1

Как устроена командная строка

Прежде чем изучать команды Linux на практических примерах, вам стоит получить общее представление о том, как работает командная строка. Этому посвящена текущая глава.

Начинающим разработчикам она поможет овладеть элементарными навыками взаимодействия с командной строкой Linux. Но и более опытные специалисты узнают кое-что ценное, например, чем отличается командная оболочка от командной строки. Иногда эта разница играет важную роль!

В этой главе мы рассмотрим такие темы:

- Что такое интерфейс командной строки, или CLI.
- Как устроены команды.
- Как применяются аргументы команд и как они выглядят, когда вы вводите команды и когда обращаетесь к справочным страницам.
- Что такое командная оболочка и чем она отличается от командной строки.
- Какими основными правилами руководствуется оболочка, когда обрабатывает команды.

Для начала разберемся, что представляет собой интерфейс командной строки. Мы поговорим о том, как он работает, и рассмотрим простой практический пример.

Вначале был REPL

Что такое **интерфейс командной строки**, он же **CLI** (command-line interface)? Это интерактивная оболочка, которая работает в текстовом режиме и позволяет взаимодействовать с компьютером, для чего осуществляет такие операции:

1. Принимает то, что ввел пользователь.
2. Выполняет вычисления, обрабатывая введенные данные.
3. Выводит на экран тот или иной результат.
4. Повторяет эту последовательность операций с самого начала.

Давайте разберем каждый шаг этой процедуры на примере конкретной команды `ls`, которую мы подробно рассмотрим далее в этой главе. Пока достаточно знать, что эта команда перечисляет содержимое каталога.

	Название операции	**Буквальный перевод**	**Что делает командная оболочка**	**Описание**
1	Read	Чтение	Принимает ввод	Вы вводите команду `ls` и нажимаете Enter
2	Evaluate	Вычисление	Обрабатывает команду	Командная оболочка ищет двоичный код команды `ls`, находит его и поручает компьютеру его выполнить
3	Print	Печать	Выводит результат	Команда `ls` производит текстовые данные — имена всех файлов и каталогов, которые она нашла. Затем командная оболочка выводит эти данные в окно терминала
4	Loop	Цикл	Переходит к шагу 1 (повторяет весь процесс)	После того как программа, которую вызвала команда, завершает работу, командная оболочка снова готова принимать данные от пользователя, и последовательность операций повторяется сначала

Цикл такого рода обозначается аббревиатурой REPL — по буквам, с которых начинаются названия четырех перечисленных операций в английском языке (Read — Eval — Print Loop[1]). Эта аббревиатура пришла из языков программирования, которые ввели в обиход и отладили соответствующую процедуру, таких как Lisp.

Цикл REPL можно записать в формате, более привычном для разработчиков, — на псевдокоде:

```
пока (истина) { // бесконечный цикл
    печать(вычисление(чтение()))
}
```

[1] Цикл «*чтение — вычисление — печать*». — *Примеч. пер.*

На самом деле в большинстве языков программирования достаточно всего нескольких строчек кода, чтобы создать REPL, который сможет выполнять элементарные вычисления. Например, вот однострочная программа командной оболочки, написанная на языке Perl:

```
perl -e 'while (<>){print eval, "\n"}'
1 + 2
3
```

Здесь код на Perl передается как параметр, а результаты вычислений выводятся в консоль в ответ на каждую строку, которую вводит пользователь. В конце вычислений программа выводит новую строку и завершает работу.

Это совсем крохотная программа, но ее достаточно для того, чтобы реализовать интерактивный цикл REPL в окружении командной строки — в **командной оболочке**. Оболочки, с которыми вы будете иметь дело в Linux и Unix, устроены гораздо сложнее, чем эта миниатюрная оболочка на Perl, но они работают по тому же принципу.

Вообще говоря, если вы разрабатываете ПО, то скорее всего, уже используете REPL, даже если не подозреваете об этом, потому что этот механизм поставляется почти со всеми современными языками сценариев. По сути, функции командной строки в Linux (а также в macOS или любой другой системе семейства Unix) работают так же, как «интерактивные оболочки» в интерпретируемых языках. Так что даже если вы не знакомы с REPL языка Lisp, то вышеприведенный сценарий на Perl наверняка заставит вас вспомнить простейшие оболочки для Ruby или Python.

Теперь, когда вы понимаете основы того, как работает интерфейс командной строки, с которым вы будете иметь дело в Linux, давайте попробуем применить первые команды. Для этого нужно знать, как устроен их синтаксис и как правильно использовать его в командной строке.

Синтаксис командной строки (чтение)

Работа любого REPL начинается с того, что он считывает данные, которые ввел пользователь. Чтобы оболочка принимала команды из командной строки Linux, они должны быть синтаксически правильными. Общая форма команды такова:

```
имя_команды параметры
```

С точки зрения программирования имя команды можно понимать как имя функции, а параметры — это набор из произвольного количества аргументов, которые ей передаются. Обратите внимание, что не существует универсального синтаксиса

са для любых параметров: каждая команда по-своему задает множество параметров, которые она готова принимать. Поэтому оболочка в общем случае не умеет проверять, правильно ли указаны параметры команды; она может только убедиться, что тому или иному имени команды соответствует исполняемый код.

ПРИМЕЧАНИЕ

В этой главе термины «программа» (program) и «команда» (command) употребляются как синонимы. Разница между ними незначительна: некоторые встроенные функции командной оболочки определены в ее коде и поэтому формально не являются отдельными программами. Но вам незачем вдаваться в эти подробности; оставьте такие нюансы ветеранам Unix.

Давайте посмотрим на более развернутый пример синтаксиса параметров команды, с которым вы будете часто иметь дело:

```
команда [-флаги,] [--example=foobar] [прочие параметры ...]
```

Этот общепринятый формат будет встречаться вам в документации, например, на справочных страницах (man-страницах), которые включены в большинство сред Linux, и этот синтаксис довольно прост:

- `команда` — это программа, которую вы запускаете;
- параметры в квадратных скобках не обязательны, а скобки с многоточием (например, `[xyz ...]`) обозначают, что в этом месте можно передать ноль или более аргументов;
- `-флаги` — это любые допустимые параметры программы (в Unix они называются флагами), например `-debug` или `-foobar`[1].

Некоторые программы также принимают короткие и длинные формы одних и тех же параметров: короткая форма начинается с одного дефиса, а длинная — с двух. (Например, параметры `-l` и `--long` могут обозначать одно и то же.) Так ведут себя не все команды, а только те, создатели которых позаботились о том, чтобы одни и те же параметры можно было задавать и в короткой, и в длинной форме.

Не все команды позволяют при вызове передавать настройки по такой схеме, но она охватывает самые распространенные случаи, которые вам встретятся.

---

[1] `foobar`, а также `foo`, `bar`, `baz` и некоторые другие похожие английские слова широко используются в компьютерной документации как универсальные имена-заместители. Они будут часто встречаться далее в книге. — *Примеч. науч. ред.*

По умолчанию пробел означает конец аргумента. Поэтому, как и в большинстве языков программирования, если в качестве аргумента передается строка, которая содержит пробелы, то ее нужно заключать в одинарные или двойные кавычки. Подробнее об этом вы прочтете в главе 12 «Как автоматизировать задачи с помощью сценариев командной оболочки».

Совсем скоро мы подробно проследим, как оболочка интерпретирует команду, которую вы вводите с помощью этого синтаксиса. Но сначала давайте четко обозначим разницу между двумя терминами, которые мы используем в этой главе и которые иногда употребляются как взаимозаменяемые: «командная строка» и «командная оболочка».

Командная строка и командная оболочка

Под **командной строкой** (command line) или **окружением командной строки** (command-line environment) в этой книге мы понимаем любую среду, которая работает в текстовом режиме и ведет себя как цикл REPL, позволяя взаимодействовать с операционной системой, интерпретатором языка программирования, базой данных и т. д. Интерфейсом командной строки называется общий принцип коммуникации с системой.

Но мы будем употреблять и более узкий термин — «командная оболочка».

Командная оболочка, **командный интерпретатор** или просто **оболочка** (shell) — это конкретная программа, которая реализует интерфейс командной строки и позволяет вводить команды в текстовом режиме. Существует много разных оболочек, которые обеспечивают похожее окружение командной строки в режиме REPL, но могут выполнять совершенно разные задачи, например:

- Bash[1] — это широко распространенная оболочка для взаимодействия с Linux и другими операционными системами семейства Unix.
- Все популярные системы управления базами данных, такие как PostgreSQL, MySQL и Redis, оснащены командной оболочкой для разработчиков, с помощью которой можно взаимодействовать с базами данных, запуская команды.
- Большинство интерпретируемых языков программирования поставляются с командными оболочками, которые позволяют ускорить разработку. В таких

[1] В названиях популярных оболочек (Bash, Zsh, tsch, fish и др.) буквы *sh* — сокр. *shell* (*командная оболочка*). Bash расшифровывается как *Bourne Again Shell* — это игра слов, которая подчеркивает как преемственность с классической оболочкой Bourne Shell, так и идею перерождения (*bourne again* созвучно с *born again* — *рожденный заново*). В свою очередь, Bourne Shell (также известная как просто sh) была названа по имени своего создателя Стивена Борна (Stephen Bourne). — *Примеч. пер.*

оболочках допустимые команды — это просто инструкции соответствующего языка. Возможно, вы встречали оболочку `irb` для Ruby, интерактивную оболочку для Python или другие подобные инструменты.

- Zsh — это альтернативная командная оболочка для операционных систем, похожая на Bash. Ее можно встретить на компьютерах тех разработчиков, кто специально настроил свою рабочую среду[1].

В этой книге под *оболочкой* мы будем понимать командную оболочку Unix (обычно это Bash) с интерфейсом командной строки, которая позволяет взаимодействовать с Linux или другой нижележащей операционной системой семейства Unix.

Каким образом оболочка узнаёт, какую программу запускать (вычисление)

После того, как командная оболочка *прочитала* команду, она должна *вычислить* ее, то есть выполнить ту или иную программу, предоставить определенную информацию или сделать еще что-то полезное.

ПРИМЕЧАНИЕ

Последующее чрезвычайно подробное описание того, как работает оболочка, может поначалу показаться занудным. Но мы обещаем, что если вы освоите эти подробности, то вам будет гораздо легче устранять неполадки в ситуации, когда нужной программы нет или когда для нее заданы неправильные права доступа.

Когда вы набираете в командной оболочке (например, Bash) команду вроде `foobar -option1 test.txt` и нажимаете Enter, происходит вот что:

1. Если для команды задан путь к файлу, оболочка обращается по этому пути. Путь может принимать разные формы:
   - Абсолютный путь — например, `/usr/bin/foobar` в команде `/usr/bin/foobar -option1 test.txt`.
   - Относительный путь — например, текущий рабочий каталог в команде `./foobar -option1 test.txt`. (Точка в начале команды обозначает текущий каталог; мы подробнее поговорим об этом позже в разделе «Абсолютные и относительные пути к файлам». По сути, эта команда означает «выполнить файл `foobar`, который находится в текущем каталоге».)

[1] Zsh является системной оболочкой по умолчанию в macOS, начиная с версии macOS Catalina (2019), а также в некоторых современных дистрибутивах Linux (например, Kali Linux). — *Примеч. науч. ред.*

- Путь может содержать особые конструкции двух типов:
  - переменные окружения — например, `$HOME` в пути `$HOME/foobar`;
  - специальные символы, которые поддерживает оболочка, — например, тильду (`~`) в пути `~/foobar` (символ `~` обозначает домашний каталог текущего пользователя).

2. Если путь к файлу не задан, оболочка проверяет, нет ли у нее информации о том, что такое `foobar`. Например, это может быть:
   - встроенная команда оболочки;
   - псевдоним выполнения, с помощью которого можно настраивать макросы или сокращенные имена команд.
3. Если ничего из этого не подошло, то оболочка обычно обращается к переменной окружения `$PATH`, которая перечисляет несколько каталогов, где следует искать команды: `/bin`, `/usr/bin`, `/sbin` и т. д. Этот перечень могут изменять как пользователи, так и различные программные продукты: диспетчеры версий для языков сценариев, виртуальные окружения для Python и многие другие программы. Оболочка просматривает каталоги в том порядке, в каком они следуют в переменной `$PATH`, пока не найдет в одном из них исполняемый файл с именем `foobar`.

Если после всего этого оболочка так ничего и не нашла, она возвращает ошибку вроде `-:foobar: command not found` [1].

С другой стороны, если на каком-нибудь этапе этой процедуры оболочка находит исполняемый файл с именем `foobar`, она запускает этот файл и передает ему в качестве аргументов `-option1` и `test.txt` (именно в таком порядке).

В этом случае оболочка знает, какая программа нужна, и запускает именно ее. По мере того как команда вычисляется, ее результаты выводятся на экран, и таким образом завершается третья стадия цикла REPL (печать). После этого командной оболочке остается только вернуться в начало цикла и повторить всю процедуру заново, приняв следующую команду от пользователя.

Процесс, который мы рассмотрели, помогает командной оболочке наилучшим образом угадать, какую программу пользователь хочет запустить, и избежать неоднозначности. Неоднозначность может вызывать проблемы и приводить к недоразумениям и ошибкам. Устраняя неполадки, часто требуется выяснить, какая команда на самом деле запущена. Для этого служит команда `which` *`имя_команды`*, которая выводит полный путь к исполняемому файлу команды и позволяет узнать, является ли она встроенной командой оболочки. В некоторых системах нет команды `which`, зато можно использовать более универсальную `command -v` — это экви-

[1] Команда foobar не найдена. — *Примеч. пер.*

валентная команда в соответствии со стандартом POSIX, о котором мы скоро поговорим[1]:

```
bash-3.2$ which ls
/bin/ls
bash-3.2$ command -v ls
/bin/ls
```

Краткое введение в POSIX

«Википедия» гласит, что **POSIX**[2] — это «семейство стандартов, утвержденное Компьютерным обществом IEEE для того, чтобы обеспечить совместимость между операционными системами». На практике это попытка определить некоторые общие стандарты, которым должны следовать системы семейства Unix, чтобы не получалось так, что в разных системах приняты совершенно разные наборы элементарных команд.

По сути, POSIX содержит требования такого рода: «В каждой системе, совместимой с POSIX, должна быть команда, которая перечисляет содержимое каталога и называется `ls`». В случае с `command -v` требование звучит так: «Каждая система, совместимая с POSIX, должна обеспечивать способ проверить, существует ли исполняемый файл, который соответствует заданному имени команды».

Если вы хотите, чтобы ваши сценарии были переносимы между разными операционными системами семейства Unix, имеет смысл ограничиться только командами, которые соответствуют POSIX. Но это не стопроцентная гарантия: многие широко распространенные дистрибутивы Linux отклоняются от POSIX в различных аспектах, причем некоторые отклонения трудно обнаружить, пока они не причинят вам неприятности.

---

[1] В этом и последующих листингах `bash-3.2$` и подобные конструкции — это *приглашение командной строки* (*prompt*). Когда оболочка готова принимать команды, она обычно отображает приглашение, а после него — мигающий курсор, который обозначает позицию ввода. Приглашение может принимать разные формы в зависимости от операционной системы, командной оболочки и пользовательских настроек: например, оно может содержать название и версию оболочки (как в этом примере), имя пользователя и компьютера, текущий каталог, специальные символы вроде → и т. д. Иногда приглашение может оформляться иначе, чем вводимые команды, например выделяться другим цветом или курсивом. Символ `$` в конце приглашения означает, что командная оболочка запущена с правами обычного пользователя (не администратора).

В листингах в этой книге фигурирует командная оболочка Bash 3.2, хотя на момент выхода англоязычного издания, а также на момент подготовки этого перевода наиболее свежей версией является Bash 5.2, выпущенная в 2022 году. — *Примеч. науч. ред.*

[2] Portable Operating System Interface (переносимый интерфейс операционной системы). — *Примеч. пер.*

POSIX — это последняя тема, которую нам нужно было затронуть, прежде чем начать на практике работать с командной строкой. К этому моменту мы с вами уже рассмотрели немало основных понятий:

- Вы узнали о том, как устроен цикл REPL («чтение — выполнение — печать») и как он воплощается в современных командных оболочках.
- Вы познакомились с общим синтаксисом команд, который вам предстоит использовать, работая в Linux.
- Вы разобрались, каким образом командная оболочка определяет, как принять вашу команду и правильно «вычислить» ее.
- Вы изучили важные понятия, которые будут часто встречаться на протяжении книги: «командная оболочка», «интерфейс командной строки», POSIX, а также еще несколько тем, владение которыми облегчит вашу профессиональную деятельность.

После того как вы освоили эти основы, можно перейти от теории к практике. В следующем разделе мы поговорим о специфическом для Linux контексте, в котором вы будете находиться, когда запускаете команды. Вы получите необходимые начальные сведения о файловой системе Linux и о том, как обращаться с разными типами путей к файлам и каталогам. После этого вся оставшаяся часть главы будет посвящена тому, как запускать различные команды Linux.

Элементарные навыки работы в командной строке

Чтобы эффективно работать с Linux, следует овладеть элементарными знаниями: как устроена файловая система, как просматривать ее структуру и перемещаться по ней, как просматривать и редактировать файлы. Все это мы рассмотрим в данном разделе, после чего вы будете уверенно ориентироваться в структуре файлов и каталогов в Linux.

Далее в книге мы подробнее углубимся в каждую тему и каждую команду, но сейчас важно, чтобы к концу главы вы получили минимальный набор практических навыков, которые можно применять в профессиональной деятельности.

Введение в файловую систему Unix

В графических пользовательских интерфейсах **каталоги** (в macOS они называются **папками**) представляются в виде значков. Возможно, вы привыкли наблюдать в своем домашнем каталоге ряды этих пиктограмм: «Рабочий стол», «Документы», «Видео» и т. д. Если дважды щелкнуть по значку каталога, откроется новое окно, в котором отображается содержимое этого каталога.

Когда мы говорим «файловая система», то имеем в виду именно это — совокупность каталогов и файлов, в которых содержатся все данные, существующие в системе. Интерфейс командной строки опирается на то же самое понятие файловой системы, просто она представлена немного по-другому.

В командной строке вы не увидите многочисленных окон и значков: все представляется в виде текста, а содержимое каталогов отображается только тогда, когда вы его запросите. Однако файлы и каталоги по-прежнему ведут себя привычным образом.

Поначалу может показаться сложным держать файловую систему в голове, когда вы работаете с файлами и каталогами, но когда вы к этому привыкнете, то убедитесь, что такой подход часто оказывается более эффективным, чем щелкать по значкам. Большинство пользователей, которые поработали в таком режиме хотя бы несколько дней, без труда удерживают в памяти подробное представление файловой системы, и им редко требуется его уточнять.

Абсолютные и относительные пути к файлам

Новичков в Linux часто сбивает с толку разница между **абсолютными** и **относительными** путями к файлам и каталогам. Если ее не уяснить, можно впустую потратить много времени, разглядывая сообщения об ошибках вроде такого:

```
No such file or directory[1]
```

Чтобы правильно запустить почти любую команду Linux, нужно понимать, как устроены пути к файлам, поэтому мы прежде всего остановимся на этом вопросе.

Абсолютный путь — это полный путь от корневого каталога до того или иного файла в файловой системе. Этот путь начинается с прямого слеша (`/`), обозначающего **корневой каталог** — каталог верхнего уровня, который лежит в основе файловой системы и содержит все остальные файлы и каталоги.

Вот несколько примеров абсолютных путей:

- `/home/dave/Desktop`
- `/var/lib/floobkit/`
- `/usr/bin/sudo`

Эти **абсолютные пути** подобны полному маршруту в навигаторе, который ведет вас по всем дорожным развязкам от известной отправной точки, например от вашего дома или, в случае системы семейства Unix, от корневого каталога.

---

[1] Такого файла или каталога не существует. — *Примеч. пер.*

Абсолютный путь можно однозначно узнать по тому, что он начинается с прямого слеша (/). Абсолютный путь работает одинаково независимо от того, в каком месте файловой системы вы находитесь, потому что он представляет собой полный и уникальный адрес файлового объекта.

Относительный путь — это частичный путь, и предполагается, что он отталкивается не от корневого, а от *текущего каталога*. Относительный путь отличается тем, что он *не* начинается со слеша.

Относительный путь — это как маршрут от того места, где вы сейчас находитесь. Если вы съехали на обочину, потому что заблудились и хотите уточнить дорогу, то вас интересует маршрут от вашего *текущего местоположения*, а не от дома. Это и есть относительный путь.

В результате относительные пути часто удобнее вводить, чем абсолютные. Например, если вы уже находитесь в каталоге `/home/Desktop`, то к файлу `mydocument.txt` проще обратиться по этому имени, чем по полному пути `/home/Desktop/mydocument.txt` (хотя если вы находитесь в указанном каталоге, то оба пути одинаково действительны). Разница между этими путями проявится, когда вы перейдете в другой каталог. Если переместиться на один уровень вверх — из каталога `/home/Desktop` в каталог `/home`, — то абсолютный путь будет попрежнему вести к тому же самому файлу, а относительный — нет. (В нашем случае, если теперь ввести `mydocument.txt`, это будет обозначать файл `/home/mydocument.txt`.)

Для примера рассмотрим фрагмент файловой системы. Предположим, что так выглядит структура каталогов и файлов внутри каталога `/home/dave/Desktop`:

```
Desktop                  # Рабочий стол[1]
├── anotherfile          # ├── другой файл
├── documents            # ├── документы
│   └── contract.txt     # │   └── договор
├── somefile.txt         # ├── какой-то файл
└── stuff                # └── разное
    ├── nothing          #     ├── неважно
    └── important        #     └── важно
```

Допустим, вы находитесь в каталоге `Desktop` («Рабочий стол»); иными словами, ваш текущий каталог — `/home/dave/Desktop` (этот путь можно просмотреть командой `pwd`).

Вот несколько примеров относительных и абсолютных путей к одним и тем же файлам в этом каталоге:

[1] Приблизительные эквивалентные имена файлов и каталогов на русском языке приведены для тех читателей, которым это поможет лучше понять пример. — *Примеч. пер.*

Относительный путь	Абсолютный путь
`anotherfile`	`/home/dave/Desktop/anotherfile`
`documents/contract.txt`	`/home/dave/Desktop/documents/contract.txt`
`stuff/important`	`/home/dave/Desktop/stuff/important`

Обратите внимание, что относительный путь — это то же самое, что абсолютный путь, из начала которого отсечена часть, ведущая к текущему рабочему каталогу.

Еще об абсолютных и относительных путях

Продолжая рассматривать наш фрагмент файловой системы, представьте, что вы работаете в командной оболочке и ваш текущий каталог — `Desktop`. Допустим, вам нужно вывести сведения о файле `contract.txt`, который находится в папке `documents`. Как обратиться к этому файлу? Есть два способа:

- `ls /home/dave/Desktop/documents/contract.txt` — абсолютный путь, который одинаково работает из любого каталога.
- `ls documents/contract.txt` — относительный путь, то есть путь относительно текущего каталога.

Как открыть командную оболочку

Чтобы в Ubuntu Linux или macOS открыть командную оболочку с интерфейсом командной строки, запустите приложение «Терминал» (Terminal).

Сведения о файлах и каталогах

Если вы еще плохо знакомы с командной оболочкой, то первое, что стоит сделать, когда вы ее открыли, — осмотреться в системе. В этом разделе мы перечислим важнейшие команды Linux, нужные для того, чтобы ориентироваться в файлах и каталогах и выводить сведения о них.

pwd — текущий каталог

Когда вы запускаете команду `pwd`[1] в терминале, оболочка выводит абсолютный путь к каталогу, в котором вы находитесь. Файловую систему Unix часто ассоциируют с деревом, но пока можно рассматривать ее как захламленный рабочий стол со множеством каталогов. Если представлять себе каждый каталог как комнату, то команда `pwd` позволяет узнать, в какой комнате сейчас находится ваше окружение командной строки[2].

---

[1] Аббр. *print working directory* (*вывести на печать рабочий каталог*). — *Примеч. пер.*

[2] Имена команд в оболочках Linux и других систем семейства Unix зависят от регистра. То есть чтобы получить имя текущего каталога, нужно ввести именно `pwd`, а не `Pwd` или `PWD`. — *Примеч. науч. ред.*

Новые сеансы командной оболочки обычно начинаются в вашем домашнем каталоге. Если вы запускаете команды в Linux, то увидите нечто подобное:

```
→ ~ pwd
/home/dave
```

В других системах семейства Unix вывод может выглядеть немного иначе. Например, вот что отображается в macOS:

```
→ ~ pwd
/Users/dave
```

Независимо от того, в каком месте файловой системы вы находитесь, можно обращаться к любому файлу в любом каталоге (см. раздел «Абсолютные и относительные пути к файлам» ранее в этой главе). Однако иногда удобнее перемещаться между каталогами. Далее в главе 5 мы подробнее рассмотрим, как устроена файловая система.

ls — содержимое каталога

Команда `ls`[1] выводит список файлов в заданном каталоге. Если запустить ее без аргументов, она просто перечисляет файлы и каталоги, которые находятся в текущем каталоге. Если в качестве аргумента передать путь к конкретному каталогу, то `ls` выведет его содержимое:

```
ls /var/log
```

Команда `ls` может принимать и другие аргументы — так называемые флаги. Их бывает много, но два самых распространенных — это `-l`[2] (подробные сведения) и `-h`[3] (размер файлов в «человекочитаемом» формате).

```
ls -l -h

#  Тот же результат; флаги можно объединять
ls -lh

# Выводит содержимое указанного каталога
ls -lh /usr/local/
```

Команда `ls` с флагом `-l` выводит результат в таком формате:

```
-rw-r--r-- 1 dcohen wheel 0 Jul   5 09:27 foobar.txt
```

---

[1] Сокр. *list* (*перечислить, вывести список*). — *Примеч. пер.*

[2] Название флага `-l` происходит от long (длинный формат вывода). В отличие от большинства флагов, у него нет длинной формы (наподобие `--long` или чего-то похожего). — *Примеч. науч. ред.*

[3] Короткому флагу `-h` соответствует эквивалентная длинная форма `--human-readable` (человекочитаемый формат). — *Примеч. науч. ред.*