появления дополнительных сообщений.
`columns (co)`	Число (умолчание=80 или ширина терминала) означает количество колонок на экране дисплея. Обычно такой параметр не должен устанавливаться вручную.
`comments (com)`	Строка (умолчание: `/*,mb:*,el:*/,://,b:#,:%,:XCOMM,n:>,fb:-`). представляет собой разделённый запятыми список символов, которые могут начинать строку комментариев в редактируемом тексте. Полезно использовать когда возникает необходимость форматировать комментарии (подробности в `:h format-comments`).
`compatible (cp)`	Булевское (умолчание=1) означает, что **Vim** начинает работать в режиме совместимости со старым **vi**, если не найден файл с именем `.exrc`. Часто переключать значение этого параметра не рекомендуется из-за краевых эффектов.
`complete (cpt)`	Строка (умолчание: `.,b`) определяет каким образом работает завершение слова при вводе, если использованы команды `Ctrl/p` или `Ctrl/n`. Строка определяет тип завершения и места для поиска подходящих завершений. Следующие значения могут иметь место:
Продолжение на следующей странице	

Внутренние параметры редактора **Vim** (продолжение)

Параметр	Значение
	• `.` (точка) – просматривать текущий буфер; • `w` – просматривать другие окна; • `b` – просматривать другие загруженные буферы; • `u` – просматривать не загруженные буферы; • `k` – просматривать файлы, представленные с параметром `dictionary`; • `k[dict]` – просматривать файл с именем *dict* (может быть использовано несколько флагов `k`). Например, `:set cpt=k/usr/dict/*,k~/russian`. • `i` – просматривать текущий и включённые файлы; • `]` – использовать завершение ТЕГА; • `t` – то же самое, что `]`.
`confirm (cf)`	Булевское (умолчание=0). Когда параметр `confirm` включён, то определённые операции, которые обычно приводят к ошибке, например, чтение нового файла, если не сохранён старый, порождают диалог с вами. Вы сможете использовать восклицательный знак как обычно.
`cpoptions (cpo)`	Строка (умолчание: `aABceFs`). Имеется много значений; подробности в `:h cpoptions`.
`cscopeprg (csprg)`	Строка (умолчание: `cscope`). Определяет команду для выполнения программы с именем *cscope*, предназначенной для анализа и нахождения ключевых слов в большом количестве программ на **C**.

Продолжение на следующей странице

Внутренние параметры редактора **Vim** (продолжение)

<table>
<tr><th>Параметр</th><th>Значение</th></tr>
<tr><td>cscopetag (cst)</td><td>Булевское (умолчание:0) означает использование программы cscope для команд работы с тегами.</td></tr>
<tr><td>cscopetagorder (csto)</td><td>Булевское (умолчание:0) означает в каком порядке команда :cstag будет производить поиск.</td></tr>
<tr><td>define (def)</td><td>Строка (умолчание: ^#\s*define) означает образец, который будет использоваться когда потребуется найти в тексте макроопределение. Это образец для поиска, похоже на образец для команды поиска /. Приведённое умолчание хорошо подойдёт для языка C. Для C++ более удобна строка
^\(#\s*define\|[a-z]*\s*const\s*[a-z]*\)
Замечание. Если будете использовать данную строку в команде :set, то не забудьте ввести каждый обратный слеш дважды.</td></tr>
<tr><td>dictionary (dict)</td><td>Строка (умолчание: "") содержит разделённые запятыми имена файлов, в которых содержатся слова, которые будут использоваться в командах завершения ключевых слов. Каждый файл должен содержать список слов по одному или более слов на строку. Слова в строке должны разделяться пробелом. Обычно такие файлы хранятся в каталоге с именем /usr/dict/.</td></tr>
<tr><td>digraph (dg)</td><td>Булевское (умолчание: 0) означает режим ввода ДИГРАФОВ, т.е. специальных символов имеющих как правило установленный восьмой бит. Ввод ДИГРАФОВ производится в виде ввода двух символов в следующем порядке:
Символ1 <BS> Символ2
Подробности о ДИГРАФАХ в Vim можно найти посредством :h digraphs.</td></tr>
<tr><td>directory (dir)</td><td>Строка (умолчание: .,~/tmp,/var/tmp,/tmp) содержит разделённый запятыми список каталогов для хранения файлов временного использования (swap files). Подробности в :h directory.</td></tr>
</table>

Продолжение на следующей странице

Внутренние параметры редактора **Vim** (продолжение)	
Параметр	Значение
`edcompatible (ed)`	Булевское (умолчание:0) означает, что **Vim** запоминает значения флагов **c** и **g** в команде `:substitute`. Если вы вторично будете использовать команду `:substitute` с той же подстрокой, то будут использоваться запомненные флаги, если они не были введены явно. Использование команды `:substitute` с другой подстрокой не будет использовать запомненные с первой подстрокой значения флагов.
`equalalways (ea)`	Булевское (умолчание:1) означает, что редактор всегда делает все окна одинаковыми по размеру, если их количество изменяется.
`equalprg (ep)`	Строка (умолчание:`""`) содержит имя программы (фильтр), обычно для форматирования текста по команде редактора `=` *object*. Например, `=2o`. По этой команде программе, имя которой хранится в переменной `equalprg` будет передано 3 строки текста (текущую строку и две следующие). Результат выполнения программы будет записан на их место. Если строка `equalprg` пуста (умолчание), то будет произведено внутреннее форматирование указанных строк в соответствии со значениями внутренних параметров `cindent` или `lisp`. Замечание. Пробелы в строке `equalprg` должны вводиться с предшествующим обратным слешем.
`errorbells (eb)`	Булевское (умолчание:0) управляет звуковым сигналом при возникновении ошибки. Подробности `:h errorbells`.
`errorfile (ef)`	Строка (умолчание: `errors.err`) содержит имя файла для режима редактора **QuickFix** (смотрите описание команды `:cf` на странице 101, а также описание параметра **-q** на странице 11).
Продолжение на следующей странице	

Внутренние параметры редактора **Vim** (продолжение)	
Параметр	Значение
`errorformat (efm)`	Строка (умолчание: не приводится из-за длины) описывает формат сообщений об ошибках. Имеющиеся средства позволяют весьма детально описать в каком формате должны генерироваться сообщения об ошибках. Подробности в `:h errorformat`.
`esckeys (ek)`	Булевское (умолчание:0) позволяет включить распознавание специальных кодовых последовательностей, которые начинаются символом `<ESC>`, в режиме ввода.
`eventignore (ei)`	Строка (умолчание:`""`) представляет собой разделённый запятыми список событий, которые должны игнорироваться автокомандами. Например, `:set ei=WinEnter,WinLeave`
`expandtab (et)`	Булевское (умолчание:0, локально для буфера) позволяет вводить необходимое количество пробелов вместо знака табуляции.
`exrc (ex)`	Булевское (умолчание:0) позволяет читать инициализационные файлы в текущем каталоге.
`fileformat (ff)`	Строка (умолчание для UNIX: `unix`, локально для буфера) определяющая формат исходного текста. Возможные значения: `unix` – для систем типа UNIX, `dos` – для систем типа DOS, `mac` – для систем типа Macintosh. Подробности в `:h fileformat`.
`fileformats (ffs)`	Строка (умолчание для UNIX: `unix,dos`) определяющая какие форматы текста будут опробованы редактором текста при переключении на новый буфер, а также при чтении нового текста.
`formatoptions (fo)`	Строка (умолчание: `tcq`, локально для буфера) определяет способ автоматического форматирования текста. Каждая буква в строке описывает различные особенности форматирования. Подробности в `:h formatoptions`.
Продолжение на следующей странице	

Внутренние параметры редактора **Vim** (продолжение)	
Параметр	Значение
`formatprg (fp)`	Строка (умолчание:`""`) содержит имя программы форматирования текста, которая будет использоваться по команде редактора `gq` (смотрите также страницу 76). Если строка пуста, то производится внутреннее форматирование. Сравните с форматированием по команде `=` приведённым на странице 82.
`gdefault (gd)`	Булевское (умолчание:0) устанавливает режим, при котором по команде `:substitute` заменяются все подходящие подстроки, а не только первая встретившаяся. Появление в команде флага `g` инвертирует ситуацию.
`grepformat (gfm)`	Строка (умолчание: `%f:%l%m,%f  %l%m`) описывает формат вывода программы **grep**, которая используется в команде `:grep`.
`grepprg (gp)`	Строка (умолчание: `grep -n`) содержит имя программы, которая вызывается по команде редактора `:grep`.
`helpfile (hf)`	Строка (умолчание: `$VIM/doc/help.txt`) содержит имя файла описания редактора. Все такого вида файлы должны быть размещены в одном каталоге.
`helpheight (hh)`	Число (умолчание=20), означающее высоту окна, которое появляется в ответ на команду `:help`.
`highlight (hl)`	Строка (умолчание: не приводится по соображениям длины) представляет собой описание способа выделения различных элементов текста. Строка содержит пары символов, первый из которых описывает что выделить, а второй – как выделять. Подробности в `:h highlight`.
`hlsearch (hls)`	Булевское (умолчание:0) выделяет все появления в тексте подстрок поиска, которые вводятся командой `/`.
`history (hi)`	Число (умолчание=20) показывает сколько команд типа `:`, а также подстрок поиска запоминается редактором.
Продолжение на следующей странице	

Внутренние параметры редактора **Vim** (продолжение)	
Параметр	Значение
`ignorecase (ic)`	Булевское (умолчание:0) позволяет игнорировать регистр, в котором введена подстрока поиска.
`include (inc)`	Строка (умолчание: `^#\s*include`) содержит регулярное выражения для поиска в тексте программы операторов `include`. Эта строка используется в командах редактора `[i`, `]I`, `[d` и т.д. Внутренний параметр `isfname` используется, чтобы распознать имя файла в операторе `include`, откуда производится включение (include) текста.
`incsearch (is)`	Булевское (умолчание:0) позволяет немедленно выделить подходящие подстроки в тексте по мере ввода подстроки поиска. Замечание. Пока не введён символ `<CR>` позиция курсора остаётся неизменной несмотря на мелькание текста на экране.
`insertmode (im)`	Булевское (умолчание:0) заставляет **Vim** использовать режим ввода в качестве обычного режима.
`isfname (isf)`	Строка (умолчание: `@,48-57,/,.,-,_,+,,,$,~`) описывающая имя файла в операторах `include`.
`isident (isi)`	Строка (умолчание: `@,48-57,_,192-255`) описывающая, что включается в имена идентификаторов.
`isprint (isp)`	Строка (умолчание: `@,161-255`) описывающая, какие символы отображаются на экране непосредственно.
`iskeyword (isk)`	Строка (умолчание: `!-~,^*,^|,^"`) описывающая, какие символы могут использоваться при поиске подстрок и распознавании команд.
`keywordprg (kp)`	Строка (умолчание: `""`) содержит имя программы, которая используется в команде `K`. Если `keywordprg=man`, то по команде `K` будет выполнена команда системы `man` *keyword*, где *keyword* есть слово, на которое указывает курсор.
Продолжение на следующей странице	

Внутренние параметры редактора **Vim** (продолжение)	
Параметр	Значение
`langmap (lmap)`	Строка (умолчание: `""`) содержит строку перекодировки для различных языков программирования при вводе текста с клавиатуры. Когда редактор находится в режиме ввода, то производится перекодировка из кода клавиатуры в коды соответствующих символов конкретного языка. Например, `:set langmap=zy,yz,ZY,YZ`. По этой команде будет изменено значение символов `Z` и `Y`, а также `z` `y`.
`linebreak (lbr)`	Булевское (умолчание: 0) означает способ переноса строк при отображении на экране, не производя никаких изменений в файле. Перенос будет производится в тех местах, где стоит символ из строки `breakat`. При этом в противоположность комбинации параметров `wrapmargin` и `textwidth` не будет вставляться символ `<EOL>` в месте переноса.
`lisp`	Булевское (умолчание: 0, локально для буфера) приводит к установлению смещения текста от левого края к стандартам, которые приняты в языке **lisp**. Чтобы все это работало требуется также установить параметр `autoindent`.
`list`	Булевское (умолчание: 0, локально для окна) позволяет показать все знаки табуляторов и концы строк.
`listchars (lcs)`	Строка (умолчание: `eol:$`) содержит строки, которые используются в установленном режиме `list`.
Продолжение на следующей странице	

Внутренние параметры редактора **Vim** (продолжение)	
Параметр	Значение
	• `eol:`*c* - символ, которым будет отмечаться конец строки в режиме `list`. • `tab:`*xy* - два символа, которые используются чтобы показать положение табулятора. Первый символ используется один раз для указания, где находится табулятор. Второй символ используется, чтобы заполнить пространство, которое занимает табулятор. • `trail:`*c* - символ, который используется, чтобы показать последовательные пробелы в строке. • `extends:`*c* - символ, который, если выключен режим `wrap`, покажет в последней колонке экрана, что строка продолжается за пределы экрана.
`magic`	Булевское (умолчание: 1) предполагает изменение специальных символов, которые могут использоваться в регулярных выражениях, используемых для поиска.
`makeef (mef)`	Строка (умолчание: `/tmp/vim##.err`) содержит имя файла для вывода сообщений об ошибках при выполнении команды редактора `:make`. Два знака `##` (если имеются) заменяются на некоторое число, чтобы сделать имена файлов уникальными, тем самым гарантируется, что очередной вывод команды редактора `:make` не уничтожит предыдущий вывод. Этот файл не используется с командой `:cf`.

Продолжение на следующей странице

Внутренние параметры редактора **Vim** (продолжение)	
Параметр	Значение
`makeprg (mp)`	Строка (умолчание: `make`) содержит имя программы, которая будет вызвана по команде редактора `:make`. Строка может содержать символы `%` и `#`, которые интерпретируются также как в командной строке. Если вы планируете использовать знак `|`, то смотрите пример на странице 119.
`matchpairs (mps)`	Строка (умолчание: `(:),{:},[:]`) содержит пары символов, которые полагаются сопряженными при использовании команды редактора `%`, которая переводит курсор с одного символа сопряженной пары на другой. Если вы хотите добавить к парам, имеющимся по умолчанию, дополнительно пары `<` и `>`, то это можно сделать: `:set mps+=<:>`
`maxfuncdepth (mfd)`	Число (умолчание:100) определяет максимальную глубину функциональной рекурсии.
`maxmapdepth (mmd)`	Число (умолчание:1000) определяет максимальную глубину рекурсии подстановки строк по команде (командам) `:map`, чтобы избежать бесконечных циклов типа: `:map x y :map y x`
`maxmem (mm)`	Число (умолчание:512) определяет максимальную величину памяти выделяемую для одного буфера в Кбайтах.
`maxmemtot (mmt)`	Число (умолчание:2048) определяет максимальное количество памяти, которое может использоваться для всех буферов в Кбайтах.
`modeline (ml)`	Булевское (умолчание: 1; локально для буфера) приводит к проверке параметра `modelines`. Если выключено, то никаких действий.
Продолжение на следующей странице	

Внутренние параметры редактора **Vim** (продолжение)	
Параметр	Значение
`modelines (mls)`	Число (умолчание:5) определяет число строк в читаемом файле (в конце файла), которые используются для специальных установок параметров редактора посредством команды `:set`. Если `modelines`=0, то это то же, что выключен параметр `modeline`.
`more`	Булевское (умолчание: 1) вывод на экран приостанавливается, если экран заполнен.
`mouse`	Строка (умолчание: `""`) содержит режим использования мышки. Обычным значением для режима является символ `a` (включены все режимы). При этом необходимо обеспечить соответствующее значение строки типа терминала `term`.
`mousefocus (mousef)`	Булевское (умолчание: 0), работает только для режима **GUI**. Если включено, то активизируется то окно, на которое указывает курсор.
`mousehide (mh)`	Булевское (умолчание: 0) означает, что курсор мышки не виден во время ввода символов. Он появляется снова во время движения мышки.
`mousemodel (mousem)`	Строка (умолчание: `extend`) содержит обозначение режима использование мышки.
`nrformats (nf)`	Строка (умолчание: `octal,hex`) определяет какие числа распознаёт редактор когда выполняет команды сложения и вычитания `Ctrl/a` и `Ctrl/x` (смотрите страницы).
`number (nu)`	Булевское (умолчание: 0, локально для окна) вывести номер стоки перед каждой строкой на экране.
`paragraphs (para)`	Строка (умолчание: `IPLPPPQPP LIpplpipbp`) определяет макросы **nroff**, которые отделяют в тексте один параграф от другого. Они представляют собой пары символов.
Продолжение на следующей странице	

Внутренние параметры редактора **Vim** (продолжение)	
Параметр	Значение
paste	Булевское (умолчание: 0) если включено, то переводит редактор в режим paste, т.е. вы можете копировать текст из одного окна в другое окно. Режим paste имеет ряд особенностей. Если вы включили этот режим, то будут выключены или переустановлены ряд параметров редактора. В частности, перестанут работать команды :ab, :map и некоторые другие. Подробности в :h paste.
patchmode (pm)	Строка (умолчание: "") содержит окончание имени редактируемого файла, которое будет добавлено к имени файла и вновь образованное имя используется для сохранения исходной версии файла. Сохранение будет происходить только в том случае, если строка patchmode не пуста и параметр backupdir также не пуст.
path (pa)	Строка (умолчание: .,/usr/include,,) содержит список имён каталогов, которые просматриваются во время выполнения команд gf, [f,]f, ^Wf и других. Имена каталогов могут быть как абсолютными так и относительными. Например, :set path=.,/usr/local/include,/usr/include
previewheight (pvh)	Число (умолчание: 12) означает высоту окна для предварительного просмотра. Окно используется в команде :ptag и в команде Ctrl/w }, также приводящей к :ptag.
readonly (ro)	Булевское (умолчание: 0, локально для буфера) позволяет защитить файл от перезаписи. Но в то же время позволяет записать файл, использовав команду !w.
remap	Булевское (умолчание: 1) разрешает отображение задаваемое командой :map выполняться рекурсивно. Смотрите также параметр maxmapdepth на странице 88.
Продолжение на следующей странице	

Внутренние параметры редактора **Vim** (продолжение)	
Параметр	Значение
`report`	Число (умолчание: 2) означает число изменённых строк, о которых редактор будет сообщать. Для команды `:substitute` будет сообщено количество подстановок.
`ruler (ru)`	Булевское (умолчание: 0) показывать строку и колонку позиции курсора.
`rulerformat (ruf)`	Строка (умолчание: `""`) содержит формат для вывода информации при установленном параметре `ruler`. Формат тот же как для параметра `statusline`. Если параметр начинается с выражения `%wid(`, тогда `wid` определяет ширину сообщения в символах. По умолчанию ширина сообщения составляет 17 символов.
`scrollbind (scb)`	Булевское (умолчание: 0, локально для окна) означает, что данное окно будет перемещаться синхронно с другим (другими) окном (окнами), где установлен тот же параметр. Детали можно посмотреть `:h scroll-binding`, `:h scrollopt`.
`shell (sh)`	Строка содержит имя оболочки для выполнения команд в режимах `:!`. При смене содержания строки полезно проверить параметры `shelltype`, `shellpipe`, `shellslash`, `shellredir`.
`shiftwidth (sw)`	Число (умолчание: 8, локально для буфера) означает количество пробелов используемых для автоматического сдвига текста при вводе.
`swapfile (swp)`	Булевское (умолчание: 1; локально для буфера) означает, что для буфера необходимо использовать резервную копию редактируемого файла (swap file). Чтобы такая копия реально создалась необходимо, чтобы параметр `updatecount` (смотрите страницу 93) не был равен нулю.

Продолжение на следующей странице

Внутренние параметры редактора **Vim** (продолжение)	
Параметр	Значение
`swapsync (sws)`	Строка (умолчание: `fsync`) содержит имя функции, которая будет использоваться для синхронизации состояния буфера и резервной копии редактируемого файла на диске. Если для синхронизации используется `fsync`, то при неожиданной аварии операционной системы может оказаться, что резервная копия на диске значительно отличается от состояния текста, с которым вы работали в момент аварии. Если используется имя функции `sync`, то синхронизация более точная, однако она может приводить к задержкам во время работы, обусловленным копированием.
`syntax (syn)`	Строка (умолчание: `""`; локально для буфера) содержит имя синтаксиса с использованием которого составлен редактируемый файл. Например, текст LaTeX имеет имя `tex` и т.д.
`taglength (tl)`	Число (умолчание: 0) означает (если не 0) сколько символов принимается во внимание при анализе тегов.
`tagrelative (tr)`	Булевское (умолчание: 1) означает, что если используется файл тегов в другом каталоге, имена файлов в таком теговском файле относительны к каталогу, где находится теговский файл.
`tags (tag)`	Строка (умолчание: `./tags,tags`) означает имена файлов, которые используются в команде редактора `tag`. Если редактор скомпилирован с параметром `+emacs_tags` тогда будут распознаваться теговские файлы как в формате **Vim**, так и в формате **emacs**.
`term`	Строка, обычно содержит тип терминала из $TERM.

Продолжение на следующей странице

Внутренние параметры редактора **Vim** (продолжение)	
Параметр	Значение
`textwidth (tw)`	Число (умолчание: 0; локально для буфера) означает (если не 0) количество символов, которые можно ввести в строку. Если вы продолжаете ввод, то редактор автоматически разобьёт вводимую строку на две и продолжит ввод с начала следующей строки.
`title`	Булевское (умолчание: 0), если установлено приведёт к тому, что окно получит имя в виде `VIM - filename`.
`titlelen`	Число (умолчание: 85, максимум 30000) означает величину, которая ограничивает длину имени окна, которое появится с использованием параметра `title` (смотрите страницу 93). Ограничение не является идеальным алгоритмом, поскольку реальное количество символов зависит от размера используемого шрифта и многих других причин. Если `titlelen`=0, то используется полное имя файла.
`titlestring`	Строка (умолчание: `""`) содержит (если не пусто) заголовок окна терминала. Этот заголовок будет устанавливаться несмотря на содержание параметров `titlelen` и `title`. Например, `let &titlestring = hostname() . "/" . expand("%:p")`
`undolevels (ul)`	Число (умолчание: 1000) означает глубину возврата (количество undo) после выполненных изменений в тексте.
`updatecount (uc)`	Число (умолчание: 200) означает количество символов, после ввода которых будет записана на диск резервная копия редактируемого файла (swap file), которая позже может быть использована для восстановления после непредвиденного завершения работы редактора (смотрите, например, `:h crash-recovery`).
Продолжение на следующей странице	

Внутренние параметры редактора **Vim** (продолжение)

Параметр	Значение
`updatetime (ut)`	Число (умолчание: 4000) означает число миллисекунд, после которого, если ничего не печатается на клавиатуре, производится резервное копирование редактируемого файла (буфера). Смотрите также `:h crash-recovery`.
`verbose (vbs)`	Число (умолчание: 0) означает уровень диагностирования предпринимаемых редактором действий. Ниже приведён список дополнительных сообщений в зависимости от значения параметра `verbose`. • `>= 1` - если файл прочитывается командой редактора `:source`; • `>= 1` - если файл `viminfo` читается или записывается; • `>= 8` - какие файлы для каких групп автокоманд выполняются; • `>= 9` - каждая выполняемая автокоманда; • `>= 15` - каждая выполняемая функциональная строка.

Продолжение на следующей странице

Внутренние параметры редактора **Vim** (продолжение)	
Параметр	Значение
`viminfo (vi)`	Строка (умолчание: `""`) содержит (если не пусто) ряд параметров редактора, которые записываются по завершению работы редактора в специальный файл `viminfo`. Если данный параметр не пуст, то при старте редактора считывается файл `viminfo`. Подробности о назначении файла находятся в `:h viminfo-file`. В этом файле содержится информация об истории введённых команда редактора, истории регулярных выражений введённых для поиска, содержимое РЕГИСТРОВ (смотрите раздел 1.12), метки для указания положений внутри файлов (смотрите раздел 1.11) и ряд других данных.
Продолжение на следующей странице	

Внутренние параметры редактора **Vim** (продолжение)	
Параметр	Значение
	Ниже приведены параметры, которые могут быть установлены (а некоторые должны быть установлены) в файле `viminfo`. Параметры обозначаются одним символом, за которым следует значение. • `'` (одиночный апостроф) – максимальное число редактируемых файлов, для которых запоминаются положения МЕТОК. Этот параметр должен присутствовать, если строка `viminfo` не пуста. • `f` – если равно 0, то МЕТКИ не запоминаются. Если данный параметр отсутствует или его равно 1, то все метки запоминаются. • `r` – определяет начало пути к файлам, для которых не должны запоминаться МЕТКИ. Может быть несколько таких параметров. Такое может понадобиться, если файлы расположены на сменных носителях (CD диски и т.д.). • `"` (двойной апостроф) – определяет максимальное количество строк сохраняемых в каждом регистре.
Продолжение на следующей странице	

Внутренние параметры редактора **Vim** (продолжение)	
Параметр	Значение
	• `:` (двоеточие) – определяет максимальное количество позиций для сохранения истории команд редактора. Если не присутствует, то используется значение параметра `history`. • `/` (прямой слеш или наклонная черта) – определяет максимальное количество пунктов для сохранения истории использованных для поиска регулярных выражений. Если не присутствует, то используется значение параметра `history`. • `@` – определяет максимальное количество позиций для запоминания вводных строк. Если не присутствует, то используется значение параметра `history`. • `n` имя файла `viminfo`. – Должно следовать за `n` без пробела. Должен быть последним параметром. • `%` – если включено, то сохраняется список буферов. Пример: `:set viminfo='50,\"1000,:0,n~/vim/viminfo` `'50` – запоминается метки для последних 50 файлов; `/` – не определено, используется умолчание; `"1000` – запоминается содержимое регистров до 1000 строк каждый регистр; `:0` – не запоминается история команд редактора; `n~/vim/viminfo` – имя файла для `viminfo`; `%` – не определено, используется умолчание.

Продолжение на следующей странице

Внутренние параметры редактора **Vim** (продолжение)	
Параметр	Значение
`wildignore (wig)`	Строка (умолчание: `""`) содержит, если не пусто, список образцов. Если имя файла или каталога содержит данный образец, то он не будет использоваться, когда требуется завершить имя файла или имя каталога. Например, если вы использовали команду `:set  wildignore=*.o,*.obj` то такого типа имена файлов не будут использоваться, если потребуется завершить имя файла.
`wildmenu (wmnu)`	Булевское (умолчание: 0=off). Когда включено, завершение имён команд выполняется в расширенном режиме. Ввод символа `{wildchar` (часто это `<TAB>`) для завершения имени команды приводит к тому, что возможные завершения появляются над командной строкой.
`writeany (wa)`	Разрешает запись в любой файл в любое время, иными словами не требуется использовать символ `!` (восклицательный знак) после команды записи.

1.19 Команды редактора Ex

Редактор имеет список из более более чем 300 различных команд.

```
:h ex-cmd-index
```

Вряд ли имеет смысл описывать здесь все команды. многие команды выполняют похожие функции. Полный список команд и их описание можно найти с помощью команды

```
:help index.txt
```

В связи с этим мы остановимся на описании нескольких команд и групп команд. В специальном разделе 1.20 мы отметим ряд важных сценариев использования возможностей редактора **Vim**.

Таблица 1.20: Команды редактора **Vim**

Команды редактора **Vim**	
Параметр	Значение
`:!`	отфильтровать одну или группу строк, или выполнить внешнюю команду.
`:!!`	повторить предыдущую команду `:!cmd`.
`:!#`	то же самое что команда `:n`, где `n` есть номер строки, куда следует переместить курсор.
`:!*`	выполнить команды редактора **Ex**, которые содержатся в регистре.
`:!<`	сдвинуть текст влево на величину `shiftwidth`.
`:!=`	показать номер строки, где стоит курсор.
`:!>`	сдвинуть текст вправо на величину `shiftwidth`.
`:!@`	выполнить команды редактора **Ex**, которые содержатся в регистре.
`:N[ext]`	перейти к следующему файлу в списке файлов.
`:P[rint]`	напечатать строки.
`:X`	запросить ключ для шифрования файла.
`:a[ppend]`	добавить текст после определённой (или текущей) строки.
`:ab[breviate]`	показать все имеющиеся символы замены и их значения, а также режим, в котором действует замена: ввод, командный, или для обоих.
`:ab[breviate] abc`	показать заменяемые подстроки, которые начинаются с символов `abc`.
`:ab[breviate] abc new`	добавить замену строк `abc` на строки `new` в таблицу замен. Обычно такая операция выполняется для длинных выражений, например: `:ab lo LoadFarm` заменить длинное имя скрипта `LoadFarm` на короткую аббревиатуру `lo`. Эту аббревиатуру можно использовать как в поле команды, так и при вводе текста. если вам потребуется отменить действие команды `ab`, то сделать это можно посредством команды `:una`.
`:abc[lear]`	очистить таблицу замен, т.е. удалить все введённые по команде `:ab[breviate]` сокращения.

Продолжение на следующей странице

Команды редактора **Vim** (продолжение)	
Команда	Описание
`:al[l]`	открыть отдельное окно для каждого файла в списке аргументов.
`:am[enu]`	ввести новое меню для всех режимов (визуального, ввода, обычного и т.д.). Используется в основном в редакторе, где поддерживается **GUI**.
`:an[oremenu]`	ввести новое меню, которое не будет переотображаться.
`:ar[gs]`	показать список аргументов редактора.
`:argu[ment]`	перейти к определённому файлу, имя которого содержится в списке параметров.
`:as[cii]`	вывести значение кодовой комбинации символа, который находится под курсором.
`:au[tocmd]`	добавить новую команду к списку команд, которые **Vim** будет выполнять автоматически при наступлении определённого события. Список основных событий приведён в таблице 1.23.
`:aug[roup]`	определить имя группы автокоманд, которая будет использоваться в команде `:au[tocmd]`. Имя группы `end` или `END` определяет группу по умолчанию.
`:b[uffer]`	перейти к редактированию определённого буфера из списка активных буферов.
`:bN[ext]`	перейти к редактированию следующего буфера из списка активных буферов.
`:ba[ll]`	открыть окно для каждого файла в списке буферов.
`:bad[d]`	добавить файл к списку буферов (не загружая сам файл).
`:bd[elete]`	удалить заданные файлы из списка буферов.
`:bl[ast]`	перейти к последнему файлу в списке буферов.
`:bm[odified]`	перейти к следующему файлу в списке буферов, который был модифицирован.
`:bn[ext]`	перейти к следующему файлу в списке буферов.
`:bp[evious]`	перейти к предыдущему файлу в списке буферов.
`:br[ewind]`	перейти к первому буферу из списка буферов.
Продолжение на следующей странице	

Команды редактора **Vim** (продолжение)	
Команда	Описание
:brea[k]	когда используется внутри цикла, то следующей исполняемой командой будет первая команда после цикла.
:buffers	показать список буферов.
:bun[load]	освободить заданный буфер (по умолчанию – текущий).
c[hange]	заменить строку или группу строк.
cN[ext]	перейти к предыдущей ошибке.
:ca[abbrev]	то же самое, что :ab[breviate], но в командном строчном режиме.
:cal[l]	вызов функции, определённой пользователем или встроенной в **Vim**. Чтобы увидеть список доступных в данный момент функций, следует воспользоваться командой fu[nction]
:cc	перейти к определённой ошибке (смотрите сценарий ОТЛАДКА в разделе 1.6).
:cd	изменить текущий каталог.
:ce[nter]	расположить текст в центре строки.
:cf[ile]	прочесть файл с сообщениями об ошибках компиляции программы (смотрите сценарий ОТЛАДКА в разделе 1.6).
:ch[ange]	удалить и вставить строки текста.
:che[ckpath]	перечислить дополнительные файлы, если они имеются.
:cl[ist]	перечислить все ошибки компиляции (смотрите сценарий ОТЛАДКА в разделе 1.6).
:cla[st]	перейти к последней ошибке компиляции (смотрите сценарий ОТЛАДКА в разделе 1.6).
:clo[se]	закрыть текущее окно.
:cm[ap]	то же что map, но только для командного строчного режима.
:cnew[er]	перейти к более свежему списку ошибок.
:cnf[ile]	перейти к первой ошибке в следующем файле (смотрите сценарий ОТЛАДКА в разделе 1.6).
:co[py]	копировать строки.
Продолжение на следующей странице	

Команды редактора **Vim** (продолжение)	
Команда	Описание
:col[der]	перейти к более старому списку ошибок (смотрите сценарий ОТЛАДКА в разделе 1.6).
:com[mand]	определить команду пользователя (если без параметров – перечислить определённые пользователем команды).
:con[tinue]	перейти к оператору while, т.е. в начало цикла.
:d[elete]	удалить строки из текущего файла.
:di[splay]	отобразить содержание регистров.
:dig[raphs]	показать все введённые ДИГРАФЫ (смотрите также страницу 81).
:dj[ump]	переход по тексту к оператору #define.
:dl[ist]	показать список всех операторов #define.
:do[autocmd]	применить автокоманды к текущему буферу.
:doauto[all]	применить автокоманды ко всем загруженным буферам.
:ds[earch]	найти оператор #define.
:dsp[lit]	разделить окно **Vim** на два окна и перейти в новом окне к оператору #define.
:e[dit]	начать редактирование файла.
:ec[ho]	вывести результат выражения.
:echoh[l]	установить режим выделения символов для информации, которая выводится последующими командами :ec[ho].
:el[se]	часть команды :if.
:elsei[f]	часть команды :if.
:en[dif]	часть команды :if (завершение команды).
:endf[unction]	завершение пользовательской функции.
:endw[hile]	завершение цикла while.
:ex	то же самое, что и команда :edit.
:exe[cute]	выполнить результат выражения.
:exi[t]	то же что и xit, т.е. записать на диск буфер, если он был изменён и выйти из редактора.
:f[ile]	установить или показать текущее имя файла.
:files	перечислить все файлы из списка буферов.
:filet[ype]	включить/выключить определение типа файла.
:fin[d]	найти файл в списке каталогов path (смотрите также описание параметра path на странице 90).

Продолжение на следующей странице

Команды редактора **Vim** (продолжение)	
Команда	Описание
`:fix[del]`	установить значение кода для `<DEL>`.
`:fu[nction]`	определить пользовательскую функцию.
`:g[lobal]`	выполнять заданные команды над заданным множеством строк текста.
`:go[to]`	перейти к заданому байту в текущем буфере.
`:gr[ep]`	выполнить команду, имя которой содержится в параметре `grepprg` (по умолчанию – `grep -n` (смотрите также описание параметра `grepprg` на странице 84, а также пример на странице 118).
`:gu[i]`	запустить режим **GUI**.
`:h[elp]`	разделить имеющееся окно **Vim** и показать страницу руководства **Vim**. Если окно со страницей руководства было открыто ранее, то используется ранее открытое окно.
`:helpf[ind]`	в режиме **GUI** открыть диалоговое окно для ввода термина, о котором хотелось бы найти страницу руководства.
`:hi[ghlight]`	эта команда предназначена для управления выделением текста в различных условиях, например, в зависимости от используемого языка программирования.
`:hid[e]`	закрыть текущее окно, если оно не является единственным окном **Vim** на экране. При этом, буфер становится НЕВИДИМЫМ.
`:his[tory]`	вывести файл истории введённых команд редактора **Vim**.
`:i[nsert]`	начать ввод (вставку) текста в текущий буфер.
`:if`	это есть начало конструкции условий `:if expr` – *commands* – `endif`. Последовательность команд *commands* будет выполняться всякий раз, как только выражение `expr` даст в результате не нуль.
`:ij[ump]`	перейти к месту в буфере, в котором содержится заданная подстрока.

Продолжение на следующей странице

Команды редактора **Vim** (продолжение)	
Команда	Описание
`:il[ist]`	перечислить все места в буфере, где встречается заданная подстрока (практически то же что `[I` или `]I`), например, `il /слить/` – показать все места в тексте, где встречается подстрока /слить/.
`:int[ro]`	вывести начальное сообщение **Vim**.
`:is[earch]`	отобразить одну строку, которая удовлетворяет выражению, например, `:is /сообщ/`.
`:isp[lit]`	разделить окно **Vim** на два и перейти к заданной подстроке, например, `isp /язык/`.
`:j[oin]`	объединить соседние строки текста в одну строку.
`:ju[mps]`	вывести список переходов по тексту.
`:k`	определить метку в данном месте текста, например, `:k b` определить метку с именем `b` в месте, где стоит курсор.
`:l[ist]`	вывести заданные строки (по умолчанию – весь буфер).
`:la[st]`	перейти к последнему файлу в списке аргументов.
`:le[ft]`	выронять строку по левому краю.
`:let`	присвоить значение переменной или внутреннему параметру редактора.
`:ls`	показать список всех буферов.
`:m[ove]`	переместить строки.
`:ma[rk]`	определить метку с заданным именем в месте, где стоит курсор.
`:mak[e]`	выполнить внешнюю команду, имя которой содержится в параметре `makeprg` (смотрите описание параметра `makeprg` на странице 88).
Продолжение на следующей странице	

Команды редактора **Vim** (продолжение)	
Команда	Описание
`:map`	показать имеющиеся или ввести новые отображения одних символьных последовательностей в другие. Например, `map! i Институт Высокой Технологии` – после ввода такой команды, при вводе буквы `i` она немедленно будет заменена на три слова `Институт Высокой Технологии`. Но будьте осторожны. Заменяемая буква совпадает с командой начала ввода `i`. Может оказаться, что когда вы попробуете начать ввод текста, то получите неожиданный результат. Механизм отображения весьма гибок, подробности следует смотреть на страницах `:h map.txt`.
`:marks`	показать все установленные метки.
`:mes[sages]`	показать ранее выданные редактором сообщения.
`:mk[exrc]`	записать значения внутренних переменных, а также введённые отображения в файл (по умолчанию `.exrc`).
`:mks[ession]`	записать все параметры данной сессии в заданный файл.
`:n[ext]`	перейти к следующему файлу в списке параметров редактора.
`:new`	открыть новое gecnjt окно редактора **Vim**.
`:nu[mber]`	вывести строки текста вместе с номерами строк.
`:on[ly]`	закрыть все окна редактора, кроме текущего окна.
`:opt[ions]`	открыть новое окно редактора для просмотра и установки значений параметров.
`:p[rint]`	вывести строки текста (по умолчанию текущую строку).
`:pe[rl]`	выполнить команду языка **perl**.
`:pre[serve]`	записать содержание всех буферов в специфальный временный файл (своп-файл).
`:prev[ious]`	перейти к предыдущему файлу в списке аргументов.
`:pu[t]`	поместить текст из регистра после указанной строки (по умолчанию текущей).
`:pwd`	вывести имя текущего каталога.

Продолжение на следующей странице

Команды редактора **Vim** (продолжение)	
Команда	Описание
`:py[thon]`	выполнить команду языка **python**.
`:pyf[ile]`	выполнить скрипт языка **python**.
`:q[uit]`	закрыть текущее окно.
`:qa[ll]`	выйти из редактора.
`:r[ead]`	прочесть файл и добавить его в текущий буфер (по умолчанию после текущей строки). Другим вариантом использования команды является чтение вывода внешней программы, например, `:r !cmd` что означает: выполнить внешнюю команду с именем `cmd` и результат поместить в строки ниже курсора.
`:redi[r]`	перенаправить сообщения редактора в файл или в регистр редактора (смотрите также раздел 1.12). Пример перенаправления смотрите на странице 122.
`:reg[isters]`	показать содержимое всех регистров.
`:retu[rn]`	вернуться из пользовательской функции.
`:rew[ind]`	вернуться к первому файлу в списке аргументов.
`:ri[ght]`	выровнить текст по правому краю.
`:rv[iminfo]`	чтение и установка параметров из файла `viminfo`.
`:s[ubstitute]`	для каждой указанной строки найти подстроку и произвести замену.
`:sN[ext]`	разделить окно на два окна и перейти к предыдущему файлу из списка параметров.
`:sa[rgument]`	разделить окно на два окна и перейти к указанному файлу из списка параметров.
`:sal[l]`	открыть окно для каждого имени файла из списка аргументов.
`:se[t]`	показать или установить значения параметров редактора (смотрите также таблицу 1.18).
`:sf[ind]`	разделить окно на два и в новом окне начать редактировать заданный файл из каталога *path* (смотрите также описание параметра `path` на странице 90). Если файл не найден, то новое окно не создаётся.
Продолжение на следующей странице	

Команды редактора **Vim** (продолжение)	
Команда	Описание
`:sh[ell]`	запустить оболочку.
`:sni[ff]`	послать запрос системе **SNiFF+**. **SNiFF+** является системой управления массивами исходных текстов в больших проектах. Система **SNiFF+** позволяет использовать **Vim** в качестве средства редактирования и отображения текстов. С другой стороны, **Vim** может обращаться к системе **SNiFF+** с разнообразными запросами. Подробности можно почерпнуть в *http://www.takefive.co.at/*.
`:so[urce]`	прочесть и выполнить команды редактора из заданного файла.
`:star[tinsert]`	начать ввод с той позиции, где стоит курсор. Практически работвет также как команда `i` в обычном режиме (смотрите также страницу 6).
`:st[op]`	приостановить редактор и перейти в оболочку.
`:sw[apname]`	показать имя текущего файла резервной копии (своп-файла).
`:sy[ntax]`	включить выделение элементов текста текущего буфера в соответствии с синтаксисом (смотрите также описание параметра `syntax` на странице 92).
`:syncbind`	заставляет все окна редактора, где выданы команды `:syncbind`, перемещаться по тексту синхронно. Иными словами, если одно из окон установлено в начало текста, то и все остальные будут установлены в начало своих буферов.
`:t`	то же самое, что команда `:copy`.
`:tN[ext]`	то же самое, что команда `:tp[revious]` – перейти к предыдущему ТЕГУ.
`:ta[g]`	перейти к указанному ТЕГУ.
`:tags`	показать содержимое стека ТЕГОВ.
`:tcl`	выполнить команду языка **Tcl**.
`:tcld[o]`	выполнить команду языка **Tcl** для каждой строки в указанном диапазоне (по умолчанию весь буфер).
Продолжение на следующей странице	

Команды редактора **Vim** (продолжение)	
Команда	Описание
`:u[ndo]`	вернуть буфер к состоянию, которое он имел до последнего изменения.
`:una[bbreviate]`	удалить аббревиатуру из списка аббревиатур.
`:unl[et]`	удалить переменную.
`:unm[ap]`	удалить отображение символов из списка отображений.
`:up[date]`	записать буфер на диск, если буфер был изменён.
`:v[global]`	то же самое, что команда `g[lobal]!`. Команда выполняется в два шага: в первом шаге производится сканирование текста в указанном диапазоне (по умолчанию весь буфер) и отмечаются строки где встречается искомая последовательность, а во втором шаге производится требуемая замена в отмеченных строках. Смотрите пример на странице 109.
`:ve[rsion]`	Показать версию редактора, а также все установленные при компиляции параметры.
`:vi[sual]`	Если введено в режиме `Ex`, то редактор перейдёт в обычный режим. В противном случае, эта команда работает также как команда `e[dit]`.
`:vie[w]`	Редакторовать указанный файл в режиме ТОЛЬКО ЧТЕНИЕ.
`:w[rite]`	записать заданную часть буфера (по умолчанию весь буфер) на диск.
`:wa[ll]`	записать все изменённые буферы на диск. Буферы, которые не имеют имени файла или с признаком ТОЛЬКО ЧТЕНИЕ, не записываются.
`:wh[ile]` *expr commands* `endw[hile]`	выполнять последовательность *commands* (между `:wh[ile]` и `:endw[hile]`) до тех пор пока выражение *expr* не равно нулю. Если при выполнении команд *commands* возникла ошибка, то выполнение будет продолжено после `:endw[hile]`.
`:wn[ext]`	записать текущий буфер в файл и перейти к следующему файлу в списке аргументов.
`:wq`	записать текущий буфер и закрыть текущее окно или выйти из **Vim**.
Продолжение на следующей странице	

Команды редактора **Vim** (продолжение)	
Команда	Описание
:wqa[ll]	записать все изменённые буферы и выйти из **Vim**.
:wv[iminfo]	записать информацию viminfo в заданный файл (умолчание viminfo).
:x[it]	записать буфер, если он был изменён, и закрыть текущее окно или выйти из **Vim**, если окно было единственным.
:xa[ll]	то же что :wqa[ll].
:y[ank]	скопировать заданные строки в указанный регистр.
:z	вывести на экран заданные строки из буфера.
:~	повторить последнюю команду :substitute.

Пример 1.19.1 Пример замены подстроки

Например, команда
`:g/section/s//chapter/g`
заменит section на chapter во всём буфере. То же самое может быть реализовано командой:
`%s/section/chapter/g`

Таблица 1.21: Параметры при компиляции **Vim**

Параметры при компиляции **Vim**	
Обозначение параметра	Описание
+autocmd	включить возможность автокоманды, смотрите описание команды :autocmd на странице 100.
+browse	включить возможность команды :browse.
+builtin_terms	включить некоторые встроенные возможности терминалов.
+byte_offset	включить дополнительные возможности по сдвигу строк при вводе текстов на языке **C**.
+cindent	включить возможность автоматического сдвига строк при вводе текстов на языке **C**.
+cmdline_compl	включить возможность завершения командной строки при вводе команды редактора.
+cmdline_info	включить возможность отображения команды.
+comments	включить возможность форматирования комментариев в программе.

Продолжение на следующей странице

Параметры при компиляции **Vim** (продолжение)	
Обозначение параметра	Описание
`+cryptv`	включить возможность шифрования файлов.
`+cscope`	включить возможность поддержки интерфейса `cscope`. Программа `cscope` есть более мощное средство анализа текста и поиска тегов, чем программа `ctags`. Она представляет собой интерактивную программу для поиска фрагментов исходного текста в одном или множестве файлов содержащих тексты **C**, **lex** или **yacc**. В результате прохода программы строится таблица перекрёстных ссылок, которая используется в качестве базы для быстрого перемещения по текстовым файлам. Программа **cscope** не является свободно распространяемой в смысле **GPL**.
`+dialog_gui`	включить возможность подтверждения выполнения команды в режиме **GUI**.
`+dialog_con`	включить возможность подтверждения выполнения некоторых команд.
`+digraphs`	включить возможность использования диграфов (смотрите описание на странице 81).
`+emacs_tags`	включить возможность использования таблицы тегов в стандарте `emacs`.
`+eval`	включить возможность вычисления выражений посредством `eval` (смотрите раздел 1.13.
`+ex_extra`	включить возможности дополнительных команд форматирования `:left`, `:center`.
`+extra_search`	включить возможности дополнительных команд `hlsearch` и `incsearch`.
`+farsi`	включить возможность поддержки языка ФАРСИ.
`+file_in_path`	включить возможности поддержки команд `gf`, `Ctrl-w_f` (смотрите также 90).
`+find_in_path`	включить возможности поиска в файлах. Используется в командах `[I`, `:isearch`, `:checkpath`, `Ctrl-w_Ctrl-i`.
Продолжение на следующей странице	

Параметры при компиляции **Vim** (продолжение)	
Обозначение параметра	Описание
`+hangul_input`	включить возможности поддержки корейского языка.
`+insert_expand`	включить возможности поддержки завершения строки в режиме ввода.
`+linebreak`	включить возможности поддержки возможностей специального разрыва строк при вводе.
`+menu`	включить возможности поддержки `menu` в режиме **GUI**.
`+modify_fname`	включить возможности поддержки модификаторов имён файлов после знаков `%`, `#` и т.д.
`+perl`	включить возможности поддержки языка **perl**.
`+python`	включить возможности поддержки языка **python**.
`+quickfix`	включить возможности поддержки цикла отладки программы КОРРЕКЦИЯ – ТРАНСЛЯЦИЯ – ВЫПОЛНЕНИЕ – АНАЛИЗ ОШИБОК.
`+rightleft`	включить режим печати справа налево.
`+scrollbind`	включить режим синхронного просмотра окон редактора (смотрите также страницу 91).
`+smartindent`	включить режим автоматического интеллектуального сдвига текста при вводе.
`+sniff`	включить поддержку интерфейса к системе **SNiFF+**.
`+syntax`	включить поддержку ситаксического выделения элементов текста.
`+tcl`	включить поддержку интерфейса к интерпретатору **tcl**.
`+terminfo`	включить использование `terminfo` вместо `termcap`.
`+textobjects`	включить использование объектов текста таких как `aw`, `aW`, `iw`, `iW` и т.д.
`+title`	включить возможность придания заголовка окну редактора.
`+user_commands`	включить возможность поддержки команд пользователя.
`+viminfo`	включить возможность `viminfo`.

Продолжение на следующей странице

Параметры при компиляции **Vim** (продолжение)	
Обозначение параметра	Описание
`+visualextra`	включить возможность дополнительных команд в ВИЗУАЛЬНОМ режиме.
`+wildignore`	включить возможность игнорирования некоторых имён файлов, когда выполняется завершение имени файла.
`+xim`	включить возможность ввода в режиме **GUI**.

Таблица 1.22: Список файлов описания **Vim**

Список файлов описания **Vim**	
Имя файла	Значение
`intro.txt`	общее описание и обозначения используемые в документации по **Vim**.
`help.txt`	обзор редактора и краткий спаравочник по **Vim**.
`index.txt`	алфавитный список всех команд редактора.
`help-tags`	список тегов, куда вы можете перейти в описании редактора.
`howto.txt`	файл справочного руководства по редактору **Vim**.
`tips.txt`	различные примеры и советы по использованию **Vim**.
`quotes.txt`	хвалебные цитаты от разных пользователей.
`todo.txt`	ивестные проблемы и возможные расширения редактора.
`message.txt`	толкование сообщений об ошибках.
Основные способы редактирования	
`starting.txt`	запуск и инициализация **Vim**.
`editing.txt`	редактирование и сохранение файлов на диске.
`motion.txt`	команды по перемещению керсора.
`scroll.txt`	перемещение по тексту в окне редактора.
`insert.txt`	вставка и замена текста.
`change.txt`	удаление и замена текста.
`undo.txt`	вернуть текст в исходное сосотояние после замены.
`repeat.txt`	повторение команд редактора.
`visual.txt`	использование визуального режима.
Продолжение на следующей странице	

Список файлов описания **Vim** (продолжение)	
Имя файла	Описание
various.txt	различные команды редактирования.
recover.txt	восстановление после аварийного окончания редактирования.
Продвинутое редактирование	
cmdline.txt	редактирование в командной строке.
options.txt	описание всех параметорв редактора.
pattern.txt	регулярные выражения и команды поиска.
map.txt	механизм аббревиатур и отображения ключевых слов.
tagsearch.txt	поиск тегов и другие виды специального поиска.
quickfix.txt	реализация сценария редактирование – трансляция – коррекция ошибок.
windows.txt	команды использования многих окон и буферов.
syntax.txt	синтаксическое выделение элементов текста.
autocmd.txt	автоматическое выполнение команд при наступлении определённых событий.
eval.txt	вычисление выражений, команды условного перехода.
Специальные средства	
term.txt	использование различных типов терминалов и мышек.
digraph.txt	список используемых диграфов.
multibyte.txt	поддержка многобайтных символов.
farsi.txt	редактирование символов Фарси (Персидский).
hangulin.txt	ввод символов корейского языка.
rightleft.txt	ввод текста справа налево.
gui.txt	пользовательский графический интерфейс.
gui_x11.txt	графический интерфейс пользователя для X11.
Интерфейсы	
if_cscope.txt	использование системы **cscope** вместе с **Vim**.
if_perl.txt	интерфейс к языку **perl**.
if_python.txt	интерфейс к языку **python**.
if_sniff.txt	интерфейс к системе **SNiFF+**.
if_tcl.txt	интерфейс к языку **tcl**.

Продолжение на следующей странице

Список файлов описания **Vim** (продолжение)	
Имя файла	Описание
`os_unix.txt`	замечания по редактору в операционной системе типа **Unix**.

Таблица 1.23: Список событий редактора **Vim**

Список событий редактора **Vim**	
Обозначение события	Значение
`BufNewFile`	наступает, когда начинает редактироваться новый файл;
`BufReadPre`	наступает, когда начинает редактироваться существующий файл, но перед чтением в буфер;
`FileReadPre`	наступает, до чтения файла по команде `:r[ead]`
`FileReadPost`	наступает после чтения файла по команде `:r[ead]`
`FilterReadPre`	наступает до чтения файла из команды-фильтра;
`FilterReadPost`	наступает после чтения из команды-фильтра;
`FileType`	наступает, если установлен параметр `filetype`;
`StdinReadPre`	наступает, перед чтением текста в буфер с устройства стандартного ввода;
`StdinReadPost`	наступает после чтения текста в буфер с устройства стандартного ввода, но до выполнения `modelline`;
`BufWrite` или `BufWritePre`	перед записью полного буфера в файл;
`BufWritePost`	после выполнения записи полного буфера в файл;
`FileWritePre`	перед выполнением записи неполного буфера в файл;
`FileWritePost`	после выполнения записи неполного буфера в файл;
`FileAppendPre`	перед добавлением в файл;
`FileAppendPost`	после добавления в файл;
`FilterWritePre`	перед записью файла в команду-фильтр;
`FilterWritePost`	после записи файла в команду-фильтр;
Продолжение на следующей странице	

Список событий редактора **Vim** (продолжение)	
Обозначение события	Описание
`FileChangedShell`	наступает после выполнения любой команды оболочки (например, `:!date`), если время модификации файла изменилось по сравнению с временем начала редактирования;
`FocusGained`	когда **Vim** обнаружил указатель мышки на окне (только для версии **GUI**);
`FocusLost`	когда **Vim** определил, что мышка ушла с экрана (только для версии **GUI**);
`BufEnter`	после ввода буфера; после события `BufReadPost`;
`BufLeave`	до переключения на другой буфер;
`BufUnLoad`	перед освобождением буфера, например, после `BufWritePost` или перед `BufDelete`;
`BufHidden`	сразу после того, как буфер стал невидимым;
`BufCreate`	сразу после создания нового буфера;
`BufDelete`	перед удалением буфера;
`WinEneter`	после входа в другое окно;
`WinLeave`	перед выходом из окна;
`GUIEnter`	после успешного старта версии **GUI** и открытия первого окна;
`VimEnter`	после успешного старта **Vim** и создания всех необходимых окон;
`VimLeavePre`	перед завершением **Vim** и перед записью файла `.viminfo`;
`VimLeave`	перед завершением **Vim**, но после записи файла `.viminfo`;
`User`	только когда автокоманды выполняются по команде пользователя `:doautocmd`.

1.20 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.

1.20.1 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**

1.20.2 Просмотр лога системных сообщений /var/log/messages

`view /var/log/messages`

Это часто удобно делать, не потому, что другие программы плохи, а просто из соображений экономии усилий, чтобы оставаться в одной и той же среде, в которой вы наработали определённые навыки и освоили (приготовили) удобные для вас инструменты.

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.

1.20.3 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`, как это описано в таблице 1.18.

Часть команд зависят от параметров, значения которых устанавливаются во время трансляции редактора **Vim**. Значения таких параметров можно получить командой редактора `:version`. А описания параметров дано в таблице 1.21.

1.20.4 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`. Описание механизма меток можно посмотреть в разделе 1.11. 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**, которая обсуждается в разделе 1.14. 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`

а также в разделе 1.6.

1.20.5 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]` (страница 104). Далее, по таблице можно перемешаться (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.

1.20.6 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, описанных в разделе 1.6.

1.20.7 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 (см. команду `v` на странице 6).

2. Выполнить команду редактора `K` (стр. 30) с предварительно установленным внутренним параметром редактора `keywordprg` (стр. 85).

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

Чтобы выполнить перечисленные шаги, выполним прежде следующrие 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")
}
```

1.20.8 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 (смотрите также раздел 1.12).

Чтобы завершить вывод сообщений в файл следует использовать команду

`redi END`

1.20.9 Примеры использования элементов внутреннего языка Vim

Пусть нам требуется вывести на экран последовательные значения переменной i. Это можно выполнить командой редактора:

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

Описание команды `:execute` приведено на странице 54.

Другой пример, требуется вывести выделенное сообщение:

`:echohl WarningMsg | echo "Don't panic!" | echohl None`

Здесь первая команда `:echohl WarningMsg` устанавливает тип выделения текста `WarningMsg` (смотрите описание команды `:echohl` на странице 54). Далее следует команда собственно вывода текста на экран

```
echo "Don't panic!"
```

Последняя команда `echohl None` убирает выделение текста, которому подвергартся весь текст, выводимый посредством команды редактора `echo`.

1.20.10 Примеры использования регулярных выражений при построении теговской таблицы программой etags

Приведённые примеры опираются на свойства программы **etags**, которая описана в разделе 1.15.

```
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=`.

1.20.11 Что делать, если что-то происходит не так

Во время редактирования с использованием редактора **Vim** может произойти что-то непредвиденное, что, впрочем, случается вовобще 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.

1.21 Полезная информация о файлах редактора

Такую информацию об установленном редакторе **Vim** можно почерпнуть в следующих файлах:

`usr/share/vim/vim54/doc/*.txt` – список файлов документов редактора. Чтобы получить список можно использовать команду редактора `:help doc-file-list`

`/usr/share/vim/vim54/doc/tags` – список тегов, которые используются для нахождения необходимых документов.

`/usr/share/vim/vim54/syntax/syntax.vim` – синтаксическая инициализация редактора.

`/usr/share/vim/vim54/syntax/*.vim` – синтаксические файлы для разных языков программирования.

`/usr/share/vim/vimrc` – общесистемный файл инициализации редактора.

`/usr/share/vim/gvimrc` – общесистемный файл инициализации редактора с режимом **GUI**.

`/usr/share/vim/vim54/optwin.vim` – скрипт используется для команды `:options`.

`/usr/share/vim/vim54/menu.vim` – общесистемное меню для инициализации редактора **gvim**.

`/usr/share/vim/vim54/bugreport.vim` – скрипт для генерации сообщения об ошибках в редакторе. Полезно выполнить команду `:help bugs`

`/usr/share/vim/vim54/filetype.vim` – скрипт для определения типа файла по его имени. Полезно посмотреть `:help filetype`

Наконец, текущее состояние редактора **Vim** лучше смотреть на сайте *http://www.vim.org/*.

Предметный указатель