1. Про Unix и файловые системы

Операционная система (ОС) — это основное программное обеспечение, которое управляет компьютером. Она управляет аппаратными ресурсами (процессор, память, устройства ввода-вывода) и предоставляет платформу для выполнения программ. ОС работает как посредник между пользователем и оборудованием. Unix — это многозадачная, многопользовательская операционная система, разработанная в 1969 году в компании Bell Labs (AT&T). Изначально она была создана для работы на серверах и рабочих станциях, но также используется в научных и образовательных целях. Unix повлиял на многие современные операционные системы, включая Linux, macOS и другие.
Файловая система в UNIX — это способ организации данных на диске и управления файлами и директориями.
1. Что такое файл?
Файл — это минимальная единица данных в файловой системе. Это может быть текстовый документ, программа, изображение или даже каталог (директория). Все в UNIX (файлы, каталоги, устройства) рассматривается как файл.
При вызове функции ls –l первый символ указывает на тип файла
- = обычный файл (сюда относятся все файлы с данными, играющими роль ценной информации сами по себе);
d = директория (это файлы, в качестве данных которых выступают списки других файлов и каталогов);
b = файл блочного устройства (файлы устройств предназначены для обращения к аппаратному обеспечению компьютера (дискам, принтерам, терминалам и др.));
c = файл символьного устройства (аналогично b);
s = доменное гнездо (socket, используется для организации процессов в ОС);
p = именованный канал (pipe, аналогично s);
l = символическая ссылка.
Блок в файловой системе — это минимальная единица хранения данных на диске, которая используется операционной системой для управления файлами. Файлы хранятся в блоках, и каждый файл занимает целое число блоков, даже если он меньше одного блока. Размер блока зависит от конкретной файловой системы, но часто составляет 512 байт или 4096 байт.
2. Что такое файловая система?
Файловая система — это структура, которая определяет, как данные хранятся и организуются на диске. В UNIX файловая система организована как древовидная структура:
В самом верху — корневая директория /.
Под ней находятся другие директории и файлы.
Пример:

```
/ (root directory)
|-- home/
|   |-- user1/
|   |   |-- file1.txt
|   |   |-- file2.txt
|-- etc/
|   |-- config.conf
|-- var/
    |-- log/
```

Здесь /home/user1/file1.txt — это путь к файлу file1.txt.
Unix позволяет монтировать файловые системы из разных устройств в дерево. Это означает, что различные устройства (жёсткие диски, USB-накопители и т.д.) могут быть доступны через структуру дерева, позволяя пользователю обращаться к ним как к обычным директориями.
3. Абсолютные и относительные пути
Абсолютный путь — это путь от корня файловой системы (от /). Он всегда начинается с /. Пример: /home/user1/file1.txt.
Относительный путь — это путь от текущей директории, в которой вы находитесь. Например, если вы находитесь в директории /home/user1/, то относительный путь к файлу file1.txt будет просто file1.txt.
4. Что такое . и ..?
Когда создается директория, в ней автоматически появляются два специальных файла:
. — это ссылка на саму себя (текущую директорию).
.. — это ссылка на родительскую директорию (директорию, на уровень выше).
Например, если вы находитесь в директории /home/user1/, то:
. — это путь к самой директории /home/user1/,
.. — это путь к родительской директории /home/. Эти файлы нужны для удобной навигации в файловой системе.
5. Жесткие ссылки и символические ссылки
В UNIX файловой системе можно создать несколько ссылок на один и тот же файл:
Жесткая ссылка — это другая ссылка на файл, которая хранится в той же файловой системе. Когда вы создаете жесткую ссылку, создается еще одно имя для файла, но файл сам не дублируется. Удалив одно имя (ссылку), файл все равно останется доступен через другую ссылку, пока не удалите все ссылки. Жесткие ссылки нельзя создавать на директории, чтобы не нарушить структуру дерева.
Символическая ссылка (или "симлинк") — это файл, который указывает на другой файл или директорию. Это как ярлык в Windows. Симлинк может указывать на файлы и директории, находящиеся даже в других файловых систе-

мах. Если удалите исходный файл, симлинк станет "битым" т. е. будет указывать на несуществующий файл.
Пример:
Файл /home/user1/document.txt
Создаем жесткую ссылку: ln /home/user1/document.txt
/home/user2/doc_copy.txt
Создаем символическую ссылку: ln -s /home/user1/document.txt
/home/user2/doc_symlink.txt
6. Shell (оболочка)
Shell — это программа, которая позволяет пользователю взаимодействовать с операционной системой через командную строку.
Имя файла — это просто метка, которая используется для обращения к файлу через файловую систему. Полный путь к файлу включает путь к его местоположению в файловой системе.
Пример пути к файлу: /home/user/documents/file.txt, где:
/home — это директория (каталог),
/home/user — подкаталог внутри /home,
/home/user/documents/ — подкаталог в директории user,
file.txt — это имя самого файла.
Что такое номер файла (inode)?
Номер файла (inode) — это уникальный идентификатор каждого файла в файловой системе Unix/Linux. В файловой системе каждый файл хранится не только по имени, но и имеет связанный с ним inode. В inode хранится важная информация о файле: размер файла, время последнего изменения, права доступа, количество жестких ссылок, физическое расположение данных на диске.
Важно: В inode не хранится имя файла. Имя файла хранится в директории и связано с соответствующим inode. Пример: Если два файла с разными именами имеют одинаковый inode, это значит, что это один и тот же файл с двумя именами.
Почему у файла может быть множество имен? У файла в Unix/Linux может быть множество имен благодаря тому, что существует понятие жесткой ссылки (это дополнительное имя для одного и того же файла). Так как несколько имен могут указывать на один и тот же файл через один и тот же inode, файл будет иметь несколько "имен" или "ссылок но физически это один и тот же файл. Пример:
Файл file1.txt имеет inode 12345. Мы создаем жесткую ссылку ln file1.txt file2.txt. Теперь у нас есть два файла file1.txt и file2.txt, но оба ссылаются на один и тот же файл через один и тот же inode 12345. Удаление одного из файлов не удаляет сам файл, так как пока остается хотя бы одна жесткая ссылка, файл продолжает существовать.

Жесткие ссылки, имена файлов и inode в Unix/Linux взаимодействуют следующим образом:
Как хранится информация о файле в директории?
Представь директорию как специальный файл (да, директория — это тоже файл), который содержит таблицу с записями. В этой таблице каждая запись соответствует файлу или поддиректории, находящейся в этой директории.
Эта запись не содержит сами данные файла (например, текст или изображение), а только имя файла и номер inode, который является уникальным идентификатором для этого файла. То есть, в директории не хранятся файлы целиком — там только ссылки (в виде имен файлов и связанных inode).
Пример структуры директории
Предположим, у нас есть директория /home/user, в которой лежит файл file1.txt. Когда ты создаешь файл, система создает запись в этой директории, которая связывает имя файла с его inode. Пример:
В директории /home/user может храниться запись:
file1.txt -> inode 12345
Это означает, что файл file1.txt имеет номер inode 12345. Вся подробная информация о файле (где он хранится на диске, размер файла, права доступа) находится в inode 12345, но имя файла хранится только в директории.
Таким образом, в директории на каждое имя файла есть запись, которая связывает это имя с inode.
Жесткие ссылки и несколько имен для одного файла
Теперь представь, что ты создаешь жесткую ссылку на файл. Жесткая ссылка — это еще одно имя для уже существующего файла. При этом создается новая запись в директории, которая указывает на тот же самый inode, что и исходный файл. Пример:
Ты создаешь файл file1.txt, который имеет inode 12345:
file1.txt -> inode 12345 Затем ты создаешь жесткую ссылку на этот файл, например:
ln file1.txt file2.txt Теперь в директории /home/user будет две записи:
file1.txt -> inode 12345
file2.txt -> inode 12345
Обе записи указывают на один и тот же inode (12345). Это значит, что у файла есть два имени — file1.txt и file2.txt, но это один и тот же файл на уровне файловой системы. Они ссылаются на одни и те же данные, расположенные в inode 12345.
Что происходит при удалении имени файла?
Если ты удаляешь одно из имен файла (например, file1.txt), система удаляет запись об этом имени в директории, но сам файл не удаляется, если на inode всё еще есть другие ссылки. В данном примере:
Если ты удаляешь file1.txt, остаётся запись file2.txt, которая по-прежнему ссылается на inode 12345. Файл не исчезнет с диска, так как inode 12345 ещё связан с file2.txt. Файл удаляется полностью только тогда, когда все ссылки (имена),

которые ссылаются на inode, удалены. Когда удаляется последнее имя файла, счетчик ссылок inode становится равным нулю, и система удаляет файл с диска. Важный момент: в директории хранятся только ссылки на файлы. Теперь важно понять, что в самой директории не хранится никакая информация о данных файла. В директории есть только записи, которые связывают имя файла с inode. В inode содержатся все данные о файле, но не в самой директории. Когда ты работаешь с файлами через командную строку или файловый менеджер, система находит файл по имени в директории, затем по номеру inode находит данные файла и показывает их тебе.

2. Ещё про inode, симлинк и хардлинк

inode - это объект файловой системы, содержащий информацию о владельце/группе, которым принадлежит файл или каталог, его права доступа к нему, его размер, тип файла, timestamp-ы отражающие время модификации индексного дескриптора (ctime, changing time), время модификации содержимого файла (mtime, modification time) и время последнего доступа к файлу (atime, access time) и счётчик для учёта количества жёстких ссылок на файл. Также содержит указатели на блоки данных, где физически хранятся данные файла на диске (но не сам путь). Каждый inode имеет собственный номер, который присваевается ему файловой системой в момент её создания (форматирования).
По сути, жесткая ссылка - это тот же самый файл, на который идёт такая ссылка. Права доступа и inode не отличаются от исходного файла. Изначально файл представляет собой запись в таблице файловой системы, которая хранит информацию о том, где находятся данные файла на диске. Хардлинк создаёт ещё одну запись, которая указывает на те же данные. При создании хардлинка не создаются новые данные, то есть данные файла не дублируются. Хардлинк указывает на те же физические данные, что и исходный файл. Если вы измените файл через хардлинк, изменения будут видны и через другие ссылки на этот файл (включая оригинальную запись), так как все они ссылаются на одни и те же данные. Хардлинки можно создавать только в пределах одной файловой системы (одного раздела диска), поскольку они напрямую указывают на блоки данных. Хардлинки нельзя создавать для директории, иначе это могло бы привести к образованию циклов в файловой системе. Например, если бы директория A имела хардлинк на директорию B, и в свою очередь директория B имела хардлинк на директорию A, это создало бы бесконечный цикл. В результате система не смогла бы корректно обрабатывать навигацию и доступ к данным. .. и . не являются хардлинком в традиционном смысле, так как его нельзя создать вручную. Это автоматическая ссылка, созданная файловой системой, чтобы обеспечить возможность навигации по иерархии директорий.
Симлинк создаётся с помощью той же команды ln но с ключём -s: ln -s file1 symlink1. Мы создаём новый (!) объект файловой системы с именем symlink1, который указывает на уже существующий файл file1. Права доступа и inode отличаются от исходного файла. По сути симлинк - просто относительный путь до файла. Он не содержит данные самого файла, а лишь указывает на другой файл или каталог. Это как ярлык или указатель на файл, который можно использовать для доступа к оригинальному файлу по его другому пути. При доступе к симлинку операционная система автоматически перенаправляет запрос на целевой файл или каталог, на который ссылается симлинк. Изменения, сделанные через симлинк, происходят в целевом файле. То есть если вы откроете симлинк и измените данные файла, изменения будут применены к оригинальному файлу, так как симлинк просто перенаправляет на него. Если удалить симлинк, сам целевой файл не будет затронут. Но если удалить оригинальный файл, симлинк станет "битой" ссылкой, указывающей на несуществующий объект. При попытке открыть битый симлинк система вернёт ошибку. Симлинки могут указывать не только на файлы, но и на каталоги. Это полезно, когда нужно создать быстрый доступ к какому-то часто используемому каталогу в другой части файловой системы. Симлинки могут указывать на файлы или каталоги на других разделах диска или даже на удалённые файловые системы. Это отличие от хардлинков, которые должны находиться в пределах одной файловой системы.

3. Права

Задание прав пользователям в Unix и других операционных системах имеет несколько важных целей. Основной причиной является обеспечение безопасности и управление доступом к данным и ресурсам. В системе прав Unix для файлов и директорий используется довольно простая, но мощная модель, основанная на трех уровнях прав доступа для трех категорий пользователей. Unix разделяет всех пользователей на три категории:
Владелец (user, u) — обычно это создатель файла или директории.
Группа (group, g) — группа пользователей, которым может быть предоставлен доступ к файлу или директории. Любой пользователь находится в группе "по умолчанию".
Прочие (others, o) — все остальные пользователи, которые не являются ни владельцем, ни частью группы.
Права доступа в Unix можно увидеть с помощью команды ls -l, которая выводит права файла в следующем формате:
-rwxr-xr–
Первый символ указывает на тип файла: - — обычный файл, d — директория, l — символическая ссылка и так далее.
Следующие девять символов делятся на три блока по три символа:
Первый блок: права для владельца файла (user).
Второй блок: права для группы.
Третий блок: права для всех остальных (others).
Каждый блок содержит три символа:
r — право на чтение файла.

w — право на запись в файл.
x — право на исполнение файла.
Права доступа к директориям (r, w, x) в Unix имеют немного другое значение, чем для обычных файлов. Важное отличие заключается в том, что директории — это контейнеры для других файлов и поддиректорий, поэтому права на директорию определяют, что пользователь может делать с её содержимым (просматривать, изменять или перемещаться по ней).
Право на чтение директории позволяет пользователю просматривать список файлов и поддиректорий, находящихся внутри этой директории. Если у пользователя есть право на чтение (r) директории, он сможет выполнить команду ls, чтобы увидеть, какие файлы находятся в этой директории. Но при этом, если у пользователя есть право на исполнение (x), он всё равно сможет работать с файлами внутри директории, зная их точные имена (например, открыть файл напрямую с помощью cat или less).
Право на запись (w) в директорию даёт возможность добавлять, удалять и переименовывать файлы в этой директории. Однако для успешного выполнения этих операций, обычно требуется также наличие права на исполнение (x) (это сделано для безопасности: пользователь может иметь право изменять содержимое директории, но система должна быть уверена, что он также имеет доступ к этому содержимому). Если у пользователя есть право на запись (w) в директорию, он сможет создавать новые файлы или поддиректории, удалять или переименовывать существующие файлы (если пользователь знает точные имена файлов внутри директории).
Право на исполнение (x) директории даёт возможность "входить" в директорию, т.е. открывать её и перемещаться по её содержимому. Оно не связано с запуском программ, как для файлов, но означает, что пользователь может получить доступ к самой директории и выполнять действия над её содержимым, зная его точные имена. Если у пользователя есть право на исполнение (x) директории, он сможет перейти в неё с помощью команды cd или получить доступ к файлам внутри директории (например, открыть файл с cat или nano). Если у пользователя нет права на исполнение, то, даже если у него есть право на чтение, он не сможет перемещаться по директории или обращаться к её содержимому напрямую.
Если право отсутствует, на его месте будет стоять -. Например, запись rwxr-xr– расшифровывается так:
Владелец файла (user) имеет полные права: чтение, запись и исполнение. Группа пользователей (group) имеет права на чтение и исполнение, но не на запись. Прочие пользователи (others) имеют только право на чтение.
кроме символа - (отсутствие прав) в выводе прав доступа к файлам и директориям в Unix и Linux можно встретить также символы + и .
Эти символы добавлены для более детального отображения прав доступа, связанных с расширенными атрибутами (ACL) и контекстом безопасности. Символ + в выводе команды ls -l указывает на то, что для файла или директории заданы расширенные списки контроля доступа (ACL, Access Control List). ACL позволяют задавать более детальные права на уровне пользователя и группы, которые выходят за рамки стандартной системы прав. Здесь символ + в конце строки прав (-rw-r–r–+ 1 user group 4096 Oct 2 14:23 myfile.txt) показывает, что для файла myfile.txt определён ACL, который может предоставлять или ограничивать доступ конкретным пользователям или группам, помимо стандартных прав (например, можно указать дополнительные права для конкретных пользователей, кроме владельца и основной группы). Символ . указывает на то, что у файла или директории нет расширенных прав доступа (ACL), но может быть назначен контекст безопасности (это дополнительные метаданные, которые используются для определения правил доступа к файлам, процессам и другим ресурсам в системе на основе политик безопасности. Они особенно важны в системах с усиленными мерами контроля доступа), связанный с системами, такими как SELinux (Security-Enhanced Linux).

chmod — это команда в Linux и других Unix-подобных операционных системах, которая позволяет изменять права доступа к файлам или каталогам.
опция -R у команды chmod в Unix используется для рекурсивного изменения прав доступа к файлам и директориям.
Текстовый метод
Для изменения прав доступа — или режима доступа — к файлу используйте команду chmod в терминале.
chmod кто=разрешения имя_файла
Где кто — любая из нескольких букв, каждая из которых обозначает, кому дано разрешение. Они следующие:
u (user): пользователь, который является владельцем файла.
g (group): группа пользователей, которой принадлежит этот файл.
o (other): другие пользователи, то есть все остальные.
a (all): все сразу; используйте вместо ugo.
До: drwxr-xr-x 6 archie web 4096 июл 5 17:37 Документы
chmod g= Документы
chmod o= Документы
После: drwx—— 6 archie web 4096 июл 6 17:32 Документы
Здесь, поскольку вы хотите отказать в разрешениях, вы не ставите никаких букв после =, где будут введены разрешения. Из этого видно, что только разрешения владельца — это rwx, а все остальные разрешения — это -.
Сокращения для текстового метода
Команда chmod позволяет добавлять и вычитать разрешения из существующих, используя + или - вместо =. Это отличается от описанных выше команд, которые по сути полностью заменяют разрешения (например, чтобы изменить разрешения с r– на rw-, вам всё равно нужно указать r и w после = в вызове команды chmod. Если вы пропустите r, то = перезапишет разрешения и таким образом удалит разрешение r. Использование + и - позволяет избежать этого, добавляя или отнимая разрешения из текущего набора разрешений).

Пример, который запрещает запись абсолютно всем:
До: -rw-rw-r– 1 archie web 5120 июн 27 08:28 foobar
chmod a-w foobar
После: -r–r–r– 1 archie web 5120 июн 27 08:28 foobar
Копирование разрешений
С помощью chmod можно взять разрешения у одного класса, например владельца, и выставить те же разрешения группе или даже всем. Для этого вместо r, w или x после = поставьте нужную вам букву кто. Например:
До: -rw-r–r– 1 archie web 5120 июн 27 08:28 foobar
chmod g=u foobar
После: -rw-rw-r– 1 archie web 5120 июн 27 08:28 foobar
Эта команда по сути означает «изменить разрешения группы (g=), чтобы они были такими же, как у владельца (=u)». Обратите внимание, что вы не можете скопировать одновременно несколько разрешений или добавить новые, то есть такая команда chmod g=wu foobar выдаст ошибку.
Числовой метод
Использование чисел — это ещё один метод, который позволяет редактировать разрешения одновременно для владельца, группы и остальных. Основная структура такова:
chmod xxx file
Где xxx это три цифры, каждая из которых может иметь значение от 0 до 7. Первая цифра задаёт разрешения для владельца, вторая — для группы, а третья — для всех остальных.
Права r, w и x соответствуют следующим числам:
r=4
w=2
x=1
Чтобы объединить нужные права в одно трёхзначное число, нужно суммировать соответствующие значения. Например, если вы хотите предоставить владельцу каталога права на чтение, запись и выполнение, а группе и всем остальным — только права на чтение и выполнение, то числовые значения будут выглядеть следующим образом:
Владелец: rwx = $4 + 2 + 1 = 7$
Группа: r-x = $4 + 0 + 1 = 5$
Остальные: r-x = $4 + 0 + 1 = 5$
chmod 755 file
Это эквивалентно следующим двум командам:
chmod u=rwx file
chmod go=rx file
Также двоичный метод, при котором каждое разрешение рассматривается как двоичное число, а затем они объединяются в обычное десятичное число.

4. Копирование файлов

Команда cp предназначена для копирования файлов и директорий. Чтобы скопировать часть дерева (директорию и её содержимое), можно использовать опцию -R (рекурсивное копирование). Она позволяет копировать директории вместе с вложенными файлами и поддиректориями. Если файл, в который происходит копирование, существует, то его содержимое заменяется без предупреждения. При копировании создаётся полная копия файла с отдельным набором данных и собственным inode. Эти файлы хранятся отдельно друг от друга и могут быть изменены независимо. Копия файла и оригинальный файл — это два независимых файла. Изменение одного из них не влияет на другой. Они занимают в файловой системе разное место на диске.
Основные опции:
-R: рекурсивное копирование. Используется для копирования директорий и всех файлов внутри неё.
-p: сохраняет атрибуты файлов, такие как права доступа, метки времени, владельца и группу.
-v: выводит на экран информацию о каждом скопированном файле.
-i: запрашивает подтверждение перед перезаписью файла.
-n - для копирования файлов без перезаписи существующих файлов в целевой директории.
Допустим, есть директория /home/user/data, и вы хотите скопировать её в /backup/:
cp p /home/user/data /backup/
Опция -p гарантирует, что, если некоторые из промежуточных директорий (например, /backup) ещё не существуют, они будут созданы автоматически.
Если нужно скопировать только определённые файлы, например, все файлы с расширением .txt, используйте маски файлов:
cp /home/user/data/*.txt /backup/data/

5. Удаление файлов

Команда rm предназначена для удаления файлов и директорий.
rm [опции] файл_или_директория
rm file.txt
Это удалит файл file.txt в текущей директории без вывода
rm file1.txt file2.txt file3.txt
Удаляет все указанные файлы.
По умолчанию, rm не спрашивает подтверждения на удаление. Чтобы включить запрос на подтверждение, используйте опцию -i: rm -i file.txt
Выводит запрос: "remove regular file 'file.txt'? и пользователь может подтвердить удаление.
Использование опции -f (force) позволяет принудительно удалить файл, игнорируя ошибки и не запрашивая подтверждения. Это удобно, если нужно удалить файл, защищённый от записи (read-only), или если вы не хотите подтверждать удаление каждого файла.
Удаление директорий: чтобы удалить пустую директорию, команда rm не используется напрямую. Для этого обычно применяют rmdir. Чтобы удалить непустую директорию с файлами внутри, используется опция -r (рекурсивное удаление): rm -r folder_name
Эта команда удаляет указанную директорию и все её содержимое.
Принудительное рекурсивное удаление директорий: опция -rf сочетает рекурсивное удаление и принудительное игнорирование ошибок.
Если файл начинается с символа -, например, -file.txt, команда может интерпретировать его как опцию. Чтобы избежать этого, укажите путь к файлу через ./: rm ./-file.txt
Для удаления группы файлов по маске: rm *.txt - удаляет все файлы с расширением .txt в текущей директории.
В современных Unix/Linux системах существует защита от случайного удаления корневой директории. Если пользователь попытается выполнить команду вроде: rm -rf /, то система может выдать ошибку или предупреждение. Для обхода этого ограничения может использоваться флаг –no-preserve-root, но использовать это крайне опасно.
rm не гарантирует полное удаление данных с диска. После использования rm, данные могут быть восстановлены с помощью специальных утилит. Для надежного удаления данных можно использовать команду shred, которая несколько раз перезаписывает данные перед их удалением, чтобы предотвратить восстановление данных с диска: shred -u file.txt
shred перезаписывает файл случайными данными и затем удаляет его.
Команда rmdir используется для удаления пустых директорий. Она работает только с пустыми директориями и не поддерживает рекурсивное удаление. Использование rmdir гарантирует, что вы удаляете только те папки, которые действительно пусты, минимизируя риск утраты данных.
rmdir folder_name
Если директория folder_name пустая, она будет удалена. Если директория не пустая, команда выдаст ошибку. Можно указать несколько директорий: rmdir dir1 dir2 dir3
Если есть несколько вложенных пустых директорий, можно использовать флаг -p для удаления всей цепочки пустых директорий: rmdir -p parent_dir/child_dir. Если parent_dir и child_dir обе пустые, они будут удалены.

6. Команды

Команда ls выводит список файлов и каталогов в заданной директории. По умолчанию она отображает содержимое текущего рабочего каталога.
ls [опции] [файлы или каталоги]
-l: Долгий формат (выводит подробную информацию: права доступа, размер, дата изменения и тд).
-a: Показывает все файлы, включая скрытые (начинающиеся с .).
-p - добавляет символ / в конец имени каждого каталога в выводе. Это позволяет отличать каталоги от обычных файлов.
-t: Сортировать по времени последнего изменения.
-r: Выводит файлы в обратном порядке (по умолчанию выводится по возрастанию имени).
-h: используется с -l для вывода размеров в файлах в разных единицах (по умолчанию - байты).
-R: Рекурсивный вывод содержимого всех подкаталогов.
-S: Сортирует файлы по размеру, начиная с самых больших.
-1: вывод файлов в столбик.
-i: Выводит inode-номера файлов.
-d: Выводит информацию только о каталоге, а не о его содержимом. Полезно, если нужно узнать о каталоге как о файле, а не его содержимом.
Если нужно вывести полный путь ко всем файлам, находящимся в каталоге, можно использовать:
ls -d $PWD/*
Когда вы используете команду ls -l, вы можете заметить строку, начинающуюся со слова total. Эта строка отображает общее количество блоков, занимаемых файлами и каталогами в текущем каталоге.
Если пользователь состоит в нескольких группах, то выводится его основная группа.

Команда cd (change directory) используется для изменения текущего рабочего каталога. Эта команда позволяет вам перемещаться по файловой системе.
cd [каталог]
Переход в домашний каталог: cd ~ (SHELL заменяет тильду на абсолютный путь к домашней директории)
Переход в родительский каталог: cd ..
Переход в указанный каталог: cd /path/to/directory
Переход в последний посещённый каталог: cd -

Команда mkdir (от английского make directory) используется для создания новых каталогов (директорий) в файловой системе.
mkdir [опции] <имя_каталога>
Если имя каталога содержит пробелы, его нужно заключить в кавычки.
-p: создать родительские каталоги, если они не существуют. Это полезно для создания вложенной структуры каталогов.
mkdir -p /path/to/directory
-m: Установить права доступа при создании каталога. Например, можно указать, что каталог должен быть доступен только для чтения и записи для владельца, и только для чтения для группы и остальных.
mkdir -m 755 my_directory

Команда echo используется для вывода текста или переменных в стандартный вывод (обычно в терминал).
echo [опции] [строка]
echo "Hello, World!"
Этот пример выведет на экран строку Hello, World!
echo -e "Line 1\nLine 2"
Используя опцию -e, можно включить интерпретацию специальных символов. В этом случае \n будет интерпретироваться как новая строка.
\n — Новая строка.
\t — Горизонтальная табуляция.
\v — Вертикальная табуляция.
\b — Удаляет предыдущий символ.
\a — Звуковой сигнал (если поддерживается терминалом).
\r — Возврат каретки (переход в начало строки).
\\— Выводит обратный слеш.
echo "Hello, World!" > output.txt
Эта команда создаст файл output.txt (или перезапишет его, если он уже существует) и запишет в него строку echo -e "12345\rabc"
abc45
В этом примере курсор вернулся в начало строки, и символы abc заменили первые три символа строки.
Опция -n отключает добавление символа новой строки в конце вывода, что может быть полезно, если вы хотите, чтобы следующий вывод был на той же строке.
echo - выводит просто -.

Команда wc (сокращение от word count) используется для подсчета строк, слов и символов в файлах или вводе данных. Она предоставляет краткую информацию о содержимом файлов или потока данных.
wc [опции] [файл(ы)]
Если файлы не указаны, команда работает с вводом через стандартный поток (stdin).
Ключевые опции:
-l — подсчитать количество строк.
-w — подсчитать количество слов.
-c — подсчитать количество байтов.
-m — подсчитать количество символов (в том числе и многобайтовых, в отличие от опции -c).
-L — вывести длину самой длинной строки в файле.
wc file.txt выводит что-то типо
10 (строк) 50 (слов) 300 (байт) file.txt
Команде wc также можно передавать на вход несколько файлов (тогда появится строка total - общее количество строк).
Строки: wc определяет строку по символу новой строки (\n). Пустые строки также учитываются.
Слова: wc считает любое количество символов, разделенных пробелами, табуляцией или новой строкой, как одно слово.
Байты: это общее количество байт в файле. В однобайтовых кодировках, таких как ASCII, количество байтов равно количеству символов.
Символы: учитывает символы в файле, включая многобайтовые (например, UTF-8).

Команда sort используется для сортировки строк в текстовых файлах или на входе через стандартный поток. Она может сортировать данные по алфавиту (по возрастанию), числовому значению, в обратном порядке и с использованием различных критериев.
sort [опции] [файл(ы)]
Если файл не указан, команда sort читает данные из стандартного ввода.
Основные опции:
-r — сортировка в обратном порядке (реверс).
-n — сортировка по числовым значениям.
-k — сортировка по определённой колонке (ключу). Например, -k 2 сортирует по второй колонке.
-t — задание разделителя для колонок (по умолчанию пробелы или табуляции).
-u — удалить дубликаты строк (оставляет только уникальные строки).
-b — игнорировать начальные пробелы при сортировке.
-d — сортирует только по символам букв и цифр, игнорируя все остальные.
-g — сортирует по "общему числовому значению поддерживает числа с плавающей точкой, экспоненты и т.д.
-f — игнорировать регистр при сортировке (сделать сортировку нечувствительной к регистру).
-M — сортировка по месяцам (например, Jan, Feb, и т.д.).
sort -k 2,2n -k 1,1 data.txt
-k2n означает: "Начать сортировку с второго столбца как с числового". Однако без указания конечной колонки сортировка может продолжиться на следующих столбцах, если значения во втором столбце одинаковы.
-k 2,2n — это более точная команда, которая указывает "Сортировать только по второму столбцу как по числовому значению". Это ограничивает сортировку строго вторым столбцом, не затрагивая последующие столбцы.
Файл data.csv содержит строки, разделённые запятыми:
John,25
Alice,30
Bob,25
Charlie,30
David,22
Команда для сортировки по второму полю (возраст) с указанием, что разделитель — запятая: sort -t "," -k 2,2n data.csv
Результат:
David,22
John,25
Bob,25
Alice,30
Charlie,30

Команда grep используется для поиска строк в файлах или потоках ввода, которые соответствуют заданному регулярному выражению или строке.
grep [опции] Паттерн [Файл(ы)]
Основные опции команды grep:
grep "error" log.txt
Ищет слово "error" в файле log.txt и выводит все строки, где оно встречается.
-i — Игнорировать регистр.
-v — Выводить строки, не соответствующие шаблону (инвертированный поиск).
-r или -R — Рекурсивный поиск в подкаталогах.
-l — Вывести только имена файлов, в которых найдены совпадения.
-c — Подсчитать количество строк, содержащих совпадения.
-n — Выводить совпадающие строки с номерами строк.
-w — Ищет совпадения только с цельными словами (не частью слова).
grep -w "hello" файл.txt
Найдёт строку с hello, но не строку с hello123.
-A N — Вывести N строк после совпадения.
-B N — Вывести N строк перед совпадением.
-C N — Вывести N строк до и после совпадения.
Выведет совпадающую строку, две строки до и две строки после неё.
-e — Задание нескольких шаблонов для поиска.
grep -e "hello" -e "world" файл.txt
Ищет строки, содержащие либо "hello либо "world".
-E - использование расширенных регулярок.
. — любой одиночный символ.
* — ноль или более вхождений предыдущего символа.
^ — начало строки.
$ — конец строки.
[] — диапазон символов. Например, [a-z] — любой символ от a до z.

Команда pwd выводит абсолютную ссылку текущей директории.

Команда cat (сокращение от concatenate) используется для чтения и объединения файлов, а также для их вывода в стандартный вывод (обычно на экран).
cat [опции] [файл1 файл2 ...]
Просмотр содержимого файла:
cat file.txt
Эта команда выводит содержимое файла file.txt в терминал.
Объединение нескольких файлов:
cat file1.txt file2.txt > combined.txt
Команда объединяет содержимое файлов file1.txt и file2.txt и записывает результат в файл combined.txt. Если файл combined.txt уже существует, он будет перезаписан.
Добавление содержимого в конец файла:
cat file1.txt >> file2.txt
Эта команда добавляет содержимое файла file1.txt в конец файла file2.txt без его перезаписи.
Создание нового файла через cat:
cat > newfile.txt
После ввода команды, всё, что вы напишете в терминале, будет записано в файл newfile.txt. Для завершения ввода нажмите Ctrl+D (символ конца файла).
Полезные опции
-n: нумерация всех строк.
-b: нумерация только непустых строк.
-s: удаление последовательных пустых строк (сжатие их до одной).
-e: показывает все символы (включая символ конца строки - $).
С помощью команды tac (обратная версия cat) можно выводить содержимое файла в обратном порядке.

Команда tail используется для вывода последних строк файла или потока данных.
tail [опции] [файл]
Если файл не указан, команда читает данные из стандартного ввода (например, другой команды через пайп).
-n [N]: выводит последние N строк файла. Если N не указано, по умолчанию выводятся 10 строк.
tail -n 5 filename.txt - Выведет последние 5 строк файла filename.txt.
tail -n +N - выводит все строки, начиная с N-ой строки
Аналогично **команда head** выводит первые строки файла.

Команда touch позволяет создавать пустые файлы, а также изменять временные метки уже существующих файлов.
touch [опции] <имя_файла>
Если указанный файл не существует, команда touch создаст пустой файл с таким именем. Если файл уже существует, touch обновит время последней модификации и время последнего доступа.
В Unix-подобных системах у каждого файла есть три временные метки:
Время последнего доступа (atime) — когда файл последний раз открывался (выводит ls -lu).
Время последней модификации (mtime) — когда файл последний раз изменялся (выводит ls -l).
Время последнего изменения статуса (ctime) — когда изменялись метаданные файла (например, права доступа) (выводит ls -lc).
Команда touch обычно обновляет atime и mtime, но не затрагивает ctime.
-a: обновляет только время последнего доступа (atime), не меняя времени последней модификации.
-m: обновляет только время последней модификации (mtime), не изменяя времени последнего доступа.
-d <дата>: позволяет установить временную метку, используя удобный формат, например "YYYY-MM-DD hh:mm:ss" или естественный язык, если система поддерживает.
-c: не создает файл, если он не существует. Эта опция просто игнорирует отсутствие файла, но все еще пытается обновить временные метки, если файл существует.
-r <другой_файл>: копирует временные метки из указанного файла, обновляя целевой файл до тех же меток.

Команда mv в Unix и Unix-подобных системах используется для перемещения или переименования файлов и каталогов. Это одна из базовых команд для работы с файлами в командной строке. Перемещение файла или каталога означает, что объект переносится из одного места в другое. Переименование — это изменение имени файла или каталога без изменения его расположения.
mv [опции] источник (что переносим) назначение (куда)
источник — файл или каталог, который нужно переместить или переименовать.
назначение — новое место для файла или его новое имя.
-i (interactive): запрашивает подтверждение перед перезаписью существующих файлов.
-f (force): принудительно перезаписывает файлы, если они уже существуют, не запрашивая подтверждения.
-n (no-clobber): запрещает перезапись существующих файлов. Если файл с таким именем уже существует в целевом каталоге, он не будет перезаписан.
-v (verbose): показывает подробную информацию о перемещаемых файлах и каталогах.
cp копирование работает медленнее, чем mv, потому что пробегается по всем копируемым файлам и записывают в

новые inode, а mv просто переписывает пути. Команда mv, в отличие от cp, не создает новые копии файлов — она просто перемещает или переименовывает их. Директория в этом случае просто перемещается вместе со всем содержимым, независимо от уровня вложенности. Поскольку команда mv не создает новых объектов, ей не нужно указывать -r, в отличие от cp.

Команда more используется для постраничного просмотра содержимого текстовых файлов. Она особенно полезна для работы с большими файлами, так как позволяет пользователю просматривать файл по частям, по одной странице за раз, а не весь текст сразу, как это делает, например, команда cat. При просмотре файла с помощью more доступны следующие команды для навигации:
Пробел (space) — переходит к следующей странице.
Enter — прокручивает вывод на одну строку вниз.
f — переходит на следующую страницу.
q — выходит из режима просмотра.
more +20 largefile.txt - просмотр файла с 20-й строки
more -s largefile.txt - просмотр с подавлением повторяющихся пустых строк
В отличие от более продвинутой команды less, more поддерживает ограниченные возможности навигации, такие как возврат к предыдущим страницам не всегда работает. Поиск в more поддерживает только движение вперед по тексту и базовые шаблоны. more не позволяет динамически изменять файл в процессе просмотра.

Команда less является более мощной и гибкой альтернативой команде more. В отличие от more, команда less загружает только те части файла, которые нужны для отображения, что ускоряет работу с большими файлами, less позволяет перемещаться по файлу в обе стороны, поддерживает как прямой, так и обратный поиск по тексту, используя регулярные выражения и встроенные команды.
-N — отображение номеров строк.
-S — отключение автоматического переноса строк.
Команда less поддерживает широкий набор команд для перемещения по файлу:
Space - переход на следующую страницу.
b — переход на предыдущую страницу.
G — переход в конец файла.
g — переход в начало файла.
Стрелки вверх/вниз — перемещение по одной строке.
PageUp/PageDown — переход на страницу вверх или вниз.
q — выход из less.

Команда ln создаёт ссылки на файлы. Она позволяет создавать как жёсткие ссылки, так и символические ссылки.
ln [опции] источник (файл) ссылка (имя ссылки)
Чтобы создать жёсткую ссылку, достаточно указать команду без дополнительных опций. Для создания символической ссылки используется опция -s.
-d: разрешить создавать жесткие ссылки для директорий суперпользователю.
-s: создать символическую ссылку вместо жёсткой.
-f (force): заменить существующую ссылку или файл с тем же именем. Если вы пытаетесь создать ссылку с именем, которое уже существует (будь то файл или ссылка), команда ln без флага -f выдаст ошибку и не заменит существующий файл. Но когда вы указываете -f, команда ln автоматически удаляет файл или ссылку с совпадающим именем и создаёт новую ссылку под этим именем, перезаписывая старую.
-i (interactive): Перед заменой существующих файлов попросить подтверждение у пользователя.
-v (verbose): Отображать информацию о создаваемых ссылках.
ln без опции -s создает симлинк на указанный файл, если в качестве файла указана директория.

Команда sed (stream editor) — утилита для обработки текстовых данных. Она применяется для выполнения поиска, замены, фильтрации и трансформации текста. sed обрабатывает текст в так называемом потоковом режиме: он принимает текст на входе (поток ввода), обрабатывает его построчно в реальном времени и выдаёт результат (поток вывода). Это отличает sed от обычных текстовых редакторов, где текст нужно сначала загрузить в интерфейс, отредактировать и сохранить.
sed [опции] 'команда' [файл]
s - поиск и замена текста (например, sed 's/old/new/')
Заменить слово "cat" на "dog" только во 2-й строке файла - sed '2s/cat/dog/' file.txt
Команда g в sed используется для глобальной замены всех вхождений шаблона в строке. Если g не указана, sed по умолчанию меняет только первое вхождение шаблона в строке.
p - вывод строк, которые соответствуют заданным условиям.
Печать только строк, содержащих определённое слово - sed -n '/error/p' example.txt
-n - отключает автоматический вывод. /error/ ищет строки, содержащие "error" а p выводит только эти строки.
Печать нескольких строк по диапазону - sed -n '3,5p' example.txt
d - удаление строк, совпадающих с шаблоном (например, sed '/pattern/d')
Удалить все строки, содержащие слово "error" - sed '/error/d' file.txt
Удалить строки с 10-й по 20-ю - sed '10,20d' file.txt

a - вставка текста после определённой строки (например, sed '3a New Line')
Добавить строку "End" после каждой строки, содержащей слово "Section" - sed '/Section/a End' file.txt
Добавить строку "New Line" после третьей строки - sed '3a New Line' file.txt
\U - преобразование в верхний регистр в команде замены (например, sed 's/.*/\U&/')
\L - преобразование в нижний регистр в команде замены (например, sed 's/HELLO/\L&/')
-e - позволяет выполнить несколько команд sed за один вызов. Это удобно, когда нужно сделать несколько преобразований в одном вызове sed. Заменить "error" на "alert" и "warning" на "caution":
sed -e 's/error/alert/g' -e 's/warning/caution/g' example.txt
-i - позволяет вносить изменения прямо в файл без вывода на экран и промежуточных файлов. Это полезно для сохранения изменений сразу в исходном файле.
Не создавая резервной копии (указали ''), меняем первое вхождение "abc" на "123" и сохраняем изменения в файле - sed -i '' 's/abc/123/g' example
sed -i '' 's/\r//g' script.sh - редактирует файл, заменяя все символы (g) "возврата каретки" на ничего (//)

globstar — это опция Bash, которая расширяет возможности шаблонов (globbing), обычно используемых для поиска файлов и директорий по шаблону.
shopt — команда для управления настройками и опциями оболочки в bash.
-s - включить расширение.
-u - вырубить.
без опций - посмотреть включенные расширения.

7. Потоки

Потоками называются файлы, с которыми можно обмениваться информацией.
Стандартный ввод (stdin, 0)
Это поток, из которого программа получает данные. По умолчанию stdin — это клавиатура.
Стандартный вывод (stdout, 1)
Это поток, куда программа выводит данные. По умолчанию stdout выводит данные в терминал (на экран).
Стандартный поток ошибок (stderr, 2)
Этот поток используется для вывода сообщений об ошибках. Обычно он выводится в терминал, как и стандартный вывод. В отличие от stdout, поток stderr по умолчанию не смешивается с основным выводом программы и не перенаправляется вместе с ним. Это позволяет отдельно обрабатывать ошибки и логи. Например, если перенаправить стандартный вывод в файл, ошибки по-прежнему будут отображаться в консоли.
Потоки можно подключать к чему угодно: к файлам, программам и даже устройствам. В командном интерпретаторе bash такая операция называется перенаправлением:
< file - Использовать файл как источник данных для стандартного потока ввода.
> file - Направить стандартный поток вывода в файл. Если файл не существует, он будет создан, если существует — перезаписан сверху.
2> file - Направить стандартный поток ошибок в файл. Если файл не существует, он будет создан, если существует — перезаписан сверху.
>> file - Направить стандартный поток вывода в файл. Если файл не существует, он будет создан, если существует — данные будут дописаны к нему в конец.
2>> file - Направить стандартный поток ошибок в файл. Если файл не существует, он будет создан, если существует — данные будут дописаны к нему в конец.
&>file или >&file - Направить стандартный поток вывода и стандартный поток ошибок в файл. Другая форма записи: >file 2>&1.
Конвейер — это механизм, позволяющий передавать вывод одной команды на вход другой. С помощью конвейера можно комбинировать несколько команд, чтобы выполнять обработку данных поэтапно. Конвейер создается с помощью символа | и работает, используя потоки ввода и вывода:
Стандартный вывод (stdout) — данные, которые команда отправляет в выходной поток.
Стандартный ввод (stdin) — данные, которые команда получает на вход.
Когда команды соединяются в конвейере, stdout первой команды становится stdin для следующей. Это позволяет выполнять сложные задачи в несколько шагов, не создавая промежуточных файлов.

8. Дополнительно

Кавычки
Одинарные кавычки ' '
Всё внутри одинарных кавычек (' ') интерпретируется дословно, без каких-либо замен или интерпретаций. Переменные, такие как $VAR, и спецсимволы (\, *, $, и т. д.) не обрабатываются.
Двойные кавычки " "
Всё, что заключено в двойные кавычки (" "), также интерпретируется почти дословно, но переменные ($VAR), ко-

манды (через ‘command‘ или $(command)) и escape-последовательности (\n, \t) обрабатываются. Двойные кавычки позволяют вставлять значения переменных в текст.
Обратные апострофы ‘ ‘
Используются для выполнения команд в оболочке и подставляет результат выполнения команды на это место в строке. Хотя обратные кавычки работают, они считаются устаревшими, и вместо них рекомендуется использовать $() для подстановки команд.

Оболочки
sh, bash и ksh — это разные типы командных интерпретаторов (оболочек - shell), которые предоставляют интерфейс для взаимодействия с операционной системой через командную строку. Каждый из них имеет свои особенности, синтаксис, функции и возможности.
sh - это одна из первых оболочек, созданных в Unix в конце 1970-х. sh имеет простой синтаксис, базовый набор команд и операций управления потоком (циклы, условия и т.д.). Она довольно ограничена по функционалу по сравнению с современными оболочками, но все еще используется для написания скриптов из-за высокой совместимости с другими Unix-оболочками.
bash — это расширенная версия sh. Она является стандартной оболочкой в большинстве современных дистрибутивов Linux. bash добавляет множество новых возможностей, таких как автодополнение команд, история команд, поддержка работы с массивами и функций, расширенные конструкции для работы с циклами и условиями. bash обеспечивает обратную совместимость с sh, так что большинство скриптов, написанных для sh, работают в bash без изменений.
ksh (KornShell) направлен на работу с бОльшим количеством пользователей.

Скрипт
Чтобы запустить скрипт, находящийся в текущей директории, используйте команды:
./script.sh или sh script.sh
Первый вариант требует read и execute, второй - только read.
Запуск, как в первом варианте, заставляет shell запускать скрипт, будто это обычный исполняемый файл (как будто мы ручками вводим в терминал то, что написано в скрипте). Тк shell ищет бинарники всех команд в $PATH по умолчанию, то ./ мы указываем, что надо искать не в $PATH, а в текущей директории. Находится "команда" script.sh и выполняется запуск скрипта.
Второй вариант запускает команду sh и передаёт ей в аргумент имя скрипта, так что запустится новый процесс.

Группы
Группы были разработаны для того, чтобы расширить возможности управления правами. Все группы, созданные в системе, находятся в файле /etc/group. Посмотрев содержимое этого файла, вы можете узнать список групп linux, которые уже есть в вашей системе. Пользователь имеет основную группу, она указывается при создании, а также несколько дополнительных. Основная группа пользователя — это группа, которая связана с ним по умолчанию. Группа по умолчанию автоматически назначается каждому новому пользователю при его создании и становится основной группой, к которой принадлежат все файлы, созданные этим пользователем (если явно не указано другое). Каждый файл или директория, созданные пользователем, будут принадлежать ему и его основной группе. Когда пользователь создаёт файл, по умолчанию его владельцем становится сам пользователь, а группа, владеющая файлом, — это основная группа пользователя. Дополнительные группы нужны, чтобы мы могли разрешить пользователям доступ к разным ресурсам добавив его в эти группы в linux.
Управление группами Linux для пользователя выполняется с помощью команды usermod. Рассмотрим ее синтаксис и опции: $ usermod опции имя_пользователя
-G — дополнительные группы, в которые нужно добавить пользователя
-g изменить основную группу для пользователя
-R удалить пользователя из группы.
usermod -g developers alice
Это изменит основную группу для пользователя alice на developers. При вызове функции ls –l указывается основная группа пользователя + в конце списка прав указывает на наличие установленных права ACL (access control list - списки контроля доступа) - списки контроля доступа.

Переменные
Переменные используются для хранения данных (значений), которые могут быть использованы в командной строке или в скриптах. Переменные можно создавать прямо в командной строке или в скриптах. Чтобы создать переменную, следуйте следующему синтаксису:
имя_переменной=значение
Между именем переменной, знаком = и значением не должно быть пробелов.
Значения переменных могут быть строками, числами или результатами команд.
Знак $ в Unix используется для:
Обращения к значению переменной.
myvar="Linux"
echo $myvar
Вывод будет: Linux
Локальные переменные: это переменные, которые доступны только в текущей оболочке (сессии). Они создаются про-

стым присвоением. После завершения сессии или выполнения скрипта эти переменные исчезнут.
Переменные окружения: эти переменные доступны не только в текущей оболочке, но и для всех процессов, которые запускаются этой оболочкой. Чтобы сделать переменную глобальной (переменной окружения), используется команда export.
Специальные переменные: в Unix есть ряд специальных переменных, которые используются для разных системных задач. Например:
$HOME: домашний каталог пользователя.
$USER: имя текущего пользователя.
$PATH: список каталогов, где система ищет исполняемые файлы
$PWD: текущий рабочий каталог.
$SHELL содержит имя текущей командной оболочки, например, bash.
Также можно использовать результаты выполнения команд внутри других команд или присваивать их переменным. Это делается с помощью конструкции подстановки команд (command substitution), где результат команды вставляется на место, где она была вызвана. Существует два способа подстановки команд:
Через обратные кавычки (backticks): ‘команда‘
Или через конструкцию $() (рекомендуемый способ): $(команда)
current_date=$(date)
echo $current_date
Получите что-то вроде: Sat Oct 19 12:34:56 UTC 2024
echo "Текущий каталог: $(pwd)"
Вывод может быть таким: текущий каталог: /home/username
mv $(ls | grep "myfile.txt") /backup/
В этом примере команда ls | grep myfile.txt ищет файл с именем myfile.txt и передаёт его в качестве аргумента команде mv, которая переместит этот файл в каталог /backup/.

Фильтры
Это команды (или программы), которые воспринимают входной поток данных, производят над ним некоторые преобразования и выдают результат на стандартный вывод (откуда его можно перенаправить куда-то еще по желанию пользователя). Фильтр-команды: cat, grep, sort, tail, head, wc, sed.

Блоки
Блок — это минимальная единица памяти, используемая файловой системой или операционной системой для управления памятью. Разделение памяти на блоки позволяет легче управлять небольшими фиксированными блоками, чем произвольными кусками данных различного размера, помогает контролировать и минимизировать фрагментацию памяти, позволяет оптимизировать доступ к памяти и дисковым ресурсам, так как операции с фиксированными размером блоками могут быть выполнены быстрее и с меньшими накладными расходами.
Файловая система делит диск на блоки. Размер блока определяет минимальный размер данных, которые могут быть прочитаны или записаны на диск. Обычно это 512 байт или 4 КБ. Файловая система управляет файлами, распределяя их по блокам. Файлы могут занимать один или несколько блоков. Файловая система использует структуры данных, такие как таблицы размещения файлов (FAT), индексы (i-nodes) или B-деревья, для отслеживания, какие блоки принадлежат каким файлам.

Директория -d
mkdir – -d (игнорим все флаги) или mkdir ./-d (указываем директорию текущую)

cp src dest и cp src dest/
cp src dest
Если dest не существует, команда создаст файл dest, который будет копией файла src.
Если dest существует и это файл, он будет перезаписан содержимым src.
Если dest существует и это директория, src будет скопирован внутрь этой директории с сохранением имени: dest/src.
cp src dest/
Здесь dest/ трактуется именно как директория.
Если dest не существует, команда выдаст ошибку: путь с / в конце должен указывать на директорию, а ее не существует.
Если dest существует и это директория, команда скопирует src внутрь dest под именем src.
Если dest - не директория, то, чтобы переписать dest, нужно право w. Если директория, то x, чтобы войти в директорию и w, чтобы записать туда файл.

Системные директории
/ (корневая директория) - корневая директория является основой всей файловой системы. От неё «растут» все остальные папки.
/**bin** (базовые исполняемые файлы) - в этой директории хранятся основные программы и команды, необходимые для работы системы.
/**sbin** (системные бинарные файлы) - содержит исполняемые файлы, которые обычно используются для системного администрирования и управления. Эти команды требуют прав суперпользователя (root) и не предназначены для ис-

пользования обычными пользователями.
/**etc** (конфигурационные файлы) - здесь находятся конфигурационные файлы (содержат настройки и параметры для операционной системы и приложений) системы и её сервисов. Пример файлов: /etc/passwd (информация о пользователях).
/**dev** (файлы устройств) - директория /dev содержит файлы устройств, которые представляют различные физические устройства, такие как жёсткие диски, CD/DVD-приводы, USB-устройства и др. Эти файлы — это своего рода интерфейсы, позволяющие приложениям и системе взаимодействовать с оборудованием. /dev/null — это специальный файл устройства, куда можно отправлять любые данные, которые не нужны. Любая информация, перенаправленная в этот файл, мгновенно удаляется и не занимает место в системе. Когда вы хотите запустить команду, но не хотите, чтобы её вывод отображался или сохранялся, вы можете перенаправить его в /dev/null.
/**proc** (виртуальная файловая система процессов) - в /proc содержатся файлы, представляющие системные процессы и информацию о них. Эти файлы генерируются динамически и отражают состояние системы в реальном времени.
/**sys** (системная информация) - похожа на /proc, эта директория динамически предоставляет интерфейс к данным ядра и оборудования. В /sys хранятся файлы и каталоги, представляющие устройства, драйверы и другие компоненты ядра.
/**usr** - содержит приложения и утилиты, которые не являются критически важными для базового запуска системы (то есть, их наличие не обязательно на уровне загрузки). Она делится на несколько поддиректорий (тоже системные библиотеки): /usr/bin — программы и утилиты для пользователей, которые не входят в базовую систему.
/usr/sbin — системные утилиты для администратора.
/usr/lib — библиотеки, необходимые для программ в /usr/bin и /usr/sbin. И тд
/**var** (переменные данные) - директория для данных, которые изменяются во время работы системы, таких как логи, файлы кэша и временные файлы приложений.
/**tmp** (временные файлы) - здесь хранятся временные файлы, которые создаются различными программами и процессами во время их работы. Директория очищается при каждом перезапуске системы, поэтому не подходит для хранения постоянных данных.
/**home** (домашние директории пользователей) - в этой директории находятся личные директории всех пользователей системы.
/**root** (домашняя директория суперпользователя) - это домашняя директория пользователя root (суперпользователя). Суперпользователь может выполнять любые действия на системе, включая просмотр и изменение любых файлов, управление пользователями и групповыми правами, установку и удаление программного обеспечения, а также настройку оборудования и служб. **sudo** (от английского "superuser do") — это команда, которая позволяет временно выполнять действия от имени суперпользователя или другого пользователя (требуется пароль).
/**lib и /lib64** (системные библиотеки) - директория /lib содержит основные библиотеки, необходимые для работы системы и её утилит, находящихся в /bin и /sbin. Эти библиотеки важны для загрузки и работы системы. В системах с 64-битной архитектурой может быть директория /lib64, где хранятся библиотеки для 64-битных приложений.
/**media** (точки монтирования (это каталог или файл, с помощью которого обеспечивается доступ к новой файловой системе) съемных устройств) - в /media автоматически монтируются внешние носители, такие как USB-накопители, CD/DVD-диски, внешние жесткие диски.
/**mnt** (временные точки монтирования) - директория для временного монтирования файловых систем.
/**opt** (дополнительные или опциональные приложения) - в /opt устанавливаются дополнительные или сторонние программы, не входящие в стандартные репозитории системы.
/**boot** (файлы загрузчика) - в этой директории находятся файлы, необходимые для загрузки системы (например, ядро - vmlinuz)
/**srv** (данные служб) - директория для данных, используемых службами, которые предоставляют ресурсы для удалённых клиентов.
/**lost+found** (потерянные файлы) - эта директория присутствует на каждом разделе с файловой системой ext4 (и других типов ext). Содержит "восстановленные" файлы, которые были повреждены или потеряны в результате сбоя. Файлы в lost+found создаются утилитой fsck при проверке и восстановлении файловой системы.

Откуда терминал знает, что нужно запустить файл, путь к которому не указан?
Пути до программ исполняемых файлов прописаны в переменной PATH, SHELL пробегает по директориям /bin.

Если существует программа и файл с одинаковыми именами, что запустится при вызове этого имени? (приоритет исполнения)
Запустится программа.

EOF
EOF (End Of File) — это условный индикатор завершения ввода данных в файл или поток.
Когда ввод производится из командной строки, EOF может быть достигнут вручную, если нажать Ctrl+D. Это сигнал для завершения ввода, после которого программа, читающая данные, прекратит ожидание.
С конструкцией <<EOF можно передавать блоки текста в команды, пока не встретится слово EOF, после которого текст будет считаться завершённым.
cat <<EOF
This is a multi-line
string input to the cat command.

EOF
Вывод:
This is a multi-line
string input to the cat command.

Экранирование
Экранирование — это способ использования специальных символов так, чтобы они воспринимались как обычные символы и не имели особого значения в строках или командах, выполняется с помощью символа обратного слэша (\).
Многие символы, такие как *, $, ?, &, |, <, >, ; имеют специальные функции в командной оболочке. Если мы хотим использовать эти символы в строке как обычные, их нужно экранировать.
Обратный слэш в конце строки позволяет продолжить команду на следующей строке.
Комбинации с используются для представления специальных символов: \n (новая строка), \t (табуляция), и т. д.

Вывести тремя разными способами какие-либо строки (например с 3 по 7) в файле
sed -n '3,7p' example.txt
head -n 7 example.txt | tail -n +3
cat example.txt | more +2 | head -n 5

&
Амперсанд (&) — это оператор, используемый для запуска команд в фоновом режиме. Обычно, когда вы запускаете команду, она блокирует терминал, пока не завершится. Использование & позволяет запускать команду асинхронно, освобождая терминал для ввода других команд.
В команде sed амперсанд используется для ссылки на всё совпадение целиком в замещающем выражении.
echo "hello world sed 's/world/&!/' - здесь & подставит найденное совпадение в результат: слово world будет заменено на world!, так как & ссылается на найденное значение world.
Амперсанд в потоках часто комбинируется с >, чтобы различать стандартный поток вывода (1) и стандартный поток ошибок (2).
command > output.txt 2>&1 - перенаправляет как стандартный вывод, так и ошибки в output.txt.
Иногда с помощью & объединяют несколько команд и перенаправляют их вывод вместе:
(command1 & command2) > output.txt - объединяет оба вывода command1 и command2 в output.txt.

*****
Одиночная звездочка * заменяет любое количество любых символов, включая отсутствие символов, в именах файлов и каталогов. Используется для выбора сразу нескольких файлов или каталогов, когда их имена совпадают по определённой маске. Ищет только в текущем каталоге и не заходит в подкаталоги.
Двойная звездочка ** используется для поиска файлов и каталогов на всех уровнях вложенности (рекурсивно) в текущем каталоге и его подкаталогах (если подрублен globstar).

globstar
Это опция Bash, которая расширяет возможности шаблонов (globbing), обычно используемых для поиска файлов и директорий по шаблону.
shopt — команда для управления настройками и опциями оболочки в bash.
-s - включить расширение;
-u - вырубить;
без опций - посмотреть активные опции.

cat > file1 < file2
Сначала > очищает файл
Затем cat выводит пустой file1
В последнюю очередь file1 перезаписывается, принимая на стандартный вход file2.

Какой алгоритм лежит в основе sort?
Соритировка слиянием

Почему поток ввода это 0 и тд
Файловый дескриптор — это целое число, которое операционная система использует для идентификации открытого файла или ресурса. Оно выступает как ссылка на структуру данных, содержащую информацию о текущем состоянии взаимодействия с файлом.
Дескрипторы уникальны только внутри одного процесса. Помимо файлов, дескрипторы могут указывать на: сокеты, каналы и устройства.
В каждом процессе в Unix автоматически открываются три стандартных файловых дескриптора (номера зарезервированы):
0 — Стандартный ввод (stdin)
1 — Стандартный вывод (stdout)
2 — Стандартный вывод ошибок (stderr)