Файловые системы

Файловые системы, в том числе на ext2/ext4. Симлинки, хардлинки.

**Оглавление**

[Файловые системы и файлы](#_2fku94s1jp8q)

[Блок или кластер](#_3pi2ojzcil2b)

[Суперблок в ФС Linux](#_6njp3qljjbyh)

[Файловые системы](#_jz5zvu2tebqk)

[Нежурналируемые файловые системы](#_fhjsjb3xtwc4)

[Журналируемые файловые системы](#_z02r9r2g3lvk)

[Атрибуты файлов в файловых системах](#_idergcemfnt9)

[Работа с файловыми системами в Linux](#_48r70ru6zdbn)

[VFS](#_v4gkx7kz8rfo)

[Файловые системы Linux](#_c0m4h4pe9gpj)

[Айноды (inode)](#_3yi1evdn6pys)

[Интересные возможности файловых систем и ОС в работе с ФС](#_44q66yu1qcki)

[Директория как диск](#_uxtjtt4xqrby)

[Монтирование](#_2uu2qr5argeu)

[Ярлыки](#_gd659bynovcy)

[Символические ссылки](#_p5sd134k8igd)

[Символические ссылки в Debian](#_zfnzlvxhfmh)

[Жесткие ссылки](#_iu7w8hwvoexd)

[Файлы . и ..](#_nddxiup0efal)

[Файловые потоки (альтернативные данные файлов)](#_dxks9pc0mwu3)

[Снимки (снапшоты)](#_b0kylk6tsimw)

[Управление томами](#_9dl93svfa6vu)

[Дополнительная информация — файловая система на примере FAT](#_9hu5s6efimw2)

[Загрузочный сектор](#_3vq2m3swfr61)

[FAT-таблица](#_x2rdofwdg6mg)

[Директории](#_qc6h4caz3ty8)

[Имя файла и расширение](#_4hhu1r6cn7wn)

[Корневой каталог](#_oxu08n7ykrhw)

[Развитие (VFAT, exFAT)](#_kga7q7vvcmyf)

[Практическое задание](#_2e38j070n6wb)

[Дополнительные материалы](#_2xcytpi)

[Используемая литература](#_1ci93xb)

# 

# Файловые системы и файлы

Файл — это именованная область данных на диске. Это может быть картинка, фильм, аудиозапись, исполняемая программа, отчет и т. д.

Чтобы с файлами можно было работать, на диске должна быть создана соответствующая структура. В современных операционных системах необходимо, чтобы можно было хранить, создавать, изменять, удалять:

* иерархически организованные папки (директории, каталоги), в которых хранятся файлы;
* файлы, двоичные и текстовые, хранящие программы или данные;
* атрибуты папок и файлов.

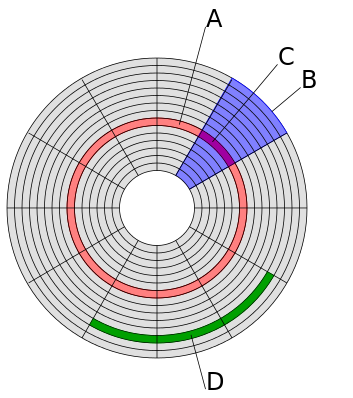
Также дополнительно могут присутствовать и другие возможности, такие как:

* ссылки (когда один файл может присутствовать в нескольких директориях);
* файловые потоки и иные возможности.

Чтобы обеспечить эти функции, используются файловые системы. Файловая система — это порядок организации, хранения и именования данных. Такие системы предоставляют API для работы с файлами. Для прикладных процессов работа с файлами на разных файловых системах выглядит примерно одинаково. Потому что обращение к файлам осуществляется через API операционной системы, который реализуется соответствующим драйвером файловой системы. Он непосредственно работает со структурами системы, умеет читать, писать и модифицировать данные на диске.

## Блок или кластер

Минимальная единица, с которой работает файловая система, — это блок или кластер. Если для диска минимальная единица, в которой может храниться информация, — это сектор, то на уровне файловых систем это блок (в ОС Linux) или кластер (в файловых системах от Microsoft). Блок состоит из нескольких секторов. Мы не можем занять в ФС места меньше, чем один блок.



*A — дорожка, B — геометрический сектор, C — сектор дорожки, D — кластер (блок)*

Кластер, как правило, состоит из нескольких соседних секторов дорожки, хотя может и из одного. Часто файловые системы оперируют кластерами, и кластер является минимальным размером, выделяемым для файла. Например, в DOS для FAT16, если размер кластера составлял 2 Кб, а требовалось записать на диск файл размером 4 байта, для записи фактически отводился весь кластер. И это несмотря на то, что в оглавлении файловой системы было указано, что размер файла — 4 байта, физически файл занимал все 2 Кб.

## Суперблок в файловой системе Linux

При высокоуровневом форматировании диска или раздела создается определенная структура: секторы группируются в блоки или кластеры. В DOS / Windows используют термин «кластер». В Linux такие группы называются блоками. Размер блока задается при запуске утилиты форматирования.

Блоки объединяются в группы блоков. Группы блоков в файловой системе и блоки внутри группы нумеруются последовательно, начиная с 1. Первый блок на диске имеет номер 1 и принадлежит группе с номером 1.

Каждая группа блоков имеет одинаковое строение:

* **суперблок** (см. ниже);
* **таблица дескрипторов группы** (Group Descriptors) — содержит начальный адрес блока;
* **битовая карта блоков** (Block Bitmap) — описывает состояние выделения блоков группы. Каждый ее бит показывает, отведен ли соответствующий ему блок какому-либо файлу;
* **битовая карта айнодов** — индексных дескрипторов (INode Bitmap). Каждый ее бит показывает, отведен ли соответствующий ему блок какому-либо айноду;
* **таблица индексных дескрипторов** (INode Table);
* **область блоков данных** — служит для хранения файлов.

Первый блок в группе блоков называется суперблоком, остальные блоки индивидуальны для каждой группы. Суперблок хранится в первом блоке каждой группы (за исключением группы 1, в которой в первом блоке расположена загрузочная запись). Суперблок является начальной точкой файловой системы. Суперблок хранит метку тома, которая может использоваться в Linux вместо файла устройства. То есть вместо **/dev/sda5** можно использовать **LABEL=rootfs**, если для устройства **/dev/sda5** присвоена метка **rootfs**.

Информация, хранимая в суперблоке, используется для организации доступа к остальным данным на диске. В суперблоке определяется размер файловой системы, максимальное число файлов в разделе, объем свободного пространства и содержится информация о том, где искать незанятые участки. При запуске ОС суперблок считывается в память и все изменения файловой системы вначале находят отображение в копии суперблока, находящейся в ОП, и записываются на диск только периодически. Это позволяет повысить производительность системы, так как многие пользователи и процессы постоянно обновляют файлы. С другой стороны, при выключении системы суперблок обязательно должен быть записан на диск, что не позволяет выключать компьютер простым выключением питания. В противном случае, при следующей загрузке информация, записанная в суперблоке, окажется не соответствующей реальному состоянию файловой системы.

# Файловые системы

Файловые системы можно разделить на две группы в зависимости от того, контролируется в них факт успешного завершения операций с диском или нет.

Чтобы записать файл на диск, нужно выполнить следующие действия:

* зарезервировать место в файловой системе;
* записать данные на диск;
* записать информацию о местонахождении данных;
* создать имя файла.

В зависимости от того, как файловая система отреагирует на сбой в процессе выполнения данных действий, она определяется как нежурналируемая или журналируемая.

## Нежурналируемые файловые системы

По происхождению нежурналируемые системы более ранние, чем журналируемые. Примеры таких файловых систем — FAT и EXT2.

Если в процессе записи действие будет остановлено, возникнут ошибки файловой системы, поврежденные файлы и директории. Именно поэтому при выключении Windows без специальной опции при следующем старте запускалась проверка системы. В FAT использовался специальный бит, который указывал, нужно ли проверять систему.

Наверняка многие, кто работал в Windows и переходил от использования FAT к NTFS, обращали внимание, что проблем с проверкой больше не возникало — независимо от корректности завершения системы. Обусловлено это тем, что NTFS относится к журналируемым файловым системам. Они более устойчивы к непредвиденным событиям (таким как неожиданное выключение питания компьютера), но при этом требуют больше ресурсов. Поэтому EXT2 считается эталоном производительности файловой системы.

Использовать нежурналируемую систему сейчас оправданно в качестве разделов для временных файлов, аналогичных программных кешей. Не путать со SWAP — это отдельная, особого образа файловая система, доступная только для ядра Linux. В Windows, напротив, используется файл.

Также EXT2 применяется там, где требуется быстродействие и совместимость, например для раздела загрузки **/boot** при использовании LVM. Так как GRUB не умеет работать с LVM, для **/boot** оставляют особый раздел в EXT2.

Кроме того, нежурналируемые файловые системы используются на флеш-накопителях (**exFAT**, **ext2**), так как меньше используют операции перезаписи из-за отсутствия журнала.

## Журналируемые файловые системы

Журналируемые файловые системы работают по транзакционному принципу. В структуре раздела есть журнал (Journal), в котором записываются действия перед тем, как будут произведены. И после их выполнения журнал очищается. Транзакция — это набор действий, выполняемых как одно целое, то есть оно или полностью завершено, или нет. Если транзакция будет прервана, то в соответствии с записью в журнале эти изменения откатятся, и транзакция будет считаться невыполненной. Такой подход делает работу удобнее и надежнее. Чтобы в нежурналируемой системе обнаружить незавершенные действия, требуется проверить целиком весь диск, всю файловую систему.

Журналируемые файловые системы работают медленнее, чем если бы они были отформатированы в аналогичные нежурналируемые. Затраты на чтение и запись делают журналируемые файловые системы малопригодными для использования во флеш-накопителях.

К журналируемым файловым системам относятся NTFS, EXT4 и многие другие.

### Атрибуты файлов в файловых системах

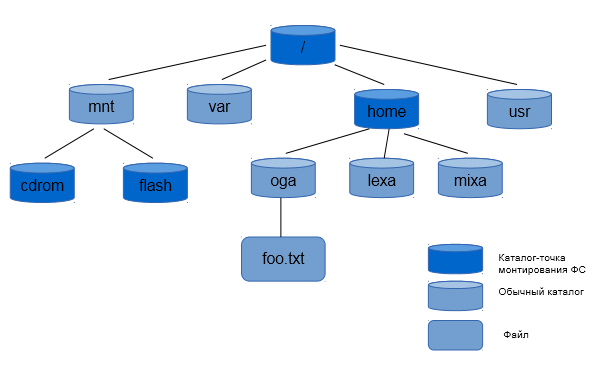
Набор атрибутов файлов полностью определяется форматом файловой системы. Например, в DOS имя состояло строго из имени и расширения. Является ли файл системным или скрытым, определялось атрибутами, а возможность его выполнения — расширением (.com, .exe, .bat).

В Linux расширений как таковых нет, возможность выполнения файла определяется его атрибутом, а понятия «скрытый» или «системный файл» отсутствуют. Системность файла определяется правами доступа и владельцем файла, а также расположением файла в иерархии файловой системы Linux. О том, что файл является скрытым, говорит точка в начале его названия (**.rc**, **.rc.local**, **.htaccess** — примеры скрытых файлов). При этом в Linux могут использоваться расширения на уровне приложений — и как дань традиции, и для опознавания файлов приложениями: это расширения **.c**, **.s**, **.cpp**, **.ko** и т. д.

# Работа с файловыми системами в Linux

## VFS

Основной файловой системой в Linux является VFS — Virtual File System. Но VFS — это виртуальная файловая система, она не присутствует на диске. VFS не является файловой системой в том виде, о котором мы говорили ранее. Это уровень абстракции над разными конкретными реализациями файловых систем. VFS предоставляет единообразный доступ к различным файловым системам.



Когда мы работаем в Linux, то взаимодействуем с корневым каталогом /. Но на самом деле корневой каталог это не то же самое, что оглавление диска c:\ в Windows. Это точка монтирования, созданная ядром Linux, то есть пустой каталог, в который присоединяется оглавление другой файловой системы. И в корневой каталог / присоединяется корень диска, в котором размещена система. При этом, если система обслуживает много пользователей, директория **/home** также может быть пустой, а в нее будет присоединяться (монтироваться) другой диск. Отдельно присоединяются внешние устройства, такие как CD-rom или flash. В отличие от DOS и Windows, имеющих отдельную иерархию в виде дисков E:, F: и т. д., в UNIX-подобных системах содержимое примонтированных дисков добавляется в единую иерархию VFS. Заранее создаются пустые директории, в которые при необходимости монтируются внешние устройства. Пока устройства смонтированы, для пользователя и приложений все выглядит так, будто файлы находятся в одной иерархии. По завершении работы, перед извлечением устройства, его нужно отмонтировать. Похожее действие выполняется для флеш-накопителей и в WIndows, в системном трее.

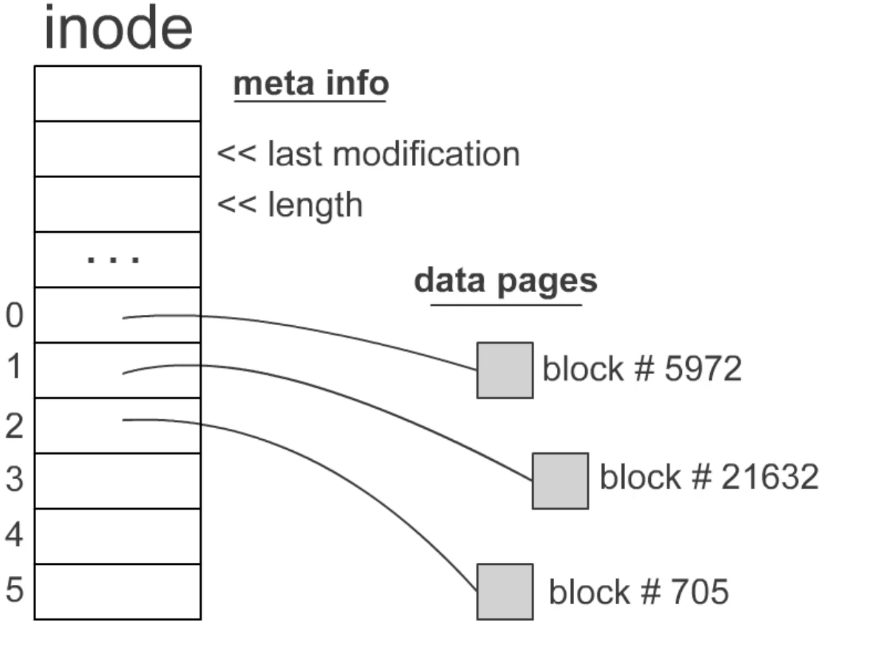
## Файловые системы Linux

* **Ext2**. Нежурналируемая файловая система. Была создана специально для Linux.
* **Ext3**. Развитие файловой системы **ext2**. Добавлен журнал. Максимальный размер файла 2 Тб. Может работать с дисками до 32 Тб (с размером блока 8 Кб).
* **Ext4.** Журналируемая файловая система, развитие **ext3**. Увеличен максимальный размер файловой системы до 1 эксбибайта — 260 байт (при размере блока 4 Кб). Увеличен размер одного файла до 16 Тб (240 байт). Временные метки наносятся с наносекундной точностью. Multiblock allocation — хранит информацию о количестве блоков, расположенных друг за другом (уменьшает фрагментацию).

## Айноды (inode)

В отличие от системы FAT, в Linux принята организация файловых системы с отдельным хранением атрибутов. Таким образом, каталог ссылается не сразу на файл, а на айнод (i-node). Он содержит атрибуты (метаинформацию о файле), который уже ссылается на файлы. Этот подход определяет логику работы с ФС в Linux.

Inode — индексный дескриптор, часть структуры **ext/ext3/ext4**.



*Примерная структура айнода*

Айнод хранит метаинформацию, атрибуты, права, время создания и модификацию, длину и ссылки на блоки.

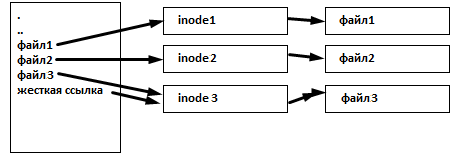
При создании файла мы занимаем 1 inode. Количество файлов не может быть больше, чем количество inode.

Те файловые системы, которые используют inode, можно разделить на 2 группы по принципу выделения айдов.

Такие файловые системы, как **ext2**, **ext3**, **ext4** содержат фиксированное количество айнодов, которое определяется при создании файловой системы, и изменить его почти невозможно. Если увеличить объем дискового пространства, то количество айнодов увеличится пропорционально добавленному объему.

Но есть файловые системы, такие как **jfs**, **xfs**, **btrfs**, **zfs**, — они позволяют увеличивать область данных для айнода.

Файловые системы с айнодами устроены таким образом, что каталог (с точки зрения файловой системы это файл особого образа) содержит ссылки на айноды, а айноды — на сами файлы.



ID айнодов можно посмотреть при помощи команды:

|  |
| --- |
| $ ls -ila |

Традиционно в Linux для каждого файла присутствуют атрибуты **r** (чтение), **w** (запись), **x** (выполнение), которые устанавливаются для трех групп: владелец, группа и остальные. На самом деле атрибутов больше: есть дополнительные (SUID, SGID, sticky bit), а также директории, символические ссылки, файловые потоки, которые тоже являются особого вида атрибутами.

Восьмеричная система счисления позволяет описать разный уровень прав просмотра и работы с файлами в виде одного из восьми значений (от 0 до 7).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | r | w | x |  | r | w | x |  |  |
| — | — | — | — |  | 0 | 0 | 0 |  | 0 |
| x | — | — | x |  | 0 | 0 | 1 |  | 1 |
| w | — | w | — |  | 0 | 1 | 0 |  | 2 |
| wx | — | w | x |  | 0 | 1 | 1 |  | 3 |
| r | r | — | — |  | 1 | 0 | 0 |  | 4 |
| rx | r | — | x |  | 1 | 0 | 1 |  | 5 |
| rw | r | w | — |  | 1 | 1 | 0 |  | 6 |
| rwx | r | w | x |  | 1 | 1 | 1 |  | 7 |

Таким образом, права 777 означают, что файл могут читать, изменять и исполнять все.

Права изменяются с помощью **chmod**:

|  |
| --- |
| $ chmod 777 myfile |

Особым образом интерпретируется атрибут **x** для директорий. По умолчанию он установлен. Если атрибут **x** снят с директории, ее невозможно сделать активной (**cd**).

Кроме того, атрибут **x** дает доступ к inode. Если атрибут **x** снят, файл невозможно ни изменить, ни прочитать — так как нельзя получить доступ к атрибутам.

Существует атрибут, задаваемый по умолчанию. Для файлов это 644, для директорий — 755. Чтобы задать атрибут по умолчанию сразу, используется следующий принцип: берется некая маска, которая вычитается из 666 для файлов и из 777 для директорий. По умолчанию это 022 — таким образом, 666 — 022 = 744, 777 — 022 = 755.

Такая операция задается командой **umask**:

|  |
| --- |
| $ umask 022 |

### Интересные возможности файловых систем и ОС в работе с ФС

Рассмотрим возможности, существующие для разных файловых систем.

#### **Директория как диск**

Команда **subst** в DOS и Windows позволяла использовать директорию как отдельный диск.

Пример:

|  |
| --- |
| C:\> subst z: c:\Users  C:\> z:  C:\> dir  C:\> subst /D z: |

#### **Монтирование**

В Linux используется только виртуальная файловая система, остальные монтируются. Точка монтирования — пустая директория, в которую после монтирования будет отображаться корень примонтированной файловой системы.

|  |
| --- |
| # [ -d /mnt/cdrom ]||mkdir /mnt/cdrom  # cd /mnt/cdrom  # mount /dev/cdrom /mnt/cdrom  # cd /mnt/cdrom  # cd /  # umount /mnt/cdrom |

В NTFS также поддерживаются точки монтирования. В среде Windows существует два вида точек монтирования:

* точка монтирования каталога (junction point);
* точка монтирования тома (англ. volume mount point).

Создание точек монтирования первого типа осуществляется через консольную команду **mklink /J**, второго типа — через команду **mountvol**.

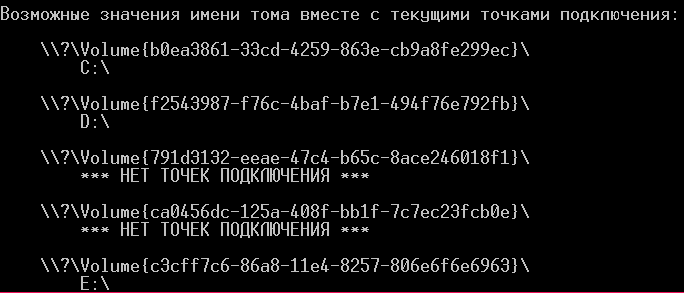
|  |
| --- |
| С:\> mklink /J c:\tmp\disk\_d d:\  C:\> dir c:\tmp\disk\_d |



В этом случае, в отличие от монтирования, в Linux создавать пустую директорию не нужно.

Для монтирования этих точек в Windows используется команда **mountvol**. По умолчанию она выводит список опций и перечень точек монтирования:

|  |
| --- |
| c:\> mountvol |



#### **Ярлыки**

Ярлыки не относятся к возможностям операционной системы, и можно ли с ними работать, определяется на уровне приложения (как правило, менеджера рабочего стола).

Изначально в Windows присутствовали файлы **PIF** для настроек параметров запуска DOS-приложений, к которым добавились файлы **.LNK**, указывающие на местоположение программы или документа. В окружении рабочего стола клик на ярлык приводил к тем же последствиям, что и клик на значок самой программы, но при удалении файла ярлык переставал работать. При перемещении ярлыка программа не меняла расположение, а при попытке открыть ярлык, например, в строке командного интерпретатора, он не срабатывал. В консоль выводился не документ, на который ярлык ссылался (даже если он текстовый), а содержание самого ярлыка.

#### **Символические ссылки**

Символическая ссылка — особого рода файл, ссылающийся на местоположение другого файла. Операционная система представляет символическую ссылку приложению так, как если бы он открыл сам файл. В этом состоит отличие символической ссылки от ярлыка.

При попытке прочитать содержимое символической ссылки на экран будет выведено содержимое файла. В работе использование символической ссылки почти никак не отличается от указания настоящего имени файла. Символические ссылки уместно сравнить с ярлыками, которые обрабатываются на уровне ядра операционной системы.

В Linux символические ссылки создаются с помощью команды **ln -s файл имя\_ссылки**.

Вывод оглавления директории с атрибутами **ls -la** также указывает и место назначения ссылок.

|  |
| --- |
| $ echo Hello >test  $ ln -s test test1  $ ls -la  $ cat test1  $ rm test  $ la -la  $ cat test1 |

При удалении файла, на который ведет ссылка, она становится «битой». По своей природе символическая ссылка — это особого рода файл с указанием местоположения. Поэтому такие ссылки могут вести на директории и на файлы, размещенные в других (смонтированных) разделах. Однако при размонтировании отдела такая ссылка не будет работать.

В остальном работа с символической ссылкой и файлом — одно и то же. В отличие от действия с ярлыком, при копировании ссылки будет создан сам файл.

В Windows символические ссылки создаются с помощью **mklink**, который по умолчанию создает символическую ссылку на файл, с опцией **/D** — на директорию, с опцией **/H** — жесткую ссылку. Об опции **/J** мы уже говорили.

#### **Символические ссылки в Debian**

Символические ссылки активно используются в Debian и Ubuntu. Например, при работе с веб-сервером Apache2 в директории **/etc/apache2/conf-available** находятся доступные конфигурационные файлы, а в **/etc/apache2/conf-enabled** — символические ссылки на соответствующие файлы из **conf-available** с аналогичными именами. Утилиты **a2enconf**, **a2disconf** и аналогичные фактически создают ссылки (**ln -s**) и удаляют их (**rm**). Подобный подход применяется и в **nginx** при использовании в Debian или Ubuntu, с той разницей, что ссылки создавать и удалять приходится через **ln** и **rm**.

#### **Жесткие ссылки**

Фактически жесткая ссылка — это еще одно имя файла. В Linux жесткая ссылка создается командой **ln имяфайла имяссылки**.

Пример:

|  |
| --- |
| $ echo Hello >test  $ ln test test1  $ ls -ila  $ cat test1  $ rm test  $ la -ila  $ cat test1 |

Даже при удалении исходного файла такая ссылка продолжает работать. Более того, без номера inode невозможно выяснить, какой файл оригинальный, а какой — ссылка. Очевидно, что жесткие ссылки невозможно создавать на другие разделы. Также невозможно создать жесткую ссылку на каталог. В Windows жесткие ссылки создаются командой **mklink** с параметром **/H**.

## Файлы . и ..

Чтобы перейти на уровень выше, нужно ввести команду:

|  |
| --- |
| $ cd .. |

Чтобы запустить программу из текущей директории, в Linux нужно указать ее полный путь:

|  |
| --- |
| $ /home/user/test.sh |

Или можно указать ее положение относительно текущей директории (при условии, что мы уже находимся в **/home/user** и там лежит скрипт **test.sh**, обладающий правами на исполнение):

|  |
| --- |
| $ ./test.sh |

А эта команда выполнит переход в текущую директорию, но мы и так в ней находимся, так что ничего не изменится:

|  |
| --- |
| $ cd . |

. и .. — не специальные обозначения, а жесткие ссылки на директории (другие виды жестких ссылок на директории в Linux запрещены).

. — это жесткая ссылка на каталог, в котором мы находимся.

.. — это жесткая ссылка на каталог, которому принадлежит тот, в котором мы находимся.

#### **Файловые потоки (альтернативные данные файлов)**

Файловые потоки были созданы в NTFS для совместимости с ФС HFS в MAC OS и использовались для хранения метаданных — таких как иконки приложений. В противовес жестким ссылкам файловый поток может содержать, кроме основного содержимого, альтернативное. Большинство программ игнорируют эту возможность. При попытке копировать файл штатными средствами Windows альтернативные данные не могут быть скопированы.

Возможности файловых потоков могут использоваться вирусами.

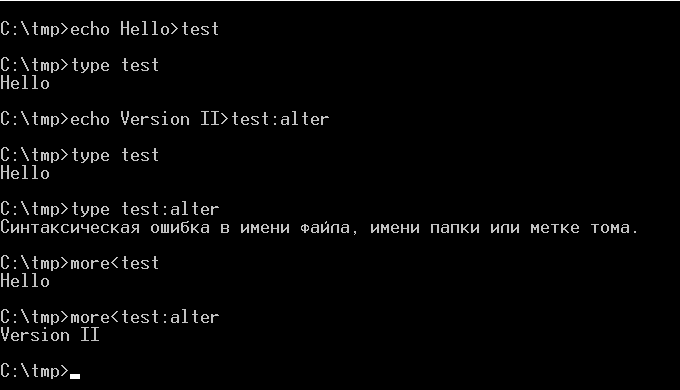
**Примеры команд**

Создать файл и альтернативный файловый поток:

|  |
| --- |
| C:\TMP\> echo Hello >test  C:\TMP\> type test  C:\TMP\>echo VersionII > test:alter |

Вывести на экран (**type** и некоторые другие привычные способы работы для потока не сработают):

|  |
| --- |
| C:\TMP\> type test  C:\TMP\> more <test  C:\TMP\> type test:alter  C:\TMP\> more <test:alter |



#### **Снимки (снапшоты)**

Снапшоты используются для создания копий и восстановления системы. Их идея заключается в том, что систему без остановки работы с диском можно «заставить» не перезаписывать старую информацию, а только добавлять данные. Если «законсервировать» всю информацию таким образом, мы получим снимок (снапшот), который можно смонтировать.

Получать снимки можно в файловой системе ZFS и в системе управления томами LVM.

#### **Управление томами**

Управление томами подразумевает возможность объединить несколько физических устройств в группу томов, в которой можно выделить логические тома и использовать как диски и разделы. Такой подход позволяет безболезненно увеличивать и перераспределять пространство между устройствами, а также перемещать данные с одних дисков на другие без остановки работы.

Управление томами обеспечивается системой LVM и также встроено в файловую систему ZFS.

# Дополнительная информация — файловая система на примере FAT

Одна из простейших файловых систем — FAT. Она до сих пор применяется, в частности для работы с флешками.

FAT (File Allocation Table, «таблица размещения файлов») — известная и простая файловая система, созданная для MS DOS Биллом Гейтсом и Марком Мак-Дональдом в 1976–1977 годах. Она по сей день используется для флеш-накопителей.

В FAT метаинформация и атрибуты хранятся в самой директории.

Рассмотрим устройство FAT на примере FAT12.

Нулевой сектор содержит Boot Record, а он — информацию о файловой системе. Затем следует таблица размещения файлов (FAT), которая дублируется, чтобы увеличить надежность. Первая и вторая таблицы занимают 1–9 и 10–18 секторы соответственно. Далее несколько секторов отводится под корневой каталог (он отличается от других и единственный, который обязательно должен присутствовать). Интересно, что первые версии FAT не позволяли создавать подкаталоги. После корневого каталога размещаются секторы с содержимым файлов.

## Загрузочный сектор

Загрузочный сектор содержит информацию о файловой системе. Его формат может варьироваться в разных версиях FAT.

Значения байтов с порядковыми номерами:

* 0–3 — JMP BootStart;
* 4–10 — игнорируются;
* 11–12 — количество байтов в каждом секторе;
* 13 — количество секторов в каждом кластере;
* 14–15 — количество зарезервированных секторов;
* 16 — количество FAT-таблиц;
* 17–18 — максимальное число файлов и папок в корневой директории;
* 19–20 — общее число секторов;
* 21 — игнорируется;
* 22–23 — количество секторов, которое требуется для каждой FAT-таблицы;
* 24–25 — количество секторов на дорожке;
* 26–27 — количество головок;
* 28–31 — игнорируется;
* 32–35 — игнорируется в FAT12;
* 36–37 — игнорируется;
* 38 — boot signature — загрузочная сигнатура;
* 39–42 — id тома;
* 43–53 — метка тома;
* 54–61 — тип файловой системы;
* 62 — далее следует начало программы BootStart (которая передаст управление в **IO.SYS**, не опираясь на оглавление, а физически считывая следующий кластер после таблиц FAT).

Здесь и далее многие поля заполняются до своей границы пробелами. Это касается метки, названия файловой системы. Такой же подход используется и в именах файлов, где под имя и расширение выделено по 8 и 3 байта соответственно.

## FAT-таблица

FAT — это таблица, которая содержит массив индексных указателей («ячеек»), соответствующих кластерам области данных. Каждый элемент содержит порядковый номер следующего кластера файла либо установлен в специальное значение, которое может меняться в зависимости от версии FAT.

Вот более подробный список значений и того, что под ними подразумевается:

* 0 × 000 — не используется;
* 0 × FF0-0 × FF6 — зарезервирован;
* 0 × FF7 — плохой кластер (поврежден и не используется);
* 0 × FF8-0 × FFF — последний кластер в файле.

## Директории

Директории, включая корневую, хранятся на диске как файлы (фактически являясь особого вида файлами) и занимают один или несколько кластеров. Каждый кластер (512 байт) содержит указатели на 16 файлов, то есть по 32 байта на каждый указатель. Указатель описывает файл на диске или поддиректорию, которая тоже является особого вида файлом, а также указывает на кластер, где находится файл.

Номерам байтов соответствовало следующее содержимое:

* 0–7 — имя файла (8 байт);
* 8–10 — расширение (3 байта);
* 11 — атрибуты;
* 12–13 — зарезервировано, не используется;
* 14–15 — время создания;
* 16–17 — дата создания;
* 18–19 — дата последнего использования;
* 20–21 — игнорируется в FAT12;
* 22–23 — дата последнего изменения;
* 24–25 — время последнего изменения;
* 26–27 — указание на первый кластер файла или директории;
* 28–31 — размер файла в байтах.

Особым образом записывались даты. Два байта содержали следующие значения:

* 0-4 — день месяца, допускаются значения 1–31;
* 5-8 — месяц года, допускаются значения 1–12;
* биты 9–15 — год считая от 1980 («эпоха MS DOS»); возможны значения от 0 до 127 включительно, то есть 1980–2107 годы.

В FAT не используется специальный признак существования файла: чтобы указать, что файл удален, применяется первый байт имени. При удалении он устанавливается в значение 0 × 29, а кластеры помечаются как свободные. Так как указание на первый кластер остается, файл возможно восстановить в том случае, если он занимал несколько соседних секторов. Такие программы, как Undelete, Unerase и встроенное восстановление файлов в Dos Navigator, запрашивали первую, затертую букву и выводили информацию о восстановлении. Например, если указанный кластер был задействован другим файлом, восстанавливать уже было нечего.

Если следующая запись при чтении каталога начиналась с 0 × 00 (условно — первый байт в позиции имени), это означало, что больше указателей в каталоге нет.

Значения байта атрибутов:

* 0 × 01 — только чтение (Read Only);
* 0 × 02 — скрытый (**Hidden. io.sys**, **msdos.sys** и некоторые другие файлы в корневом каталоге имели атрибут, по умолчанию не отображались при выводе **dir**);
* 0 × 04 — системный (**System, io.sys**, **msdos.sys** в корневом каталоге имели этот атрибут);
* 0 × 08 — **Volume Label** — метка тома, хранится так же, как и файл. В отличие от остальных, имя не делится на имя и расширение, потому в его значении может быть до 11 символов. Кроме того, в комбинации с предыдущими атрибутами метка указывает, что это не запись отдельного файла, а часть длинного имени LFN;
* 0 × 10 — указание на то, что файл является папкой;
* 0 × 20 — архивный (Archive); использовался нечасто, но предполагалось, что его будут применять программы архивации для указания, что файл был заархивирован или подлежит архивации;
* 0 × 40 — не используется (зарезервировано для устройств);
* 0 × 80 — не используется.

Значения могут комбинироваться. Например, если нужно, чтобы файл представлял собой скрытую системную папку, соответствующий байт устанавливается в значение 0 × 16, потому что 0 × 16 = 0 × 02 + 0 × 04 + 0 × 10.

Для расширения LFN используется признак 0 × 0F = 0 × 01 + 0 × 02 + 0 × 04 + 0 × 08. Каждая такая запись является не файлом, а частью имени длинного файла.

Утилита **attrib** позволяла устанавливать четыре из указанных атрибутов: **R** — только для чтения, **A** — архивный, **S** — системеный, **H** — скрытый.

## Имя файла и расширение

В FAT12 выделяется 8 символов под имя файла и 3 символа под расширение — так называемый формат 8.3. Если имя файла занимает меньше 8 символов, оставшиеся заполняются пробелами (ASCII 32 или 0 × 20), то же касается расширений. Директории также могут иметь расширение. Точка никак не хранится, но при работе в DOS необходима. Расширение может быть пустым, в то время как имя файла должно содержать хотя бы один символ.

Все имена и расширения хранятся в ВЕРХНЕМ РЕГИСТРЕ, при этом для работы регистр значения не имеет: COMMAND.COM, Command.Com и command.com — одно и то же имя файла. В одной директории не может быть одновременно двух файлов (обычных или директорий) с одинаковым именем и расширением: если есть файл **config.sys**, директорию **config.sys** уже создать нельзя.

Примеры:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Имя файла** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| command.com | С | O | M | M | A | N | D |  | C | O | M |
| io.sys | I | O |  |  |  |  |  |  | S | Y | S |
| msdos.sys | M | S | D | O | S |  |  |  | S | Y | S |
| hello.c | H | E | L | L | O |  |  |  | C |  |  |
| DOS | D | O | S |  |  |  |  |  |  |  |  |
| WINDOWS | W | I | N | D | O | W | S |  |  |  |  |
| descript.ion | D | E | S | C | R | I | P | T | I | O | N |

Имена файлов даны как при выводе **dir** или в файловом менеджере: файлы — в нижнем регистре, директории — в верхнем. На самом деле это условность, так как файловая система здесь регистронезависима. Иногда с файлами поставлялся файл **descript.ion**: фактически его имя было **descript**, нестандартное разрешение — **ion**. Обычно он содержал описание файлов и / или ASCII-арт.

Также в директории присутствуют специальные файлы «.» и «..».

«.» является ссылкой на текущую директорию (**cd .** не меняет текущую директорию). «..» — ссылкой на родительскую директорию (**cd ..** переходит на уровень выше).

## Корневой каталог

По структуре корневой каталог аналогичен другим каталогам (директориям, папкам), но на него не бывает ссылки. Он не имеет ни имени (в системе к нему можно обратиться по имени «\», **cd \** — перейти в корень диска), ни атрибутов. По той же причине корневой каталог имеет фиксированный размер. Это накладывает ограничение на количество размещаемых в корне файлов и при этом позволяет передать управление в **io.sys**, не опираясь на сам корневой каталог и учитывая только физическое расположение **io.sys** и **msdos.sys** после корневого каталога.

## Развитие (VFAT, exFAT)

VFAT — совместимое с FAT расширение, добавляющее записи в формате LFN (Long File Name) в кодировке Unicode-16. На одну SFN (Short File Name — традиционную для FAT) добавляется одна запись или несколько LFN-записей. С появлением такого расширения максимальный размер имени файла увеличился до 255 символов. Кроме того, теперь точка включалась в состав имени. Таким образом, файл с длинным именем в Windows 95 в MS DOS или Windows 3.11 выглядел как РАБОЧИ~1 — это SFN-запись.

**exFAT** (extended FAT) — это расширенная версия FAT, оптимизированная под флеш-накопители, которые имеют ограниченное число циклов перезаписи. FAT как нежурналируемая система щадяще относится к перезаписям, не тратя дополнительные операции на журнал. В exFAT были созданы дополнительные механизмы для уменьшения числа перезаписи: исключение перезаписи одного и того же сектора, улучшение распределения записанных кластеров благодаря бит-карте свободного места и т. д.

# Практическое задание

Ответьте на вопросы в Google Docs, максимально подробно объясняя свою точку зрения. Приложите ссылку на документ к уроку.

1. В чем плюсы нежурналируемой файловой системы?
2. В чем плюсы журналируемой файловой системы?
3. Что такое inode?
4. Где хранится имя файла?
5. Можно ли увеличить количество inode?
6. Тратится ли inode при создании жесткой ссылки (hard link)?
7. Почему жесткие ссылки можно делать только в пределах одной файловой системы?
8. Что такое . и ..?
9. Почему нельзя уменьшить размер файловой системы без простоя?
10. Что такое суперблок в **ext4**?

Задачи со \* предназначены для продвинутых учеников.

# Дополнительные материалы

1. [Суперблок в линуксе](https://4admin.info/%D1%81%D1%83%D0%BF%D0%B5%D1%80%D0%B1%D0%BB%D0%BE%D0%BA-%D0%B2-%D0%BB%D0%B8%D0%BD%D1%83%D0%BA%D1%81%D0%B5/).
2. [Ext2 и Ext3: концепции и анализ](http://artlib.osu.ru/Docs/piter/bookchap/978546901311.html).
3. [Восстановление удаленных разделов](http://www.rdm.kiev.ua/pages/lessons/urok4/).
4. [Команда MOUNTVOL — создание, удаление и просмотр точек подключения томов](http://ab57.ru/cmdlist/mountvol.html).
5. [Использование альтернативных файловых потоков NTFS](http://hex.pp.ua/using-alternate-data-streams.php).
6. [ZFS](http://xgu.ru/wiki/ZFS).
7. [LVM](http://xgu.ru/wiki/LVM).
8. [Сравнение\_ZFS\_и\_Linux\_LVM\_+\_RAID](http://xgu.ru/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_ZFS_%D0%B8_Linux_LVM_+_RAID).
9. [Подключаем новый жесткий диск в Debian](http://freecoder.ru/content/podklyuchaem-novyi-zhestkii-disk-v-debian).
10. [SWAP](http://help.ubuntu.ru/wiki/swap).
11. [Командная строка и режим Rescue](http://www.lexpr.ru/node/413).
12. [Пишем загрузочный сектор](http://www.compdoc.ru/prog/asm/boot_sector/).

# Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. [Размер кластера по умолчанию для файловых систем FAT, NTFS и exFAT](https://support.microsoft.com/ru-ru/help/140365/default-cluster-size-for-ntfs,-fat,-and-exfat).
2. [Ext2 и Ext3: концепции и анализ](http://artlib.osu.ru/Docs/piter/bookchap/978546901311.html).
3. [Разбираемся в устройстве файловой системы FAT12](http://www.pvsm.ru/fajlovaya-sistema/14110).
4. [FAT](https://ru.wikipedia.org/wiki/FAT).
5. [Суперблок](https://www.opennet.ru/docs/RUS/linux_base/node67.html).
6. [Структура дискового раздела ext2](https://www.opennet.ru/docs/RUS/linux_base/node66.html).
7. [Block Bitmap](https://www.opennet.ru/docs/RUS/linux_base/node69.html).