# Строковый редактор ed

*Андрей С. Кухар*
*plan9@kp.km.ua*

**АБСТРАКТНО**

В этом документе дается описание строкового редактора *ed* версии операционной системы Plan 9. В наименовании здесь также не обошлось без трэш-кино «Plan 9 From Outer Space», а точнее без его режиссера, Эда Вуда (Ed Wood), как раз в честь него назван почтенный редактор :)

## Введение

Текстовые редакторы используются людьми различных профессий. И любознательные пользователи, и писатели, и программисты, и системные администраторы, работая в Plan 9, время от времени обращаются к помощи текстового редактора. Но даже если вы не писатель, вам по меньшей мере следует знать, как редактировать конфигурационные файлы системы, чтобы заставить Plan 9 работать нужным образом.

Хотя способ использования компьютера и диктует тип необходимого программного обеспечения, по мнению большинства пользователей половину времени работы компьютера занимает работа с текстовыми файлами. Поскольку текстом может быть и документ, и электронное письмо, и исходный текст программы, и сценарий оболочки, и исходные данные для программы, и системный конфигурационный файл, то текстовый редактор является широко применяемой программой.

## История редактора

В 1970 году Bell Labs приобрели машину PDP-11 для проекта по подготовке текста. При этом был получен лишь процессор и основная память, накопители на магнитных дисках отсутствовали. Все ПО хранилось на бумажных лентах, как таковой, операционной системы не было. Элементарная версия Unix была переписана (все еще на ассемблере) для этой машины, теперь включая простой строковый редактор *ed*, который был написан Кеном Томпсоном (Ken Thompson) как программа рендеринга для форматизаторов текстов `runoff` и `roff`. На редактор повлиял популярный в те времена QED. Из ранних свойств редактора можно выделить основную строковую ориентацию, радикально упрощенные регулярные выражения (в нем присутствовал лишь метасимвол *, никаких чередований и круглых скобок) и отсутствие функций для работы с несколькими буферами. Последующие версии *ed* для Unix (уже написанные на C) усложнялись и были более пригодны для работы; была добавлена возможность обратного обращения в регулярных выражениях, которые теперь совсем не включали ни регулярные языки, ни свободные от контекста языки, но входили в контекстно-зависимые языки.

Редактор обрел заслуженную популярность среди пользователей Unix, «Ученые в области computer science любят *ed* не потому что он пришел первым, а потому что он стандартный. Все остальные любят *ed* потому что это *ED*!», так говаривали они.

Когда в конце 80-х начались работы над проектом Plan 9, одной из первых переписанных программ (т.е. оставленных из мира Unix) был почтенный (venerable) редактор *ed*. Автором «новой» версии стал некто Том Дафф (Tom Duff) — автор оболочки `rc`, системы растровой графики, библиотеки панелей, броузера `mothra` и большого количества других полезных

вещей. Plan 9 версия *ed* отличается от ранних Unix версий необходимыми изменениями, вызванными операционной системой, для уведомлений взамен сигналам и т.п., плюс поддержкой UTF-8. Кстати, Том даже портировал Plan 9 версию редактора (которую он считает лучшей из истинных потомков оригинального *ed*) обратно в Unix, и запускает ее на работе (в Pixar) на машине с IRIX и дома на FreeBSD.

## Причины и рекомендации

Если вы хотя бы поверхностно знакомы с редакторами Sam и Acme, то вполне логичным прозвучит вопрос о затрате времени на, казалось бы, устаревшую, примитивную и, наверное, весьма уродливую программу «старой школы». Дело в том, что *ed* выполняет некоторые операции весьма успешно, несмотря на свой возраст. О стабильной работе в среде Plan 9 не стоит и говорить. Кроме того, *ed* легко запускать из файлов-сценариев и он незаменим при работе без графического режима, скажем, на файловом сервере.

С вступительной частью закончено, если желание разобраться в *ed* не оставило вас, тогда читайте дальше. Рекомендуемый путь изучения *ed* — это чтение документа с одновременным использованием редактора для выполнения примеров и упражнений. (Помощь опытных пользователей также весьма полезна.) В приложении приведен полный список всех команд *ed* с кратким описанием.

## Приступим

Здесь и далее в документе предполагается, что вы успешно вошли в систему и создали окно допустимого размера. Для запуска редактора выполните команду:

```
ed
```

У редактора *ed* НЕТ символа приглашения, точнее есть, но это пустая строка :). Как правило, редактированию подвергается некий временный файл, который вы впоследствии можете скопировать в любой другой файл. Этот временный файл называется *буфером редактирования*, так как он выполняет функции буфера между вводимым вами текстом и файлом, куда записываются изменения. Например, введите такие строки:

```
a
это строка 1
это строка 2
это строка 3
это строка 4
```

Теперь в новой строке введите точку «`.`». В результате этих действий в буфер будут записаны четыре строки, чтобы просмотреть их, введите:

```
1,4p
```

где 1,4 — это диапазон (интервал) номеров строк, а p — команда просмотра строк.

Теперь попробуйте выполнить следующее:

```
2p
```

для вывода строки под номером 2. Если же просто ввести:

```
p
```

то результатом будет содержимое текущей строки (которой сейчас является строка 2). По умолчанию, большая часть команд *ed* манипулируют именно текущей строкой.

Если указан аргумент файл, *ed* применяет к нему команду e (см. ниже); т.е. файл считывается в буфер *ed*, дабы его можно было редактировать.

Опции редактора:

> - подавляет вывод количества байт командами e, r и w, а также выдачу приглашения ! после команды !команда.
>
> -o (для внешнего канала) записывает весь вывод на стандартный файл ошибок в отличие от сохранения командой w. Если файл не задан, то файлом запоминания стает /fd/1.

## Основы

### Буфер

Каждый раз, когда вы запускаете *ed*, для выполнения всех ваших команд выделяется специальная область внутренней памяти машины. Эта область называется *буфером*. При редактировании файла, его содержимое копируется в этот буфер и вы работаете именно с ним. Замена оригинала производится только при сохранении изменений.

### Команды

Как и обычные команды Plan 9, команды редактора вводятся с клавиатуры и заканчивается нажатием клавиши Enter. Выполнение команды начинается после того, как вы нажали Enter. Большинство команд состоят из одного символа, перед которым могут указываться номер строки или диапазон номеров строк. Большая часть команд редактора работает с текущей строкой (см. ниже «Номера строк»). Многие команды имеют аргумент имя файла или строка.

### Номера строк

После каждой команды, изменяющей количество строк в буфере, *ed* выполняет перенумерацию строк. Каждой строке буфера всегда присвоен номер. Многие команды редактора используют либо одиночные номера строк, либо диапазон номеров, которые передаются команде в качестве аргументов. Эти аргументы определяют конкретные строки в буфере, над которыми нужно выполнить команду (команды).

## Использование редактора

### Вход и выход из *ed*

Как уже описывалось выше, для запуска редактора следует ввести команду:

```
ed
```

Также может использоваться команда такого вида:

```
ed имя файла
```

где имя файла — это имя нового или уже существующего файла. Для завершения сеанса работы с *ed* используется команда q (от *quit*):

```
q
```

или комбинация клавиш ctrl-d. В ответ вы получите символ приглашения операционной

системы Plan 9 (точнее символ приглашения оболочки `rc`). Если вы еще не записали изменения в файл, *ed* лаконично (как всегда) предупредит вас, что при выходе произойдет потеря изменений:

```
?
```

Если нестерпимое желание выхода из редактора не оставило вас, введите команду q (или ctrl-d) еще раз.

Для «легального» выхода из редактора вам нужно ввести 2 команды:

```
w
q
```

т.е. завершение работы *ed* с предшествующим сохранением изменений.

**Добавления текста: команда «a»**

При вызове *ed* без аргумента файла, создается новый буфер. Для заполнения буфера нужно набрать текст или скопировать его из файла. На данном этапе изучения работы редактора рассмотрим первый вариант, со вторым разберемся немного позже.

В терминологии *ed* про текст, с которым вы работаете, принято говорить, что он находится в буфере. Буфер подобен временному хранилищу информации, которую вы добавляете, редактируете и записываете в файл на диске.

Для управления процессом работы с текстовыми файлами *ed* владеет набором специальных команд.

Изучение команд начнем с команды добавления текста a (от *append* или *add*). После ее выполнения, все вводимые вами строки будут добавляться в буфер, к примеру:

```
a
Настало время
для всех нормальных пацанов
не слабо погудеть.
.
```

Чтобы закончить добавление, введите на пустой строке символ точки «`.`» или нажмите Del (при этом редактор выведет ? и перейдет в командный режим). Точка «`.`» сообщает *ed*, что вы закончили ввод.

После завершения добавления текста в буфере будут следующие три строки:

```
Настало время
для всех нормальных пацанов
не слабо погудеть.
```

Символы «a» и «`.`» отсутствуют, так как они не являются частью текста. Чтобы добавить еще строки к уже имеющемуся тексту, дайте еще раз команду a и продолжайте ввод.

**Запись файла: команда «w»**

Для записи содержимого буфера в файл используется команда w (от *write*). Выполнение этой команды приводит к копированию содержимого буфера в указанный файл, старое содержимое файла при этом удаляется. Например, чтобы сохранить текст в файле с именем text, введите:

```
w text
```

В ответ *ed* сообщит количество записанных в файл символов (байт). Вы должны увидеть примерно следующее:

```
61
```

(Подсчету подлежат также пробелы и символы новой строки.) Команда записи не изменяет содержимое буфера, а лишь копирует его, поэтому вы можете продолжать добавлять в него текст. При вызове *ed* командой ed имя файла, команда w, заданная без аргумента, записывает буфер в файл с именем имя файла.

Следует также помнить, что *ed* всегда работает с копией файла, оригинал не редактируется, пока вы не дадите команду w.

*Упражнение 1:*

Запустите *ed* и создайте какой-нибудь текст:

```
a
... текст ...
.
```

Сохраните его в файл с помощью команды w. Теперь покиньте *ed* командой q и просмотрите его содержимое, скажем, командой `cat`:

```
cat имя файла
```

или `pr`:

```
pr имя файла
```

**Редактирование файла: команда «e»**

Для копирования содержимого файла в буфер используется команда e (от *edit*). Чтобы загрузить в редактор какой-нибудь текст, возьмем, к примеру, три строки с текстом «Настало время» и т.д. из предыдущего примера, выполните команду:

```
e text
```

При этом в буфер загружен весь файл text и на экране появится количество его символов.

```
61
```

*Если в буфере до этого уже была какая-либо информация, она уничтожается.*

Если вы использовали команду e для считывания файла в буфер, то в команде w вам необязательно указывать имя файла. Редактор *ed* помнит имя файла, которое последним использовалось в команде e и команда w по умолчанию будет делать запись в этот файл. Таким образом, сеанс работы с редактором может быть таким:

```
ed
e file
[процесс редактирования]
w
q
```

**Изменение имени файла для записи: команда «f»**

Чтобы узнать имя файла, в который производилась последняя запись, можно использовать команду f (от *file*). Для этого выполните f без аргументов. С помощью команды f можно также менять имя файла запоминания редактора. Например, в результате выполнения

следующих команд:

```
ed precious
f junk
```

в буфер будет загружен файл precious и затем команда f сохранит его содержимое в файле junk. Исходный файл precious не подвергнется изменениям.

**Считывание файла: команда «r»**

Бывают ситуации, когда нужно считать в буфер файл, не стирая имеющейся до этого в нем информации. Для этой цели подходит команда r (от *read*). К примеру, по команде:

```
r text
```

файл text считывается и добавляется в конец имеющейся в буфере информации. Предположим, что вы выполнили команды:

```
e text
r text
```

После этого в буфере будет находиться *две* копии файла text (шесть строк):

Также как и команды w и e, после выполнения команды r на экран выводится количество считанных в буфер символов.

*Упражнение 2:*

Поэкспериментируйте с командой e на различных файлах. Возможно, вы получите сообщение ошибки ?name, где name — это указанное вами имя файла, это означает, что такой файл не существует или вы допустили опечатку в имени или, возможно, у вас нет разрешения для чтения этого файла или записи в него. Убедитесь, что команда

```
ed имя файла
```

эквивалентна

```
ed
e имя файла
```

Что делает следующая команда?

```
f имя файла
```

**Просмотр строк: команда «p»**

Для вывода содержимого буфера (или его частей) используется команда печати p (от *print*). К примеру, чтобы просмотреть первые две строки буфера редактирования (строки с 1 по 2), введите:

```
1,2p
```

*Ed* выведет:

```
Настало время
для нормальных пацанов
```

Предположим, что вам нужно просмотреть *все* строки буфера. Вы, конечно, можете дать команду, скажем, 1,3p, если знаете, что в буфере именно три строки. Но, как правило, вам редко будет известно точное количество строк буфера, и поэтому в *ed* имеется специальный

символ (метасимвол), обозначающий «номер последней строки в буфере» — это знак доллара $. Попробуйте такую команду:

```
1,$p
```

В результате будут выведены *все* строки (с первой по последнюю).

Чтобы напечатать *последнюю* строку буфера, введите:

```
p,$p
```

Довольно простая команда, не так ли? Но *ed* позволяет сократить ее до такого вида:

```
$p
```

Вы можете просмотреть любую строку, указав перед p в команде ее номер, к примеру:

```
1p
```

выведет на экран только первую строку буфера:

```
Настало время
```

*Для вывода одной строки вводить символ p необязательно*:

```
$
```

при этом *ed* выведет последнюю строку из буфера.

Вы можете также использовать символ $ в комбинациях наподобие этой:

```
$-1,$p
```

в результате будет напечатано две последние строки буфера.

Следующей возможностью, предоставляемой редактором, является указание с помощью знаков + (плюс) и - (минус) смещения относительно номеров строк, обозначаемых долларом $ или точкой `.`. Команда

```
$-1
```

выводит предпоследнюю строку из буфера. Для вывода последних шести строк, дайте команду:

```
$-5,$p
```

Если в буфере меньше шести строк, то появится сообщение об ошибке.

Команда

```
.-3,.+3p
```

печатает текущую строку, включая три строки перед ней и три строки после нее. (В данном случае знак + указывать необязательно, т.е. команда .-3,. 3p выполнит те же действия.)

Вы можете также использовать знаки + и - в качестве номеров строк, например, введя команду:

```
-
```

вы переместитесь назад на одну строку в буфере относительно текущей строки. Вы можете указать в командной строке столько знаков -, на сколько строк вам надо переместиться назад.

Например:

```
- - -
```

вернет вас на три строки назад также как и команда

```
-3
```

Команда

```
 -3,+3p
```

эквивалентна следующей

```
?.-3p+3p
```

**Вывод текущей строки: команда «.»**

Предположим, что в буфере имеются следующие шесть строк:

```
Настало время
для всех нормальных пацанов
не слабо погудеть.
Настало время
для всех нормальных пацанов
не слабо погудеть.
```

Если вы введете:

```
1,3p
```

то *ed* покажет:

```
Настало время
для всех нормальных пацанов
не слабо погудеть.
```

если же теперь введете

```
p
```

на экране появится:

```
не слабо погудеть.
```

Эта строка является третьей строкой в буфере и последней строкой, с которой вы работали. Вы можете еще раз ввести p и *ed* будет продолжать ее печатать. Это происходит потому, что *ed* помнит номер строки, с которой вы работали последней, и при необходимости использует его в командах. К этой строке можно обратиться с помощью точки `.`. Точка является таким же номером строки, как и $, она обозначает «текущую строку» или «строку, над который вы только что выполнили какую-нибудь операцию». Точкой можно пользоваться различными способами. Например, команда:

```
 .,$p
```

печатает все строки, начиная с текущей (и включая ее) и до конца буфера. В нашем примере это строки с 3 по 6.

Некоторые команды изменяют значение номера текущей строки, некоторые нет. Команда p устанавливает значение для точки равным номеру последней напечатанной строки. В данном примере текущей станет строка 6.

Точка часто используется в комбинациях типа:

```
.+1
```

что эквивалентно

```
.+1p
```

Эта команда выводит следующую строку. Это один из способов последовательного просмотра буфера редактирования. Вы можете также ввести:

```
.-1
```

При этом будет выведена строка перед текущей. Еще один полезный пример, печатающий три строки перед текущей:

```
.-3,.-1p
```

Помните, что все эти команды изменяют значение номера текущей строки. Вы в любой момент можете узнать ее номер, выполнив команду:

```
.=
```

в ответ на которую *ed* выводит значение номера.

Перед командой p можно вообще не указывать номер, или указывать один или диапазон из двух номеров строк (если указываются два номера, то первый всегда должен быть меньше второго номера). Если номер не указан, то выводится текущая строка. Если задан только один номер, то *ed* печатает строку с этим номером и она становится текущей. Если указан диапазон, то будут выведены все строки, входящие в него, и текущей станет последняя строка.

Каждое нажатие на клавишу Enter эквивалентно команде:

```
.+1p
```

Введите ее, затем попробуйте ввести знак -. Результат будет тот же, что и у команды:

```
.-1p
```

*Упражнение 3:*

Потренируйтесь в работе с командой p над каким-нибудь текстом. Вы, возможно, столкнетесь с тем, что невозможно вывести строку с номером 0 или строку с номером, большим, чем количество строк в буфере, а также с невозможностью вывода строк в обратном порядке, например

```
3,1p
```

**Удаление строк: команда «d»**

Предположим, что вам необходимо удалить три лишние строки из буфера. Для этой цели предназначена команда d (от *delete*). Ее действие подобно действию команды p за исключением того, что d удаляет строки вместо их вывода. Строки, которые требуется удалить, указываются в команде точно так как в команде p. Например, команда:

```
4,$d
```

удаляет строки, начиная с четвертой по последнюю. В нашем примере после выполнения этой команды осталось только три строки и вы можете проверить это, выполнив:

```
1,$p
```

Обратите внимание на то, что текущей является 3-я строка. Указатель `.` устанавливается на следующую вслед за последней удаленной строку, если только в результате удаления не произошла очистка буфера. В этом случае указатель `.` устанавливается на текущую строку $.

*Упражнение 4:*

Поэкспериментируйте с командами a, e, r, w, p и d пока не убедитесь, что вы понимаете, как они работают и как в этих командах можно указывать номера строк.

Попробуйте указать номера строк в командах a, r и w. Вы обнаружите, что команда a добавляет строки после строки с указанным номером; команда r считывает файл в буфер после указанной строки (а не обязательно в конец буфера); команда w записывает в файл только указанные вами строки, а не весь буфер. Все это бывает иногда очень удобно. Например, вы можете вставить файл в начало буфера командой:

```
0r имя файла
```

или вставить строки в начале буфера введя:

```
0a
[ . . . текст . . . ]
.
```

Помните, что команда

```
.w
```

*очень сильно* отличается от:

```
.
w
```

так как первая записывает в файл только одну строку, в то время как последняя — весь буфер.

**Замена текста: команда «s»**

Одной из наиболее важных команд редактора является команда s, которая используется для замены отдельных слов или букв в строке или группе строк. Также она используется для исправления опечаток и ошибок.

Предположим, что в результате опечатки первая строка буфера выглядит так:

```
Настало врем
```

В слове время отсутствует последняя буква я. Чтобы исправить ошибку, вы можете использовать команду s следующим образом:

```
1s/врем/время/
```

В результате чего в строке 1 символы врем будут заменены на время. Чтобы проверить корректность замены, введите:

```
p
```

чтобы просмотреть исправленную строку:

```
Настало время
```

Обратите внимание, что после выполнения команды замены, строка, в которой делались изменения, стала текущей.

Команда замены имеет следующий синтаксис:

```
начало-строки,конец-строки s/заменить что/чем/
```

В результате действия команды *вся* последовательность символов между первой парой слэшев (косых черт) заменяется на всю последовательность символов из второй пары слэшев во всех строках, начиная с номера *начало-строки* и кончая номером *конец-строки*. В каждой строке будет заменено только первое вхождение последовательности между первой парой слэшев. Замена сразу всех вхождений *заменить что* рассматривается позднее в этой главе. Правила нумерации строк такие же как для команды p, за исключением того, что текущей становится последняя измененная строка. Если замена не произошла и текст не изменялся, то и номер текущей строки также *не* изменяется. При этом появляется сообщение об ошибке ?.

Таким образом, можно ввести:

```
1,$s/ашипка/ошибка/
```

и исправить первое вхождение последовательности ашипка на ошибка в каждой строке файла.

Если в команде замены не указываются номера строк, замена будет сделана только в текущей строке.

Команда:

```
s/что-нибудь/что-нибудь другое/p
```

делает исправления в текущей строке и затем печатает ее для контроля. (Обратите внимание на то, что команда p указана в той же строке, что и команда замены. За небольшим исключением, команду p можно указывать и после других команд.)

Также можно дать команду:

```
s/строка//
```

в результате которой первая найденная в текущей строке последовательность символов строка будет найдена и удалена. Такая команда бывает удобна для удаления лишних слов в строке или букв в слове. Например, если у вас есть строка:

```
Настало00 время
```

вы можете ввести:

```
s/00//p
```

чтобы получить:

```
Настало время
```

Помните, что // (два стоящих рядом слэша) означают «отсутствие символов», а не пробел. *Есть* разница. (Смотрите ниже другое значение //.)

*Упражнение 5:*

Потренируйтесь с командой замены. Посмотрите, что происходит, когда вы заменяете в строке слово, встречающееся в ней несколько раз. Например, если вы введете:

```
a
накладная на товар
```

```
.
s/на/не на/p
```

То строка примет вид:

```
не накладная на товар
```

Команда замены меняет только первое вхождение искомого текста в строке. Чтобы выполнить замену всех имеющихся вхождений, добавьте в команде замены символ g (от *global*):

```
s/.../.../g
```

Попробуйте использовать в качестве разделителей вместо слэшев любые другие символы.

**Контекстный поиск: команда «/.../»**

После того как вы познакомились с командой замены, можно перейти к изучению следующей важной функции редактора — контекстного поиска.

Предположим, у вас в буфере редактирования наличествует такой текст:

```
Настало время
для всех нормальных пацанов
не слабо погудеть.
```

Предположим, вы хотите найти строку, в которой есть словосочетание не слабо, чтобы потом заменить его на плотно. Когда у вас только три строки в буфере, довольно просто их просмотреть и найти нужную строку. Но если буфер содержит сотни строк, такой способ уже не годится. Контекстный поиск является простым методом нахождения требуемой строки независимо от ее номера. Для этого необходимо указать между слэшами искомую последовательность символов:

```
/шаблон/
```

Например, команды

```
/не слабо/
```

достаточно, чтобы найти нужную строку с этими символами. В команде контекстного поиска второй слэш необязателен, т.е. предыдущую команду можно записать так:

```
/не слабо
```

Команда поиска делает строку, в которой найдена искомая последовательность текущей и выводит ее на экран.

```
не слабо погудеть.
```

Редактор *ed* осуществляет поиск со строки с номером .+1 до конца буфера, затем переходит в начало буфера и продолжает поиск до строки с номером точка. *Ed* просматривает все строки в буфере до тех пор, пока не найдет нужный текст или не дойдет до строки с номером точка. Если искомая последовательность не найдена, появляется стандартное лаконичное сообщение об ошибке (?). В противном случае, *ed* печатает найденную строку. Вы можете также указать обратное направление поиска, заключив искомую последовательность в вопросительные знаки ? вместо /, например:

```
?не слабо?
```

или

```
?не слабо
```

Это бывает удобно, когда вы точно знаете, что искомая строка расположена до текущей.

Слэш и знак вопроса являются единственными символами, которые можно использовать для обозначения контекстного поиска. (Если вы получите неожиданные результаты при использовании символов

```
^       .       $       [       *       \       &
```

см. главу «Шаблоны и регулярные выражения».)

Вы можете одновременно выполнять поиск *и* замену, например:

```
/не слабо/s/не слабо/плотно/p
```

После выполнения этой команды вы получите:

```
плотно погудеть.
```

Данная команда выполняет сразу три действия: поиск по шаблону, выполнение замены и вывод измененной строки.

Выражение /не слабо/ здесь является шаблоном для контекстного поиска. В самом простом случае все выражения для поиска состоят из последовательности символов, заключенной между слэшами. Контекстный поиск может использоваться для определения номера строки и последующего его использования в других командах. Рассмотрит несколько примеров.

Предположим, что в буфере есть строки:

```
Настало время
для всех нормальных пацанов
не слабо погудеть.
```

Номера строк, определяемые выражениями:

```
/Настало/+1
/нормальных/
/погудеть/-1
```

являются командами контекстного поиска и все они определяют одну и ту же строку (строку 2). Чтобы сделать изменения в строке 2, введите:

```
/Настало/+1s/нормальных/конкретных/
```

или

```
/нормальных/s/нормальных/конкретных//
```

или

```
/погудеть/-1s/нормальных/конкретных//
```

Выбор команды определяется только удобством работы. Чтобы просмотреть все три строки, выполните команду:

```
/Настало/,/погудеть/p
```

или

```
/Настало/,/Настало/+2p
```

(или подобную им). Первый способ лучше в случае, если вы не знаете количество строк для вывода.

Помните, что выражение контекстного поиска является обычным номером строки и его можно применять везде, где требуется указание номеров строк.

Предположим, вы выполняете поиск слова штучка:

```
/штучка/
```

и когда строка с ним выводится на экран, вы обнаруживаете, что это не та строка и вам надо повторить поиск. Набирать повторно команду с шаблоном не требуется, так как можно использовать специальную конструкцию:

```
//
```

которая означает повторение поиска предыдущего текста. Она может быть повторена произвольное количество раз. Повторить поиск в обратном направлении можно командой:

```
??
```

// можно использовать в команде замены в качестве шаблона, подлежащего замене. Например, выполните следующие действия:

```
/штучка/
```

*Ed* выведет на экран строку, содержащую штучка.

```
s//загогулина/p
```

В результате слово штучка заменится на загогулина.

*Упражнение 6:*

Поработайте с командой контекстного поиска. Просмотрите весь буфер в поисках какой-нибудь последовательности символов. Попробуйте использовать команды контекстного поиска в качестве номеров строк в командах замены, вывода и удаления. (Их можно также использовать в командах r, w и a.) Выполните поиск в обратном направлении, указывая вместо / знак вопроса ?.

**Изменение и вставка текста: команды «c» и «i»**

В этом пункте рассматриваются команда изменения c (от *change*), применяющаяся для изменения или замены одной или более строк, и команда вставки i (от *insert*), использующаяся для вставки одной или более строк.

Команда c используется для замены группы строк строками, вводимыми с клавиатуры. Например, чтобы изменить строки с .+1 по $ на какие-либо другие, введите:

```
.+1,$c
[ . . . напечатайте здесь нужные вам строки . . . ]
.
```

Строки, которые вы напечатаете после команды c и до команды ., заменят указанную группу строк. Это удобно при замене строки или группы строк с ошибками.

Если в команде указан только один номер строки, то заменена будет только одна строка. (Строк замещения может быть сколько угодно.) Если номер строки вообще не указан, то заменяется текущая строка. После выполнения команды c текущей становится строка, которая вводилась последней.

Команда i похожа на команду добавления a. Например, чтобы вставить строки перед строкой, содержащей слово штучка, введите:

```
/string/i
[ . . . напечатайте здесь строки, которые необходимо вставить . . . ]
.
```

Текст, вводимый после команды i, вставляется перед указанной строкой. Если номер строки не указан, то текст вставляется перед текущей строкой. После выполнения команды текущей становится последняя вставленная строка.

*Упражнение 7:*

Команда c аналогична последовательному выполнению команды удаления и вставки строк. Проверьте это, выполнив команды:

```
начало,конец d
i
...текст...
.
```

и

```
начало,конец c
...текст...
.
```

Они не будут работать в точности одинаково в том случае, если удалению подверглась последняя строка.

Поэкспериментируйте с командами a и i и вы увидите, что они отличаются только тем, что команда:

```
номер-строки a
. . . текст . . .
.
```

вставляет строки *после* строки с указанным номером, а команда:

```
номер-строки i
. . . текст . . .
.
```

*перед* этой строкой. Если номер строки не указан, подразумевается текущая строка.

**Перемещение строк: команда «m»**

Команда перемещения m (*move*) позволяет переместить группу строк из одного места в буфере в другое. Предположим, вам нужно поместить первые три строки в конец буфера. Это можно сделать следующим способом:

```
1,3w temp
$r temp
1,3d
```

Здесь temp является именем временного файла. Намного проще это сделать одной командой:

```
1,3m$
```

Она перемещает строки с 1 по 3 в конец буфера. В общем случае команда перемещения m выглядит так:

```
нач.строка,кон.строка m после-какой-строки-вставить
```

В команде перемещения должна указываться строка, после которой необходимо поместить перемещаемый текст. Строки, которые необходимо переместить, можно указывать с помощью команд контекстного поиска. Предположим, что у вас есть текст:

```
Первый параграф
. . .
конец первого параграфа.
Второй параграф
. . .
конец второго параграфа.
```

Вы можете поменять параграфы местами командой такого типа:

```
/Второй/,/конец второго/m/Первый/-1
```

Обратите внимание на -1, текст помещается после нее. Текущей становится последняя перемещенная строка.

В качестве примера рассмотрим часто употребляемую команду, позволяющую менять порядок следования двух соседних строк. Предположим, вы находитесь на первой строке. Тогда команда:

```
m+
```

перемещает текущую первую строку через строку после текущей. Если вы находитесь на второй строке, то команда:

```
m-
```

перемещает текущую строку через строку вверх.

Команда m более эффективна чем запись, удаление и повторное считывание. Основная трудность при использовании команды m заключается в том, что в случае применения команд поиска для задания перемещаемой группы строк и адреса, куда их надо переместить, необходимо быть очень внимательными, в противном случае вы можете переместить не те строки. Поэтому если вы будете выполнять действия постепенно, вам легче будет себя проверить на каждом шаге. Также рекомендуется выполнять команду w перед выполнением любой сложной команды, тогда, если вы ошиблись, легко вернуться к предыдущему состоянию.

Дополнительную информацию о том, как перемещать текст, см. в разделе «Создание меток на строки в файле» в этой главе.

**Глобализация: команды «g» и «v»**

*Глобальные* команды g и v используются для выполнения одной или более команд редактирования над всеми строками, которые содержат (g) или не содержат (v) указанную последовательность символов. Например, команда:

```
g/план/p
```

выводит на экран все строки, содержащие слово план. На текст, заключенный между слэшами, распространяются те же правила и ограничения, что и в командах поиска и замены.

К примеру:

```
g/^\./p
```

показывает все форматирующие команды в файле. По поводу использования символов ^ и \ см. в разделе «Шаблоны и регулярные выражения».

Команда v аналогична g за исключением того, что она работает со строками, не содержащими указанный текст. Например, команда:

```
v/^\./p
```

выводит на экран все строки, кроме тех, которые начинаются с точки.

После команд g и v может следовать любая команда. Например, следующая команда удаляет все строки, начинающиеся с «.»:

```
g/^\./d
```

Следующая команда удаляет все пустые строки:

```
g/^$/d
```

Наверное, наиболее полезной командой, которая может использоваться вместе с глобальными командами, является команда замены. Например, чтобы заменить слово ПЛАН на план сразу во всем файле и проверить, что замена на самом деле произведена, выполните команду:

```
g/ПЛАН/s//план/gp
```

Обратите внимание, мы применили // в команде замены для обозначения текста, использовавшегося в последней команде поиска, в нашем случае план. Команда p выполняется для каждой строки, в которой найдено искомое слово.

Глобальные команды просматривают файл дважды. За первый проход они помечают все строки, в которых найден указанный текст. На втором проходе каждая помеченная строка по очереди становится текущей и выполняется вторая команда.

Например, команда:

```
g/^\.P/+
```

печатает строки, начинающиеся с команды .P. Помните, что знак + обозначает строку, следующую после текущей.

Команда:

```
g/тема/?^\.H?p
```

ищет все строки, содержащие слово тема, затем просматривает файл назад в поиске строки, начинающейся с .H и выводит ее. Другая команда:

```
g/^\.EQ/+,/^\.EN/-p
```

печатает все строки, находящиеся между строками, начинающимися с форматирующих команд .EQ и .EN.

Перед глобальными командами также можно указывать номера строк, для которых будет выполнена глобальная команда.

Под управление глобальной команды можно передать более одной команды. Например, скажем, вам нужно заменить x на y и a на b во всех строках, содержащих слово график. Для этого введите команду:

```
g/график/s/x/y/\
s/a/b/
```

Бэкслэш \ сообщает команде g, что команда продолжается на следующей строке. В противном случае, команда g заканчивается на строке, не содержащей \.

Обратите внимание на то, что команда замены в команде g не может использоваться для вставки новой строки.

Команда:

```
g/x/s//y/\
s/a/b/
```

не будет работать так, как вы ожидаете. Дело в том, что запоминаемый шаблон для поиска и замены // в этой команде будет меняться с x на a и наоборот. Поэтому для корректности данную команду следует переписать в таком виде:

```
g/x/s/x/y/\
s/a/b/
```

Можно выполнять команды a, c и i как часть глобальной команды. Как и для других многострочных команд, добавляйте бэкслэш в конце каждой строки, за исключением последней. Чтобы добавить .nf и .sp перед каждой строкой, содержащей .EQ, введите:

```
g/^\.EQ/i\
.nf\
.sp
```

Указывать завершающую команду i точку `.` нет необходимости, так как имеются еще другие команды, которые должны быть выполнены под управлением глобальной команды.

**Показ знаков табуляции и управляющих символов: команда «l»**

В редакторе *ed* для вывода содержимого буфера кроме команды p есть еще команда l (от *list*). l делает видимыми символы, которые обычно (в команде p) не выводятся, например, знаки табуляции, символы забоя и прочие управляющие символы. Если вы выводите командой l строку, содержащую такие символы, то символы табуляции будут отображены как \t, символы забоя как \b, бэкслэши или обратные косые черты (не путайте с чертями :) как \\, непечатаемые символы — слэшем \, буквой x с четырьмя шестнадцатеричными цифрами.

Длинные строки поддаются делению, начиная со второй и последующие под-строки выводятся с отступом на один стоповый символ табуляции. Если последний символ в строке является пробелом, то за ним следует \n.

**Функция отмены: команда «u»**

Иногда после выполнения замены в строках вы можете обнаружить, что этого не надо было делать, или делать, но совсем по-другому. Команда отмены u (от *undo*) восстанавливает предыдущее содержимое первой адресуемой строки (sic), которая должна быть последней строкой, в которой была выполнена подстановка (sic вдвойне).

**Создание меток на строки в файле: команда «k»**

Команда k дает возможность присвоить конкретной строке метку и в последующем обращаться к этой строке, указывая ее метку вместо номера. Это бывает полезно при перемещении строк. Например, команда:

```
kx
```

присваивает текущей строке метку x. Если перед символом k в команде указать номер строки,

то будет помечена эта строка. (В качестве меток можно использовать только одиночные символы.) К помеченной строке можно обратиться командой:

```
'x
```

Обратите внимание на использование здесь одинарной кавычки '. Метки очень удобны при перемещении блоков. Пометим сначала начало блока, который нужно переместить:

```
ka
```

и затем конец:

```
kb
```

Теперь перейдем в место, куда требуется поместить блок, и введем:

```
'a,'bm.
```

В каждый конкретный момент времени строка может иметь только одну метку.

**Копирование строк: команда «t»**

Ранее упоминалась идея сохранения строк, которые часто используются, для сокращения времени, необходимого для их ввода. В редакторе *ed* для копирования группы строк в любое место предназначена команда t. Ее использование бывает часто легче, чем запись и чтение.

Команда t аналогична команде m, только вместо перемещения строк в указанное место, она их копирует. Например:

```
1,$t$
```

дублирует содержимое буфера редактирования.

Обычно команда t используется для создания последовательности строк, которые не сильно отличаются друг от друга. Например, вы можете ввести (за шарпами — комментарии):

```
a
Настало время для всех нормальных пацанов не слабо погудеть.
.
t.                          # делаем копию
s/пацанов/братков/          # немного изменяем
t.                          # делаем еще одну копию
s/Настало/Вчера было/       # еще изменяем
```

В результате ваш файл будет выглядеть так:

```
Настало время для всех нормальных пацанов не слабо погудеть.
Настало время для всех нормальных братков не слабо погудеть.
Вчера было время для всех нормальных пацанов не слабо погудеть.
```

**Выход в оболочку rc: команда «!»**

Иногда бывает удобно временно выйти из редактора для выполнения команды Plan 9. Команда выхода ! позволяет осуществлять это.

Если вы введете:

```
!команда
```

ваше текущее состояние редактора будет сохранено и будет выполняться команда операционной системы Plan 9. Когда выполнение команды закончится, *ed* выведет

восклицательный знак !. Начиная с этого момента вы можете продолжать редактирование.

## Шаблоны и регулярные выражения

Вы могли заметить, что иногда происходят непонятные действия, когда вы используете такие символы как точка `.`, знак доллара $, звездочка * в контекстном поиске и замене. Дело в том, что *ed* обрабатывает эти символы как специальные. Например, при контекстном поиске или в команде замены точка `.` обозначает не символ точки, а шаблон для любого символа, т.е. команда:

```
/x.y/
```

будет искать строку, в которой между x и y может стоять любой символ. Вот полный список специальных символов, при использовании которых могут возникнуть проблемы:

```
^       .       $       [       *       ]       \       /       &
```

В следующих пунктах описывается как использовать эти символы для описания шаблонов текста в командах поиска и замены. Эти шаблоны называются *регулярными выражениями* и используются также во многих командах и утилитах Plan 9.

Напомним, что символ g, стоящий в конце команды замены, вызывает замену всех вхождений указанного текста. Команды:

```
s/что/чем
```

и

```
s/что/чем/g
```

различаются только тем, что первая заменяет только первое слово что в строке на чем, и, если в строке имеются еще слова что, то они не будут заменены. Вторая же команда заменяет *все* вхождения слова что во всем файле.

После любой из форм команд замены можно указывать p или l для вывода содержимого строки. Например:

```
s/что/чем/p
s/что/чем/l
s/что/чем/gp
s/что/чем/gl
```

Перед любой из команд замены s можно указать один или два номера строк для задания группы строк, в которой требуется произвести замены. Так, команда:

```
1,$s/очипятка/опечатка/
```

заменяет *первое* вхождение слова очипятка на опечатка в файле. А команда:

```
1,$s/очипятка/опечатка/g
```

заменяет *все* вхождения слова очипятка на опечатка в файле.

Если вы добавляете p или l в конце любой из команд замены, то выводиться будут не все измененные строки, а только последняя измененная строка. Позднее мы объясним, как просматривать все измененные строки.

### Точка: `.`

Первый метасимвол, который мы рассмотрим — это точка «`.`». Если она указывается в левой

части команды замены s, или в команде поиска, то обозначает любой одиночный символ. Так, команда поиска:

```
/x.y/
```

будет искать строку, в которой x и y разделены одним любым символом, например:

```
x+y
x-y
x y
xzy
```

и т.п.

Так как точка является метасимволом, обозначающим любой символ, она позволяет работать со специальными символами, показываемыми командой l. Предположим, у вас есть строка, которая показана командой l в таком виде:

```
э\tто
```

Допустим, вам нужно удалить символ \t, который соответствует символу табуляции. Наиболее очевидной кажется команда:

```
s/\t//
```

но она не сработает. Другой вариант — это перепечатать всю строку. Это довольно разумно, в случае, если строка не слишком длинная. Другим путем решения проблемы является использование метасимвола точки. Так как \t на самом деле представляет собой только один символ, то, если ввести команду:

```
s/э.то/это/
```

то задача будет выполнена. В этой команде точка будет обозначать любой символ, стоящий между буквами э и т.

Команда:

```
s/./,/
```

заменяет первый символ в строке на запятую. Специальное значение символа точки можно отменить, указав перед ним бэкслэш \.

В зависимости от применяемого контекста, метасимвол точки может принимать различные значения. В приводимой ниже команде используются сразу все три возможных значения метасимвола точки `.`

```
.s/./\./
```

Первая точка в команде обозначает номер текущей строки. Вторая точка является метасимволом, соответствующим любому символу в этой строке. И только третья точка обозначает обычный символ точки «`.`». (Помните также, что точка используется для завершения ввода командами a и i.) В правой части команды замены символ точки не имеет никакого специального значения. Если вы выполните указанную команду над строкой:

```
Настало время.
```

то в результате получите:

```
.астало время.
```

что, скорее всего не совсем то, что вам требуется. Чтобы заменить точку в конце

предложения на запятую, введите:

```
s/\./,/
```

Специальное значение метасимвола отменяется бэкслэшем, стоящей перед ним.

**Бэкслэш:** \

Так как метасимвол точка `.` обозначает любой символ, то возникает вопрос, что делать, когда требуется указать именно символ точки. Например, как преобразовать строку:

```
Настало время.
```

в

```
Настало время?
```

Бэкслэш отменяет все специальные значения у символа, следующего сразу после него, в частности, \ делает из метасимвола точки `.` обычный символ, и вы можете использовать его в команде замены:

```
s/\./?/
```

Пара символов \. рассматривается редактором *ed* как один обычный символ точки.

Бэкслэш можно также использовать при поиске строк, содержащих специальные символы. Предположим, вы ищете строку, в которой имеется .DE в начале строки. Команда поиска:

```
/.DE/
```

не выполнит того, что вам надо, так как она будет искать строки с текстом типа:

```
JADE
FADE
MADE
```

так как точка будет соответствовать любому символу. Но, если вы укажете команду:

```
/\.DE/
```

то будет найдена нужная вам строка.

Бэкслэш можно использовать для отмены специального значения и других символов кроме точки. Давайте попробуем найти строку, содержащую символ \. Команда:

```
/\/
```

не будет работать, так как в данной команде символ \ не является контекстом для поиска, а обозначает, что следующий за ним символ / больше не служит для обозначения границ контекста. Правильная команда выглядит вот так:

```
/\\/
```

Небольшое замечание об использовании бэкслэшев и специальных символов: в командах замены в качестве разделяющих символов вы можете применять любые символы (а не только слэши). В командах контекстного поиска слэши обязательны. Например, чтобы удалить все слэши в строке:

```
//exec //sys.fort.go // etc...
```

введите команду:

```
s:/::g
```

в результате будет:

```
exec sys.fort.go  etc...
```

Когда вы добавляете текст командами a, i или c, то бэкслэш не имеет специального действия и вы можете вводить ее как обыкновенный символ.

*Упражнение 8:*

Придумайте две команды замены, каждая из которых преобразует строку:

```
\x\.\y
```

в строку:

```
\x\y
```

Есть несколько решений, например:

```
s/\\\.//
s/x../x/
s/..y/y/
```

**Знак доллара: $**

Метасимвол $ обозначает конец строки. Допустим, у вас есть строка:

```
Настало
```

и вы хотите добавить слово время в конец строки. Примените знак $ следующим образом:

```
s/$/ время/
```

и вы получите:

```
Настало время
```

Пробел перед словом время в команде необходим, так как в противном случае вы получите:

```
Наставловремя
```

Чтобы заменить в следующем предложении:

```
Настало время, для всех нормальных пацанов,
```

вторую запятую на точку, введите команду:

```
s/,$/./
```

и вы получите:

```
Настало время, для всех нормальных пацанов.
```

В данном контексте знак доллара $ указывает редактору, какую запятую надо заменить. Без знака доллара в команде изменению подверглась бы первая запятая в строке:

```
Настало время. для всех нормальных пацанов,
```

Чтобы изменить:

```
Настало время.
```

на

```
Настало время?
```

можно дать команду:

```
s/.$/?/
```

Также как и метасимвол `.`, $ имеет несколько значений в зависимости от контекста. В команде:

```
$s/$/\$/
```

первый знак обозначает последнюю строку файла, второй относится к концу этой строки, а третий является обычным символом $, который должен быть добавлен на этой строке.

**Галочка: ^**

Символ ^ обозначает начало строки. Например, вам надо найти строку, начинающуюся с и. Если вы введете:

```
/и/
```

то, скорее всего, вы сначала найдете несколько строк с и в середине строк, и потом только нужную вам строку. Но, если вы укажете команду:

```
/^и/
```

то существенно сузите рамки поиска.

Используя символ ^ вы можете, например, вставить что-нибудь в начало строки:

```
s/^/ /
```

Эта команда вставляет пробел в начале текущей строки.

Метасимволы можно объединять. Чтобы найти строку, которая содержит только символы .P, вы можете дать команду:

```
/^\.P$/
```

**Звездочка: ***

Допустим, у вас есть строка, которая выглядит так:

```
текст x   y текст
```

где под текст понимается какой-нибудь текст, а между x и y стоит неизвестное количество пробелов. Предположим, вам нужно заменить все пробелы между x и y на один единственный пробел. Строка слишком длинная, чтобы вводить ее заново, и пробелов слишком много, чтобы их пересчитывать. Для решения таких задач имеется специальный метасимвол *. Конструкция, состоящая из символа и следующей за ним звездочки, означает любое количество стоящих подряд таких символов. В нашем случае, чтобы заменить все пробелы между x и y, достаточно ввести команду:

```
s/x *y/x y/
```

Звездочка может использоваться с любым символом, а не только с пробелом. Если бы наш пример выглядел так:

```
текст x----------y текст
```

то все минусы можно заменить на один пробел командой:

```
s/x-*y/x y/
```

Предположим теперь, что наш исходный текст выглядит так:

```
текст x.................y текст
```

Если вы дадите команду:

```
s/x.*y/x y/
```

то результат ее выполнения может быть весьма непредсказуем. Если в строке нет больше x и y, то замена сработает. Метасимвол . обозначает любой символ, и поэтому конструкция .* соответствует любой последовательности одинаковых символов, и данная команда несмотря на вашу осторожность, может удалить больше символов, чем вы ожидаете.

Например, если строка была такая:

```
x текст x....y текст y
```

то, дав команду:

```
 s/x.*y/x y/
```

вы сотрете весь текст между первым символом x и последним символом y. Чтобы избежать этого, укажите обратную черту перед метасимволом точки, чтобы отменить ее специальное значение:

```
s/x\.*y/x y/
```

Но бывают ситуации, когда шаблон .* обозначает именно то, что вам требуется. Например, чтобы изменить строку:

```
Настало время для всех нормальных пацанов ...
```

на:

```
Настало время.
```

используйте конструкцию .* для удаления всех символов после для:

```
s/для.*/./
```

Использование метасимвола * имеет свои особенности, наиболее важной из которых является то, что слова «любое количество» означают ноль или более символов. Например, если строка содержит так:

```
xy_текст_x__y_текст
```

здесь символами «_» обозначены пробелы, и вы дадите команду:

```
s/x_*y/x_y/
```

то первые символы xy подходят под указанный шаблон, так как между ними нулевое количество пробелов. В результате замена будет выполнена только над первыми символами xy. Чтобы обойти это, достаточно записать шаблон в виде:

```
/x__*y/
```

Такая запись означает: x, пробел, затем еще любое количество пробелов и символ y.

Другой нюанс использования * также связан с тем, что последовательность одинаковых символов может быть нулевой длины. Например, команда:

```
s/x*/y/g
```

выполненная над строкой:

```
abcdef
```

изменит ее на:

```
yaybycydyeyfy
```

Это происходит поскольку данная команда заменяет все последовательности x нулевой длины на y. Если вы не хотите, чтобы это происходило, укажите команду следующим образом:

```
s/xx*/y/g
```

Здесь шаблон xx* определяет один или более символ x.

**Квадратные скобки: [ и ]**

Допустим, вы хотите удалить любые числа, которые могут быть в начале всех строк в файле. Вы можете сделать это, выполнив несколько команд типа:

```
1,$s/^1*//
1,$s/^2*//
1,$s/^3*//
```

и т.д., но ясно, что это займет много времени, особенно, если числа довольно большие. Редактор *ed* позволяет выполнить задачи такого типа с помощью одной команды, использующей квадратные скобки.

Конструкция:

```
[0123456789]
```

является шаблоном для любой из указанных в скобках цифр и называется *символьным классом*. Используя символьный класс, можно очень просто выполнить нашу задачу, дав команду:

```
1,$s/^[0123456789]*//
```

Внутри символьного класса можно задавать любые символы. Только символы ^, ] и - сохраняют свое специальное значение. Чтобы найти специальные символы, вы можете, например, дать команду:

```
/[.\$^[]/
```

Перечислять в символьном классе все нужные вам цифры или символы довольно утомительно. В *ed* есть возможность использования сокращений, например: [0-9] — означает все цифры, [a-z] — все строчные буквы, [A-Z] — все прописные буквы.

Внутри квадратных скобок символ [ не является специальным. Чтобы поместить символ ] или - в символьный класс, необходимо указывать их в нем первыми.

Вы можете также задать символьный класс, определяющий все символы, за исключением указанных. Это делается заданием в качестве первого символа символьного класса знака ^.

Например, шаблон:

```
[^0-9]
```

означает все символы, за исключением цифр. Таким образом вы можете, например, найти строку, которая не начинается с пробела или символа табуляции:

```
/^[^(пробел)(символ табуляции)]/
```

Внутри символьного класса символ ^ имеет специальное значение только, если стоит первым. Проверьте, что команда:

```
/^[^^]/
```

находит строку, не начинающуюся с символа ^.

**Амперсанд: &**

Чтобы сократить время на ввод команд замены, можно использовать амперсанд для обозначения текста, указанного в левой части команды замены. Предположим, у вас есть строка:

```
Настало время
```

и вы хотите привести ее в такой вид:

```
Настало лучшее время
```

Вы можете ввести команду:

```
s/Настало/Настало лучшее/
```

Слово Настало повторять необязательно. Амперсанд & заменяет его. Амперсанд, указанный в правой части команды замены, обозначает текст, который только что был найден. Таким образом, вы можете давать команду в виде:

```
s/Настало/& лучшее/
```

В некоторых случаях использование амперсанда позволяет существенно сократить время, избежать лишнего ввода текста и опечаток. Например, чтобы заключить строку в скобки, независимо от ее длины, введите:

```
s/.*/(&)/
```

В правой части команды замены амперсанд может встречаться более одного раза. Например, команда:

```
s/Настало/& лучшее время\. & худшее время/
```

изменит текст на:

```
Настало лучшее время. Настало худшее время
```

А команда:

```
s/.*/&?&!!/
```

превратит строку в:

```
Настало время? Настало время!!
```

Чтобы отменить специальное значение амперсанда, используйте перед ним бэкслэш, например:

```
s/амперсанд/\&/
```

заменяет слово амперсанд на символ &. Амперсанд не является специальным символом, когда употребляется в левой части команды замены.

**Деление строк**

В редакторе *ed* есть возможность разбиения строки на две или более строк путем вставки новых строк. Предположим, например, что в результате редактирования строка стала слишком длинной. Если она выглядит, скажем, так:

```
. . . текст xy текст . . .
```

то вы можете разделить ее между x и y командой:

```
s/xy/x\
y/
```

Это одна команда, хоть и вводится на двух строках. Поскольку бэкслэш отменяет специальное значение символов, то знак конца строки перестает иметь специальное значение.

При желании вы можете разделить одну строку на несколько строк, используя этот же прием. В качестве примера рассмотрим случай, когда нужно выделить в тексте слово очень наклонным шрифтом, для чего приведем его в отдельной строке и предварим командой форматирования .I. Предположим, первоначальный вид строки следующий:

```
очень большой текст
```

Команда:

```
s/ очень /\
.I\
очень\
/
```

разбивает строку на четыре меньшего размера, располагает перед словом *очень* строку .I и убирает лишние пробелы.

Когда в строку подставляется новая строка, то текущей становится последняя созданная строка.

**Объединение строк**

Строки могут быть объединены с помощью команды j. Предположим, у вас есть строки:

```
Настало
 время
```

и текущей является первая из них. Тогда команда:

```
j
```

соединяет их вместе и мы получаем:

```
Настало время
```

Указанная без параметров, команда j соединяет текущую и следующую за ней строку. Но можно соединить любую последовательность строк. Для этого достаточно указать начальный и конечный номера строк, которые хотите объединить, например, команда:

```
1,$jp
```

соединяет все строки в файле в одну большую строку и выводит затем ее на экран.

**Перестановка текста внутри строки: \( и \)**

Вспомним, что & является сокращением для текста, подлежащего замене в команде замены. Примерно таким же образом можно помечать найденные части строки. Единственное отличие заключается в том, что вы должны в левой части команды указывать, какие части вас интересуют.

Предположим, у вас есть файл, строки которого состоят из фамилий и инициалов такого типа :)

```
Плотский-Поцелуев, В.И.
Медуза-Горгонер, С.
```

Это можно сделать с помощью обычных команд редактирования, но это утомительно и возможны ошибки. Альтернативный способ предлагает сначала пометить конкретные части строк (в нашем случае фамилии и инициалы), а затем переставить их. Если в левой части команды замены часть искомого текста находится между \( и \), то он запоминается и затем может быть использован в правой части команды. В правой части команды замены символы \1 относятся тексту, заключенному между первыми знаками \( и \), \2 — к следующему и т.д.

Команда:

```
1,$s/^\([.*]\),*\(.*\)/\2\1/
```

хоть и довольно неудобочитаемая, но все же выполняющая требуемые действия. Первые знаки \( и \) находят и запоминают фамилию, которая является текстом до запятой. В правой части команды замены к фамилии обращаются с помощью \1. Вторые знаки \( и \) выделяют инициалы и они обозначены в правой части как \2.

При выполнении таких сложных последовательностей редактирования рекомендуется просматривать с помощью глобальных команд g или v строки, для которых выполнялись команды замены для проверки, что все сделано именно так, как вы планировали.

## Повышение скорости редактирования

Одним из наиболее эффективных путей повышения скорости редактирования является знание строк, на которые должна действовать команда. Если вы не указываете какие строки надо изменить и какую строку сделать текущей после выполнения команды, то скорость редактирования будет довольно низкой. Если вы сможете редактировать файл не указывая лишние номера строк, то сэкономите много времени.

Например, если вы дадите команду поиска типа:

```
/штучка/
```

то окажетесь на следующей строке, содержащей слово штучка. Вам не потребуется указывать номер строки для таких команд как s, p, l, d, a или c, если вы хотите работать именно с этой строкой.

Если не найдено вхождений слова штучка, то значение текущей строки не изменяется.

Текущая строка также не изменяется, если курсор уже показывал на единственное в файле слово штучка. Эти же правила действуют и при использовании ?...?; различие заключается только в направлении поиска.

После выполнения команды удаления d текущей становится строка, следующая после удаляемой. Если же вы удаляете последнюю строку файла (т.е. с номером $), то текущей становится новая последняя строка.

Команды a, c и i, по умолчанию работают с текущей строкой. Если вы в этих командах не указываете номер строки, то команда a добавляет текст после текущей строки, команда c изменяет текущую строку и команда i вставляет текст перед текущей строкой.

Общим у этих команд является то, что после их выполнения текущей становится строка, которая вводилась последней.

Например, вы можете ввести:

```
a
. . . текст . . .
botch               # в этом месте опечатка
.
s/botch/bitch/      # исправляем опечатку
a
. . . текст . . .
.
```

не указывая никаких номеров строки ни в команде замены, ни в команде добавления. Если вы сделали много ошибок при вводе, то можете исправить их такими командами:

```
a
вводимый текст
stubid botch        # в этом месте много ошибок
.
c                   # заменяем всю строку целиком
исправленная строка
.
```

Команда r считывает текст из файла в конец буфера редактируемого вами текста, если вы не указываете номер строки, после которой надо вставить текст файла. В любом случае текущей становится последняя считанная строка. Помните, что вы можете дать команду:

```
0r
```

которая поместит содержимое файла в начале вашего текста. Вы также можете использовать 0a или 1i для добавления текста в начало буфера.

Команда w записывает весь буфер в файл. Если вы перед командой укажете один номер, то только эта строка будет записана. Два номера строк, указанные в команде, обозначают группу строк, которую требуется сохранить в файле. Команда w не изменяет значения текущей строки, независимо от количества записанных строк. Это верно и в случае когда в команде осуществляется контекстный поиск:

```
/^\.AB/,/^\.AE/w абстрактно
```

Основное правило при работе следующее: текущей всегда становится последняя измененная строка; если строки не изменялись, то и текущая строка остается прежней. Чтобы проиллюстрировать это, предположим, что в буфере имеется три строки и текущей является вторая:

```
x1
x2
```

```
x3
```

Тогда по команде:

```
-,+s/x/y/p
```

будет выведена третья строка, которая является последней строкой, в которую вносились изменения. Но если бы наши строки были другими:

```
x1
y2
y3
```

и вы дали ту же самую команду, то изменена и выведена была бы только первая строка и она же стала бы текущей.

**Точка с запятой: ;**

Поиск, задаваемый командами /.../ и ?...? начинается с текущей строки и выполняется в сторону конца или начала файла, соответственно, до тех пор, пока не будет найден искомый текст или не вернетесь опять на текущую строку. В некоторых случаях вам этого не требуется. Предположим, например, что в буфере имеются следующие строки:

```
.
.
.
ab
.
.
.
bc
.
.
.
```

Если вы находитесь на первой строке и захотите распечатать все строки между ab и bc с помощью команды:

```
/a/,/b/p
```

то у вас ничего не получится. Оба поиска (и для a и для b) выполняются одновременно, и, следовательно, оба найдут строку с ab. В результате будет напечатана только одна эта строка. Более того, если перед строкой с ab была бы строка, содержащая b, тогда при выполнении этой команды была бы ошибка, так как второй номер строки оказался бы меньше, чем первый в команде печати. Это происходит из-за того, что запятая, разделяющая номера строк, не меняет номер текущей строки после определения адреса строки в команде, каждый последующий поиск начинается с того же самого места, что и предыдущий. В редакторе *ed* знаком «точка с запятой» можно пользоваться также, как и обычной запятой. Единственная разница заключается в том, что редактор, дойдя при выполнении командной строки до точки с запятой принудительно делает текущей строку, обработанную последней. Так, в нашем примере команду следовало записать так:

```
/a/;/b/p
```

и тогда после того, как будет найдена буква a, строка с ней станет текущей и поиск буквы b начнется уже с этой строки. Это свойство очень полезно в следующих ситуациях:

Предположим, вы хотите найти в тексте место, где второй раз встречается слово штучка. Вы можете, конечно, ввести:

```
/штучка/
//
```

но при этом будет распечатана и первая строка с штучка, что вам не требовалось. Правильная команда:

```
/штучка/;//
```

Она означает: найти первое слово штучка, сделать эту строку текущей и затем найти еще одно слово штучка и распечатать только его. Чтобы найти все вхождения (кроме первого) какого-либо текста в файле, достаточно дать команду:

```
?текст?;??
```

Помните, что если вы хотите найти место, где в файле впервые встречается какое-нибудь интересующее вас слово, недостаточно ввести команду:

```
1;/штучка/
```

так как если слово штучка находится в первой строке, то оно не будет найдено. Команда:

```
0;/штучка/
```

выполняет поиск начиная с первой строки. Это один из немногих случаев, когда 0 разрешается использовать в качестве номера строки.

### Прерывание команд

Последнее замечание, касающееся установления текущей строки. Если вы нажмете клавишу Del во время выполнения редактором *ed* команды, ваш файл будет восстановлен до состояния, которое он имел до начала выполнения этой команды (в максимально возможной степени). Естественно, некоторые изменения восстановить нельзя. Например, если вы считываете или записываете файл, делаете подстановки или удаляете строки, то эти команды могут быть остановлены в непредсказуемом месте (именно поэтому не рекомендуется прерывать их). Текущая строка может остаться прежней.

Если вы используете команду печати, то текущая строка не изменяется до тех пор, пока команда не будет завершена. Таким образом, если вы решите выводить текст на экран до тех пор, пока не появится интересующая вас строка и затем нажмете на клавишу Del, чтобы прекратить вывод, то текущая строка не будет переопределена.

## Файлы

### Вставка одного файла в другой

Предположим, у вас есть файл с именем memo, и вы хотите вставить другой файл с именем table сразу после ссылки в тексте на Табл. 1. Таким образом в файле memo есть где-то строка:

```
Табл. 1 показывает ...
```

и данные, хранящиеся в файле table, должны быть помещены после этой строки. Чтобы сделать это, найдите в редактируемом файле memo текст Табл. 1 и добавьте в буфер прямо после этого файл table:

```
ed memo
/Табл\. 1/
.............. # в этой строке ответ редактора ed
.r table
```

Команда r считывает файл и здесь вы указываете ей поместить его сразу после текущей строки. Команда r, указанная без номера строки, по умолчанию добавляет считываемый файл в конец буфера, т.е. также, как команда $r.

**Запись части файла**

Другой задачей является запись в файл части редактируемого вами буфера. Например, вы хотите вывести таблицу из предыдущего примера в отдельный файл, чтобы ее можно было проверить и отформатировать отдельно. Предположим, что в редактируемом файле есть такие строки:

```
.TS
[ текст таблицы ]
.TE
```

так описывается таблица для программы `tbl` (препроцессор `troff`). Чтобы вывести таблицу в отдельный файл, необходимо найти начало таблицы (строка с .TS) и затем записать интересующую часть. Например,

```
/^\.TS/
```

На экране появится найденная строка:

```
.TS
```

Теперь введите:

```
.,/^\.TE/w table
```

и задача будет выполнена. Вы можете все это сделать сразу с помощью команды:

```
/^\.TS/;/^\.TE/w table
```

Обратите внимание на то, что здесь команда w записывает группу строк, а не весь файл. При желании вы можете записать всего одну строку; для этого достаточно указать только один номер вместо двух. Если у вас есть какая-либо сложная строка и вы знаете, что она вам впоследствии еще понадобится, то ее проще сохранить, чем вводить вновь. Например, находясь в редакторе, введите:

```
a
[ какой-нибудь  текст]
сложная строка
.
.w temp
a
[ еще какой-нибудь текст]
.
.r temp
a
[ еще какой-нибудь текст]
.
```

## Макрокоманды редактирования

Если какая-либо сложная последовательность редактирующих команд должна быть выполнена над группой файлов, то самый простой способ — это создать «список команд редактирования» (т.е. файл, содержащий все команды, которые вы хотите выполнить) и затем применять этот список к каждому файлу по очереди.

Например, предположим, вы хотите заменить все «UNIX» на «Unix» и все «США» на «Америка» в большом количестве файлов. Поместите следующие строки в файл es (от *ed script*):

```
g/UNIX/s//Unix/g
g/США/s//Америка/g
w
q
```

Теперь вы можете вводить:

```
ed -файл1< es
ed -файл2< es
...
```

Эта команда указывает редактору *ed* выполнять команды из заранее подготовленного файла с именем es. Помните, что вся работа должна быть заранее спланирована и что, используя оболочку Plan 9, вы можете организовать цикл по нужным файлам. Опция минус - подавляет сообщения от редактора.

При подготовке списка редактирующих команд вам может потребоваться поместить в строке только один единственный символ точки для обозначения конца ввода по командам a и i. Находясь в редакторе *ed* это сделать довольно трудно, так как точка, которую вы наберете, завершит ваш ввод и не будет помещена в файл. Обратная черта специального значения точки не отменяет. Единственным способом является использование символа @ для обозначения конца ввода. Затем позднее укажите для замены этого символа @ на точку команду:

```
s/^@$/./
```

## Краткое описание команд

Ниже приводится полный список команд редактора *ed*. Основной синтаксис команд редактора следующий: один или два необязательных номера строк, имя команды, и в случае команд e, f, r и w еще указывается имя файла. На строке может быть только одна команда, но команда p может указываться после любой команды (кроме e, f, r, w и q).

| Команда | Описание |
|---|---|
| a | Добавляет строки в буфер (если не указан номер строки, то вставляет их после текущей строки). Добавление прекращается после ввода на новой строке символа точки. Текущей становится последняя добавленная строка. |
| c | Заменяет указанные строки на вводимый позднее текст. Ввод новых строк прекращается после ввода точки также как в команде a. Если номер строки не указан, то заменяется текущая строка. Текущей становится последняя добавленная строка. |
| d | Удаляет указанные строки. Если не указан номер строки, то удаляется текущая строка. Текущей после этого становится следующая после удаленной строка. Если была удалена последняя строка, то текущей становится стоявшая перед ней строка. |
| e | Редактирует новый файл. Все предыдущее содержимое буфера уничтожается, поэтому сначала сохраните его командой w. |
| E | Безусловная e; смотрите ниже описание q. |
| f | Выводит на экран имя файла, которое помнит редактор. Если после f указать имя, то редактор запомнит его. |

| | |
|---|---|
| g | Команда g/регулярное выражение/команды выполняет команды над строками, в которых найдено совпадения с запросом регулярных выражений. |
| i | Вставляет строки в буфер перед указанной строкой (если номер строки не указан, то вставляет их перед текущей строкой) до тех пор пока не будет введен на новой строке символ точки . . Текущей становится последняя добавленная строка. |
| j | Выполняет объединение указанных строк. Промежуточные символы новой строки удаляются. Текущей стает результирующая строка. |
| k | Присваивает адресуемой строке метку. Текущая строка не изменяется. |
| l | Выводит на экран строки, делая видимыми непечатаемые символы и знаки табуляции. В остальном одинакова с командой p. |
| m | Перемещает указанные строки после строки с номером, указанным после символа команды m. Текущей становится последняя перемещенная строка. |
| n | Команда нумерации выдает указанные строки, предваряя каждую номером строки и символом табуляции; текущей становится последняя выданная строка. Команду n можно добавлять к любой команде, кроме e, f, r и w. |
| p | Выводит указанные строки. Если номера строк не указаны, то печатает текущую строку. Один номер строки без символа p эквивалентен команде. При простом нажатии клавиши Enter печатается следующая после текущей строка. |
| P | Синоним для p. |
| q | Выход из редактора. Ваша работа не будет сохранена, если вы не дадите команду w. Введите команду дважды, если вы в любом случае хотите покинуть редактор. |
| Q | Выход из редактора без проверки, были ли сделаны изменения в буфере с момента выполнения последней команды w. |
| r | Считывает файл в буфер (до конца, если не указано до какого места). Текущей становится последняя считанная строка. |
| s | Команда s/шаблон/замена/ заменяет последовательность символов замена при совпадении с шаблоном в указанных строках. Если номера строк не указаны, то замена производится только в текущей строке. Текущей остается строка, в которой замена была сделана последней. Если замены не было (т.е. нет совпадения с шаблоном в указанных строках), то текущей остается прежняя строка. Команда s заменяет только первое вхождение текста шаблона в строке; чтобы заменить все имеющиеся в строке вхождения, надо указывать после последней косой черты символ g. |
| t | Копирует указанные строки после строки, номер которой стоит после символа t в команде. Текущей становится последняя скопированная строка. |
| v | Команда v/регулярное выражение/список команд выполняет заданные команды только в тех строках, в которых нет последовательности символов шаблона. |
| u | Восстанавливает предыдущее содержимое первой адресуемой строки, которая должна быть последней строкой в которой была выполнена подстановка. |
| w | Записывает буфер редактирования в файл. Текущая строка не изменяется. |
| W | Эта команда аналогична описанной выше команде записи, за исключением того, что добавляет указанные строки в конец файла, если он существует. Если файл не существует, он создается, как описано выше для команды w. |

| | |
|---|---|
| = | Выводит на экран номер текущей строки. |
| !команда | Выполнение команды операционной системы Plan 9. |
| /шаблон/ | Контекстный поиск по шаблону. Текущей становится строка, в которой найден указанный шаблон. Начинается со строки .+1. |
| ?<br>шаблон? | Контекстный поиск в обратном направлении. Начинается со строки .-1. |

## Примечание

Примечание касается авторства документа. Мне, Андрею Кухару, принадлежит адаптация и компиляция существующих источников, представляющих описание редактора. За основу был взят перевод 3-й главы руководства пользователя по ОС Unix (System V/386) «Строчный текстовый редактор *ed*» Брайана Кернигана (Brian Kernighan), автор перевода неизвестен.