CENTRE ESPECÍFIC
D´EDUCACIÓ A DISTÀNCIA DE
LA COMUNITAT VALENCIANA

UD 06.
LINUX: FILE MANAGEMENT

Computer Systems
CFGS DAW

Álvaro Maceda
a.macedaarranz@edu.gva.es

2022/2023
Version:221113.2151

License

Nomenclature

Throughout this unit different symbols will be used to distinguish important elements within the content. These symbols are:

Important

Attention

Interesting

TABLE OF CONTENTS

UT 06. Linux: file management

1. What is a file

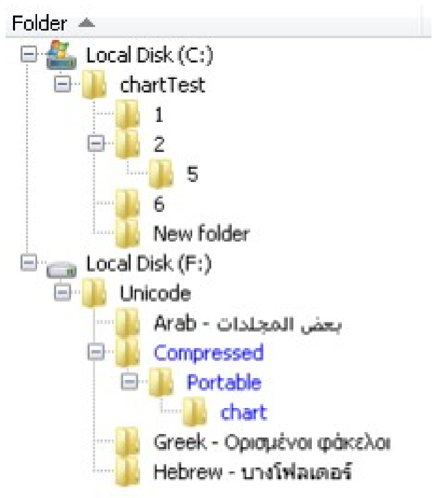
In computer systems, a file is just a series of ordered bits. As we have seen in previous units, these bits can represent any type of information: text, images, programs, etc. The files are usually stored on secondary storage systems such as hard disks or USB sticks.

The files are not written sequentially to disk: there are different ways of managing the disk space, for different purposes: speed, reliability, safety... Each method leads to a different disk organisation, which is called a file system.

It is important to bear in mind that files are only a series of bits, and that it is the applications that interpret their content as text, images, etc.

Folders

Since a system may require tens of thousands of files to operate, we nee a way of organizing them. The usual way of doing that is by using folders. Folders are containers where we can put the files. They may also contain other folders, forming a hierarchical structure. Folder information is also stored in the file system.



Example of a folder structure

2. File systems

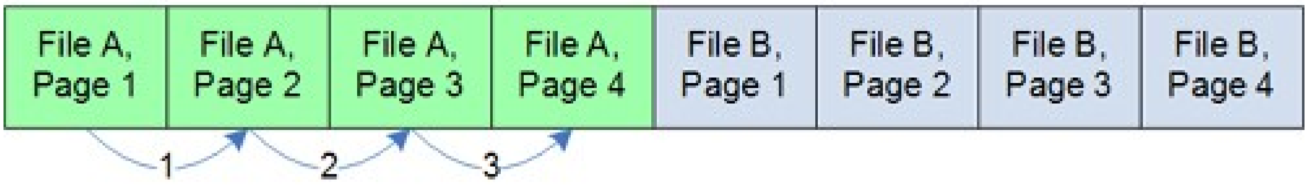
You can't write a file to a disk. Most of them present an interface which exposes the disk as a series of **sectors**. You can't write individual bits to a disk, only whole sectors. The most usual sector sizes are 4KB and 512 bytes.

A **sector** is the smallest unit that a disk allows to read or write

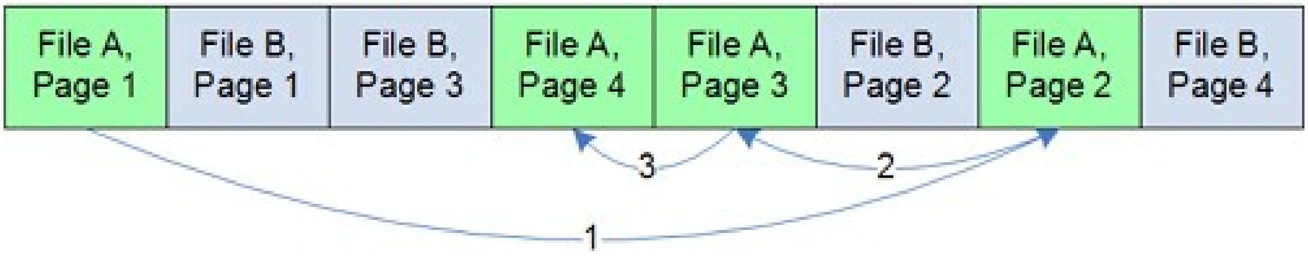
We could write the bytes of a file sequentially to disk but, if we did that, we could have similar problems as with memory allocation, having enough space inside the disk but fragmented in a way that we won't be able to store files of a certain size. In addition, the folder structure must also be stored in the file system. So file systems must decide how to organise the sectors of the disk to store information efficiently.

Different file systems have different approaches to arranging their data. Some file systems are faster than others, some have extra security features, and some support large storage capacity drives while others only work with smaller ones. There are file systems more robust and resistant to file corruption, while others trade robustness for extra speed.

What most filesystems do is something similar to memory pagination: they divide the disk space into blocks of a certain size, and the contents of the file are distributed through the disk using as many blocks as needed.



This would be a file stored sequentially



A file system can split the file into several non-contiguous blocks on the disk

A **block** or **cluster** is the minimum size that can be read or written to a file system

Usualy the block size is a multiple of the sector size to improve performance.

2.1 Most common file systems

There are hundreds of different file systems, but some of them are widely spread and used in most computers:

- **FAT** (File Allocation Table): It's relatively simple from the technical point of view. It was the default files system for Windows prior to Windows 2000, and nowadays it's compatible with all major operating systems. It can't work with large files or disks.
- **exFAT** (Extended File Allocation Table): Designed by Microsoft, is able to manage larger files than FAT. It's compatible with many operating systems and devices. Linux can manage this file system but you will probably need to install some packages (usually `exfat-fuse` and `exfat-utils`)

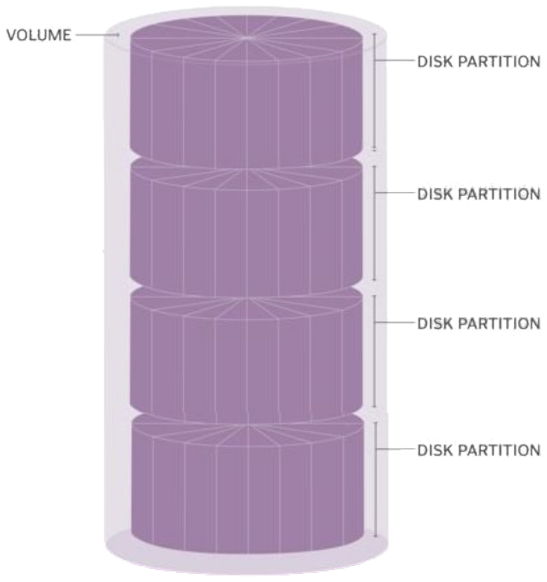
- **NTFS** (New Technology File System): It's the file system used as the default for Windows computers. Can be used by Linux and Windows machines, and read by Mac computers with additional drivers.
- **APFS** (APple File System): A proprietary file system used in Apple computers. It has features like snapshots and encryption.
- **ext4** (EXtended filesysTem): It's the default file system for most Linux distributions. Windows and Mac computers can't read it by default.
- **ZFS** (Zettabyte File System): It's an enterprise-level file system capable of merging devices, creating RAID volumes and adding or removing disks to the storage pool. Some Linux distributions include it by default.

The operating system should be able to interpret the file system organisation in order to work with it. If we try to use an unsupported file system, we won't be able to read or write information. For example, if we try to use a disk with the ext4 filesystem on a Windows machine, we won't be able to do it.

3. Partitions

Sometimes we don't want to use the same file system for the entire disk. This could be, for example, because we want to use different operating systems in the computer. Or, in the case of UEFI, because we need an special partition to boot the computer. In that case, we use partitions.

A disk **partition** is a region of the hard drive that is separated from other similar regions. Partitions enable users to divide a physical disk into logical spaces that behaves more or less like independent disks.



We can divide a disk into partitions, which behave like independent disks.

3.1 The partition table

There is an special place at the start of the disk to store how the disk is partitioned,. This information is called the **partition table**.

You can break your computer when tinkering with partitions. Use virtual machines for experimenting with them.

The main standards for storing the partition table are MBR and GPT.

A partition can take up the entire disk.

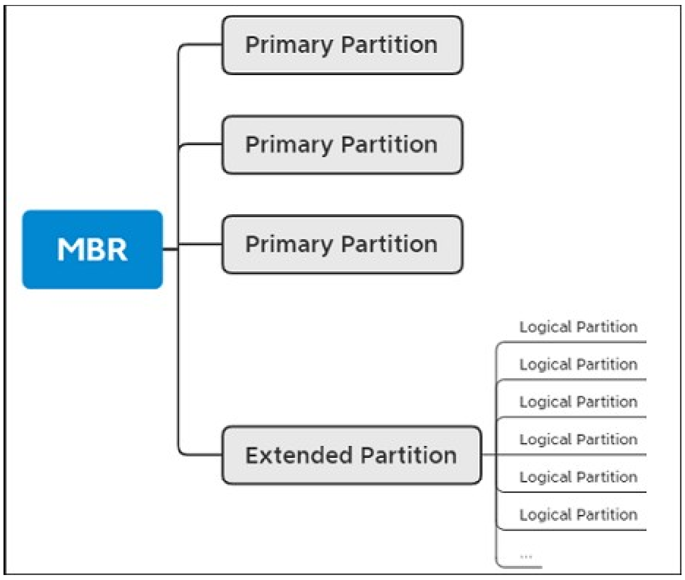
3.1.1 Master Boot Record (MBR)

MBR stands for Master Boot Record. It was introduced for the first time in 1983 with IBM PC DOS 2.0 It’s a legacy system and has been replaced by GPT, but is still used in some devices like USB drives.

The MBR is normally 512 bytes in size, which was the old size of a sector on a disk. It is always on the same place of the disk: Cylinder 0, Head 0, Sector 1 (non mechanical disks doesn’t have cylinders, heads or sectors, but the MBR is on the first addressable space of the disk). In addition to the partition table, it also includes a program that starts the booting process of the operating system.

MBR partitions can be of three types: primary partitions, extended partitions, and logical partitions. The reason for that is because MBR only allows 4 partitions. This limitation is overcome by extended and logical partitions.

If you need to create more than four partitions, what you can do is to create an extended partition. Inside that partition you can create as many logical partitions as you want. Some operating systems, like Windows, can’t boot from a logical partition.



The partition type is stored in the partition table so that the operating system knows how to manage the partition. The partition types of MBR partition tables are limited to one byte. You can view all MBR partition types in this link:

https://en.wikipedia.org/wiki/Partition_type#List_of_partition_IDs

MBR is a legacy system and should be used only for small drives that are not meant to boot a computer.

3.1.2 GUID Partition Table (GPT)

GPT is part of the United Extensive Firmware Interface standard (UEFI) the proposed replacement for the BIOS. It was designed to overcome the limitations of the MBR standard. The main advantages of GPT are:

- You can create an unlimited number of partitions
- It can handle drives up to 18 exabytes
- GPT is safer, adding checksums to avoid data corruption
- The boot process is faster if you use GPT and UEFI firmware
- The partition types are identified as a GUID (like `21686148-6449-6E6F-744E-656564454649`) instead of the 255 MBR partition types. You can check all the partition types here: https://en.wikipedia.org/wiki/GUID_Partition_Table#Partition_type_GUIDs

GPT uses Logical Block Addressing (LBA), an alternative to historical cylinder-head-sector addressing, where all disk space is viewed as a “stream” of consecutive storage addresses. The first sector is reserved for compatibility with MBR, so the GPT information starts at LBA(1).

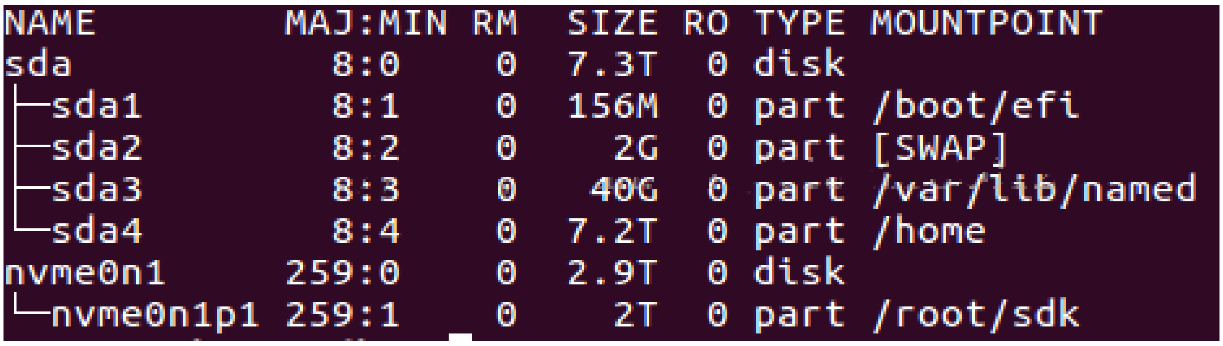
3.2 Managing partitions with Linux

To manage partitions in Linux we need admin permissions. For the terminal, we can open a root console executing `sudo -i`. For the graphical tools, the system will ask us for our password or an admin password.

To manage partitions we first need to know which devices we have connected to our system. That can be achieved with the command:

```
lsblk -e7 --all
```

The `-e7 --all` parameters are for excluding some special disk types from the output like ram disks (you can try `lsblk` directly on your system to view all block devices) An example output of this command could be:



```
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda           8:0    0  7.3T  0 disk
├─sda1        8:1    0  156M  0 part /boot/efi
├─sda2        8:2    0    2G  0 part [SWAP]
├─sda3        8:3    0   40G  0 part /var/lib/named
└─sda4        8:4    0  7.2T  0 part /home
nvme0n1     259:0    0  2.9T  0 disk
└─nvme0n1p1 259:1    0    2T  0 part /root/sdk
```

In this example, the system has two disks: `sda` and `nvme0n1`.

In Linux, SCSI, SATA and USB disks have the identifier `/dev/sdX`, where `X` is a letter starting with `a`. In this case, we have one SATA disk identified as `/dev/sda`. The partitions on those disks are identified as `/dev/sdXN`, where `N` is the partition number inside the disk. In this case, the disk has four partitions identified as `sda1`, `sda2`, `sda3` and `sda4`. If we had another SATA disk, it will have the identifier `/dev/sdb`, and the partitions will be `sdb1`, `sdb2`, etc. Partition numbers can be non-contiguous. This usually happens in the case of logical and extended partitions on MBR disks.

We have another disk connected as a NVMe disk. Those disks are identified as `/dev/nvmeN` where N is a number. The `n1` following the disk identifier is the namespace number (a feature that allows NVMe disks to be split in different disks) and it will usually be `n1`. Partition identifiers for NVMe disks are `nvmeNn1pM`, where `M` is the partition number.

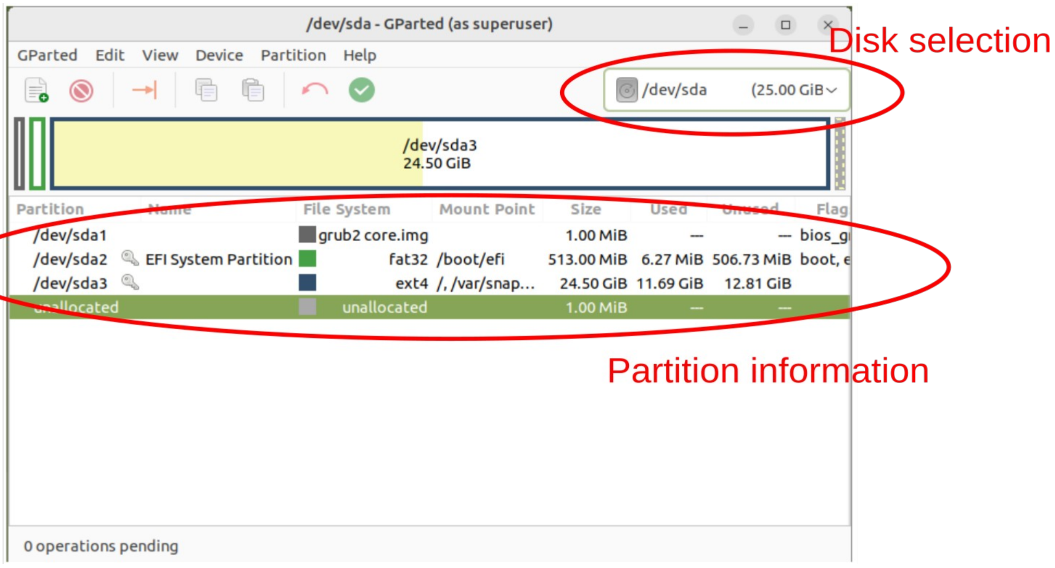
The tools that we will see for managing Linux partitions are gparted and cfdisk.

3.2.1 Gparted

gparted is a graphical tool to manage disk partitions. In some distributions it's not installed by default, you will need to install it with this command:

```
sudo apt install gparted
```

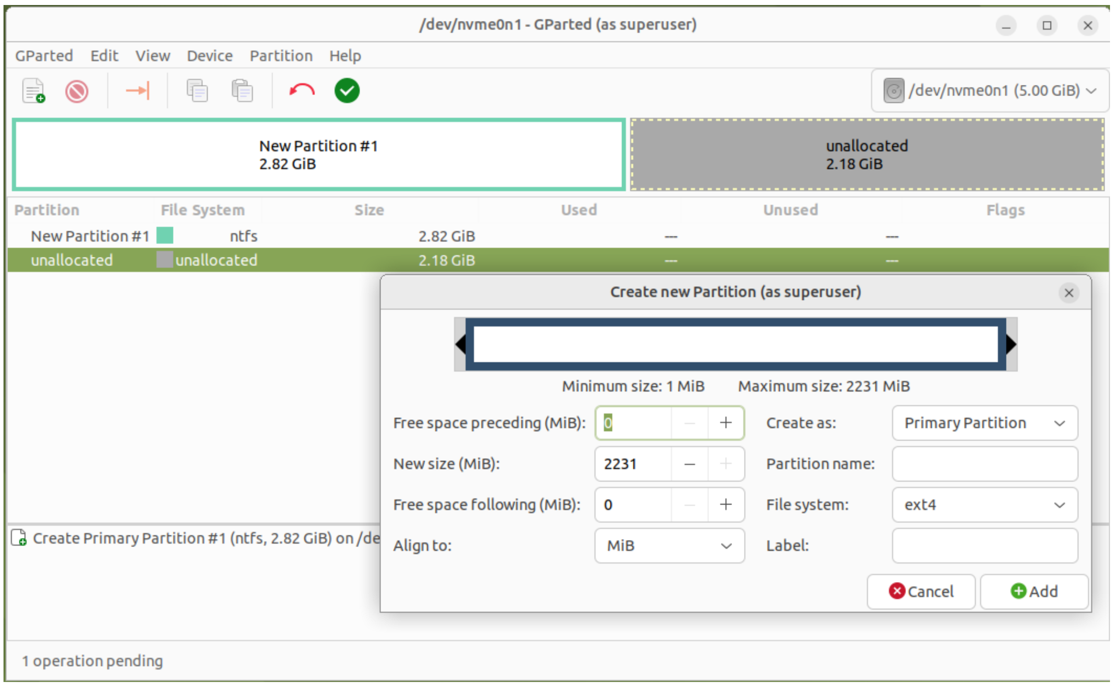
When you start gparted it will select a disk and show you information about the partitions on that disk. You can change the disk using the disk selector on the top right side of the window:



Partition	Name	File System	Mount Point	Size	Used	Unused	Flag
/dev/sda1		grub2 core.img		1.00 MiB	---	---	bios_g
/dev/sda2	EFI System Partition	fat32	/boot/efi	513.00 MiB	6.27 MiB	506.73 MiB	boot, e
/dev/sda3		ext4	/, /var/snap...	24.50 GiB	11.69 GiB	12.81 GiB	
unallocated		unallocated		1.00 MiB	---	---	

Previously to manage unit partitions, you will need to create the partition table. It can be done with the "*Device/Create partition table*" menu option. It will erase all the data on the disk.

Then you can select unallocated space and create a new partition using the toolbar or "*Partition/New* "menu option. The changes won't be done until you confirm them using the toolbar or *"Edit/Apply all operations"*.


/dev/nvme0n1 - GParted (as superuser)
GParted Edit View Device Partition Help
/dev/nvme0n1 (5.00 GiB)
New Partition #1
2.82 GiB
unallocated
2.18 GiB

Partition	File System	Size	Used	Unused	Flags
New Partition #1	ntfs	2.82 GiB	—	—	
unallocated	unallocated	2.18 GiB	—	—	

Create new Partition (as superuser)

Minimum size: 1 MiB Maximum size: 2231 MiB

Free space preceding (MiB):	0	Create as:	Primary Partition
New size (MiB):	2231	Partition name:	
Free space following (MiB):	0	File system:	ext4
Align to:	MiB	Label:	

Cancel Add

Create Primary Partition #1 (ntfs, 2.82 GiB) on /de

1 operation pending

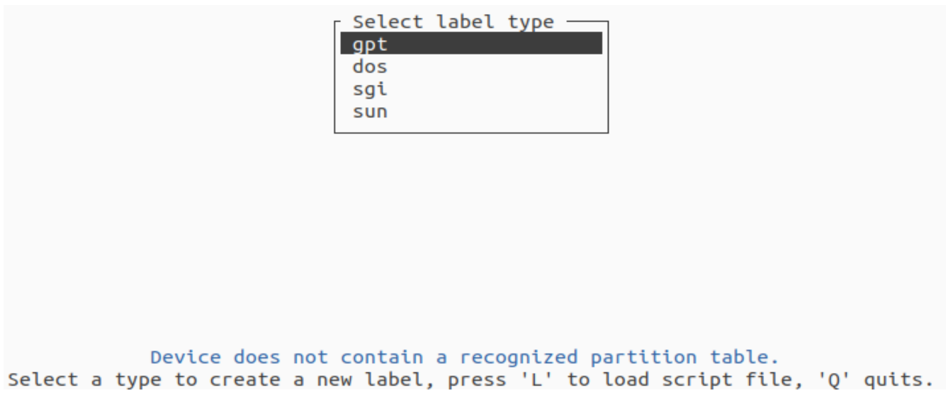
Creating a new partition. There is already a pending operation, the creation of the first partition on the disk.

3.2.2 cfdisk

`cfdisk` is a partitioning tool meant to be run from console. You can start partitioning a disk with

```
cfdisk <device>
```

Where device is the `/dev/XXXX` device corresponding to the disk. If the disk doesn't have a partition table, it will show a screen for selecting the partition table for that disk:

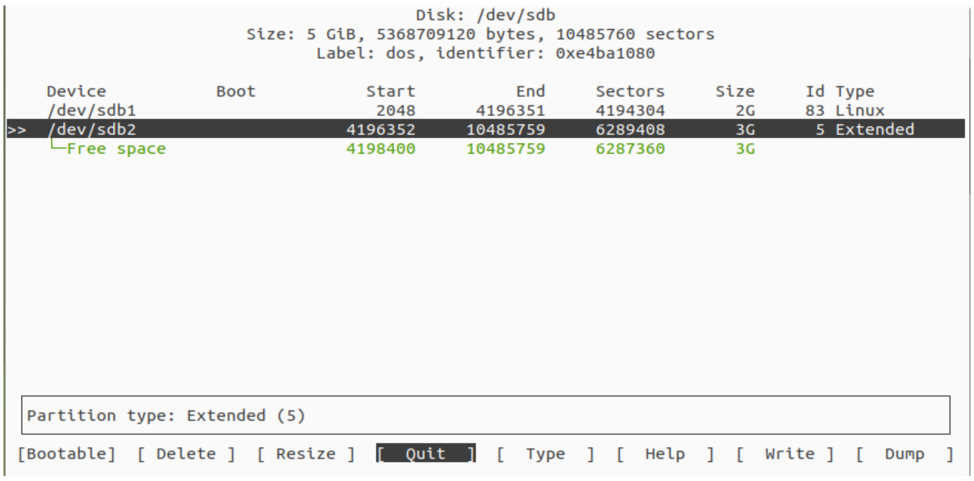


```
Select label type
gpt
dos
sgi
sun

Device does not contain a recognized partition table.
Select a type to create a new label, press 'L' to load script file, 'Q' quits.
```

If you need to rebuild the partition table using a different type, you can use another tool like `fdisk <device>` or gparted

Once the partition table is created, you can add and remove partitions using a text interface:



```
                                  Disk: /dev/sdb
                 Size: 5 GiB, 5368709120 bytes, 10485760 sectors
                        Label: dos, identifier: 0xe4ba1080

    Device            Boot          Start        End     Sectors     Size     Id Type
    /dev/sdb1                        2048    4196351     4194304       2G     83 Linux
>>  /dev/sdb2                     4196352   10485759     6289408       3G      5 Extended
      └─Free space                4198400   10485759     6287360       3G

 Partition type: Extended (5)

 [Bootable]  [ Delete ]  [ Resize ]  [  Quit  ]  [  Type  ]  [  Help  ]  [  Write ]  [  Dump  ]
```

The changes won't be written to the disk until you select the *Write* option.

It's difficult and slow to change a partition size once it's created without losing data, so choose wisely when partitioning a disk.

3.3 Formatting partitions

Once we have the partitions defined, we need to format them. By formatting a partition we create a blank file system ready to be used by the operating system. In this step is when we decide which file system that partition will have.

All data in the partition will be erased when we format it

You can format a partition in Linux with the command `mkfs`, running as root (or with `sudo` command): `mkfs -t <file system type> <device> <options>` .For example, for formatting the partition `sda1` with an ext4 filesystem you should execute:

```
sudo mkfs -t ext4 /dev/sda1
```

There are individual commands for formatting partitions with a given filesystem, like `mkfs.ext4` for formatting ext4 file systems, `mkfs.fat` for formatting FAT file systems, etc.

Each file system can be formatted with different options, depending on the file system parameters, which can help to fine-tune the file system performance. With `man mkfs.<type>` you can get information about the options for each operating system. In this course, we are leaving the default parameters when formatting a disk.

You can see all the file systems supported by a Linux computer with:

```
cat /proc/filesystems | grep -v nodev
```

In Linux you can install support for other file systems. For example, for installing exFat file system support you could execute:

```
sudo apt install exfat-fuse exfat-utils
```

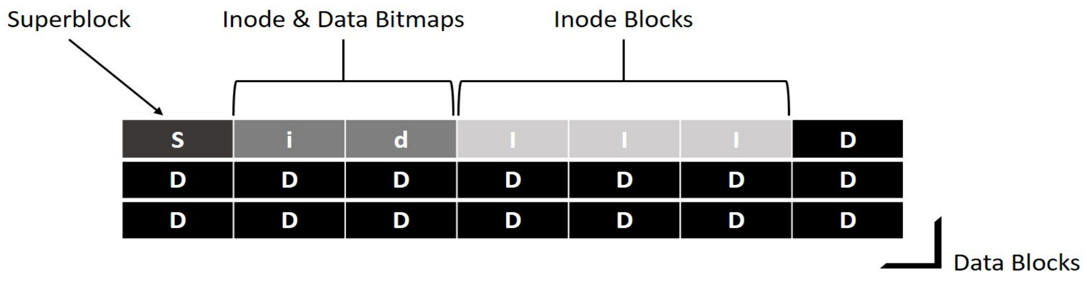
Tools like `gparted` also allow to format partitions using a graphical interface.

4. Linux filesystem

All the files in Linux are under the root directory, `/`. This is where all the system directories and files hang. Linux uses in almost all distributions the ext4 file system. That will be the one we will study in this course, although most of the commands can be applied to other file systems thanks to the virtual file system feature of Linux.

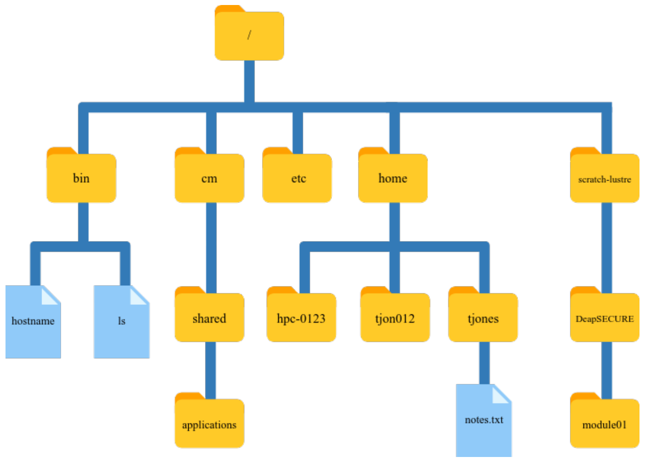
Some of the main concepts of the ext4 file system are:

- **Superblock**: it's essentially file system metadata and defines the file system type, size, status, etc. The superblock is very critical to the file system and therefore is stored in multiple redundant copies for each file system
- **Inode**: An inode is a data structure that describes a file system object, like a file or a directory. Its main task is to store the blocks that are used by a file. The inode does not contain the name of the file.
- **Inode table**: The inodes are stored in a table at the start of the disk. Before that, there is a structure where the system stores which blocks of data and inodes has been already used.
- **dentry** (directory entry): A directory entry links the name of the file with its corresponding inode. When we change the name of a file, the dentry changes, but the inode (the blocks where the file is stored) remains the same.



Birdview of an ext4 filesystem

The Linux file system is a tree of folders, starting from the root directory `/` . The directories can contain other files or directories, forming a recursive structure.



Example of a directory tree

Linux has several important subdirectories, listed below. This structure may vary in some distributions, but the scheme will be very similar.

- `/bin` contains executable command files usable by all users. Here we have the programs that can be launched by all users of the system. It can be a link to `/usr/bin` in some distributions.
- `/sbin` is for executables for exclusive use by the superuser. It can be a link to `/usr/sbin` in some distributions.
- `/home` is a directory where the personal directories of the system users are located. For example, for the user user1 her personal files will be on `/home/user1`
- `/usr` contains utilities and general user programs:
  - `/usr/bin` contains general-purpose programs.
  - `/usr/share` contains sharable, architecture-independent files.
  - `/usr/etc` contains globally usable configuration files.
  - `/usr/share/doc` contains some system documentation.
  - `/usr/share/man` contains manuals.
  - `/usr/include` contains C and C++ headers.
  - `/usr/lib` contains our program libraries.
  - `/usr/sbin` contains the system administration programs.
  - `/usr/src` contains the source code for our programs.
- `/dev` contains special block and character files associated with hardware devices. Here we find all the physical devices on the system (all our hardware) like disks (`/dev/sdX` or `/dev/nvmeXXX`), webcams (`/dev/videoN`) etc.
- `/lib` contains system libraries and compilers. It contains the libraries necessary for run the programs we have in `/bin` and `/sbin` only.
- `/proc` contains the files that receive or send information to the kernel. You should not modify the contents of this directory.
- `/etc` contains configuration and administration utility files.
- `/var` contains files for the administrator. This directory contains variable information, such as logs, server data, etc.
- `/boot` contains the system boot configuration files, such as GRUB.
- `/media` contains all the physical drives we have mounted: hard disks, DVD drives, pen drives, etc.
- `/opt` is used to support new files created after the system modification. It is a mount point from which additional application packages are installed. It can be used to install applications that do not come from the repositories, for example, those that we compiled by hand.
- `/tmp` is where temporary files are stored. It's deleted on each system boot.

4.1 Types of files

In Linux there are basically 5 types of files:

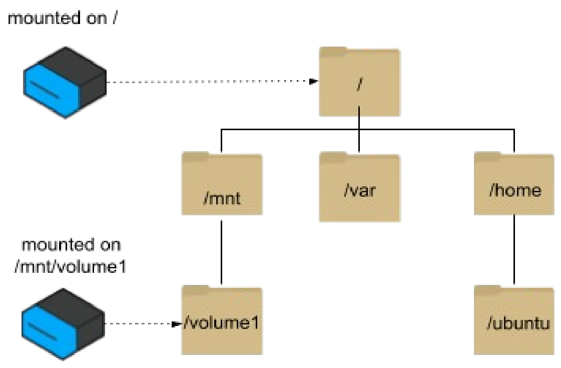
- **Ordinary files**. They contain the information that each user works with.
- **Directories**. These are special files that contain references to other files or directories.
- **Physical or hard links**. If two directory entries point to the same inode, we have a hard link. The two files will be the same, because they share the data blocks in the disk. The file won't be deleted until we remove the last directory entry of that file.
- **Symbolic links**. They are also used to assign a second name to a file. The difference with hard links is that symbolic links only refer to the name of the original file so, if we have a symbolic link and we delete the original file, we will lose the data.
- **Special files**. They usually represent physical devices, such as storage units, printers, terminals, etc. In Linux, every physical device connected to the computer is associated with a file. Linux treats special files as ordinary files. There are also socket and named pipes that are used for inter-process communication.

4.2 Mount points

One of the main features of the Linux file system is that it allows the user to create, delete and access files without needing to know exactly where they are located. In Linux, there are no physical partitions like `c:` or `d:`. All files are accesed the same way, regardless of where they are located.

It we want to use a file system in other partition, we need to use a **mount point**. A mount point is a directory (or file, but we won't be using that) at which a new file system, directory, or file is made accessible. To mount a file system or a directory, the mount point must be a directory; and to mount a file, the mount point must be a file.

Typically, a file system, directory, or file is mounted over an empty mount point, but that is not required. If the file or directory that serves as the mount point contains any data, that data is not accessible while it is mounted over by another file or directory. In effect, the mounted file or directory covers what was previously in that directory. The original directory or file that has been mounted over is accessible again once the mount over it is undone.



The root directory is mounted at the filesystem root, and another partition is mounted in /mnt/volume1

We can mount a file system using the `mount` command. It will take the filesystem type, the device and the directory. For example, for mounting `sdb1` partition with an `ext4` file system on `/second_disk` we should execute:

```
mount -t ext4 /dev/sdb1 /second_disk
```

If we don't specify the type, mount will try to find out the file system type and will act accordingly.

You can pass options to mount with the `-o` parameter. Some options are dependent on the file system. For example, to open a fat32 filesystem in read only mode:

```
mount -o ro,fat=32
```

To mount a device, you don't need to specify the device but also the partition you want to access. In other words, instead of `/dev/sdb`, you mount `/dev/sdb1` or `/dev/sdb2`, etc.

Running the mount command without parameters, or with the `-l` parameter, will display all the file systems mounted. You can filter the file system type with `-t <type>`. For example, for displaying all ext4 file systems mounted you can execute:

```
mount -t ext4
```

To unmount a file system we use the `umount` command, specifying the directory or the device. To umount `/dev/sbd1` previously mounted on `/second_disk`, we could run `umount /dev/sdb1` or `umount /second_disk`.

4.2.1 fstab

You can permanently configure where a drive attached to a machine should be mounted by editing the file `/etc/fstab`. That file is a configuration file that defines how and where the main filesystems are to be mounted, especially at boot time.

The syntax of a fstab entry is :

```
[Device] [Mount Point] [File System Type] [Options] [Dump] [Pass]
```

- **device**: The device/partition (by /dev location or UUID) that contain a file system.
- **mount point**: The directory where the partition will be mounted
- **file system type**: Type of file system
- **options**: Mount options (the same we will use for mount command)
- **dump**: Enable or disable backing up of the device/partition (the command dump). This field is usually set to 0, which disables it.
- **pass num**: Controls the order in which fsck checks the device/partition for errors at boot time. The root device should be 1. Other partitions should be 2, or 0 to disable checking. It is usually set to 0 for non root devices.

This is an example of a fstab file. The lines starting by `#` are comments are are not processed:

```
# <file system>           <mount point>  <type>   <options>                     <dump>  <pass>
/dev/sda1                 /              ext4     errors=remount-ro             0       1
UUID=B782-7AFB            /boot/efi      vfat     umask=0077                    0       1

/dev/sdb1                 /second_disk   ext4     defaults,errors=remount-ro    0       0
UUID=8dde8fbf-b7fb-...    /data          ext4     defaults,errors=remount-ro    0       0
192.168.13.140:/dir1      /network       nfs4     noauto,hard,bg,user           0       0
```

This fstab specifies the following mounts:

- The root file system (`/dev/sda1`) mounted at `/`
- A vfat file system with UUID B782... mounted at `/boot/efi`
- The partition `/dev/sdb1` mounted at `/second_disk`
- The partition with UUID 8dd... mounted at `/data`
- The network nfs filesystem `192.168.13.140:/dir1` mounted at `/network`

It’s better to use partition UUIDs instead of device names because device names can change if we modify the computer’s hardware (changing the SATA port where a disk is connected, for example, or adding new hardware) To get the partition UUID we can use the command `blkid`.

If you want to mount something that is defined in fstab, you only need to specify the directory. For example you can use `mount /second_disk` to mount `/dev/sdb1` at `/second_disk` with the options defined in fstab.

5. Managing files with linux

We will see here some of the commands used to manage files and directories in Linux.

> Linux is case-sensitive when naming files and directories. We can have a file named `John` and another file named `john` in the same directory

5.1 File system navigation

5.1.1 Current directory

From the Linux shell, we can move through the entire file system to find a specific directory or file.

The first step is to know where we are at any given moment. To find out which directory is the current directory we have the `pwd` command. Its name comes "print working directory" and its only task is to display on the screen a line that tells us the absolute path to the current directory. For example

```
$ pwd
/home/user
```

This will tell us that we are in the directory `/home/user`

5.1.2 Moving through directories

Once we know which directory we are in, we can start to scroll through the directory tree with the `cd` command. The name `cd` comes from "change directory", and it does exactly what it sounds like it does: it changes the current directory to the one we specify in its argument.

On Linux. we can refer to files and directories using absolute or relative paths

5.1.3 Absolute paths

Absolute paths are formed by starting at the root directory and ending at the point we want to refer to. For example, if our user's name is `john` and we want to make reference to his home directory, the absolute path would be `/home/john`.

> Absolute paths always start with `/`

5.1.4 Relative paths.

They are formed using the special directories `.` and `..` For example, if whe are in `/tmp/example/subdirectory` then:

- `.` refers to the current directory, `/tmp/example/subdirectory`
- `..` refers to the parent directory, `/tmp/example`

Therefore, we can use the cd command with both absolute and relative paths. For example, we are in the directory `/usr/share/man` and we want to go to `/usr/bin`. If we use the absolute path we will do:

```
cd /usr/bin
```

Or using a relative path:

```
cd ../../bin
```

> You can have more than one relative path to the same file. For example: `../bin/file` and `../bin/../bin/file` would refer to the same file

Using `cd` without parameters will change the current directory to the user's home directory.

5.2 Listing files

Once we are in a particular directory we will most likely want to know the contents of the directory. To do this we have the `ls` command. Its name comes from "list" and that is precisely what it does, it lists the contents of the directory.

If we pass it an argument, it will show us the contents of the directory we pass it and if not, it will show us the contents of the current directory. The `ls` command has a multitude of options that you can view with `man ls`

To show the directory structure we can use the `tree` command. It will display the files and directories under the working directory.

5.3 Creation

Once we have seen how to change the working directory and how to list the contents of a specific folder, we move on to see commands that will allow us to create directories and files

> You can use spaces as part of the name of a file or a directory. To do that, just put the name between quotes: `mkdir “directory with spaces”`

5.3.1 Creating directories

The `mkdir` command is used to create a folder or directory whose name will be the one we pass as an argument to this command. If, for example, we execute the command

```
$ mkdir newdirectory
```

the directory `newdirectory` will be created in the current working directory. We can also create directories in places other than the one we are in, by using absolute or relative paths. So, if we want to create the directory `Music` in `/etc` and we are in `/home/student`, we can use absolute paths:

```
$ mkdir /etc/Music
```

or relative paths:

```
$ mkdir ../../etc/Music.
```

With the `mkdir` command, we can also create several directories in the same command. To do this, we only have to enter as many arguments as folders we want to create. Thus, the command

```
$ mkdir Documents Pictures Music
```

will create those 3 directories in the current working directory.

5.3.2 Creating files

The `touch` command is used to change the access and modification date of a file that we pass as an argument. If such a file does not exist, it will create an empty file with the name that we have passed as an argument.

For example, if we execute the command

```
$ touch example.txt
```

it will create a file called `example.txt` if it did not exist before, or it will simply modify its access and modification date to the current system date, if it existed before. As with the `mkdir` command, with the `touch` command we can create multiple files with a single command:

```
$ touch fich1 fich2 fich2 fich3 fich3 fich4
```

will create the files `fich1`, `fich2`, `fich3` and `fich4`

We can also create files by redirection, as we saw in the previous unit, or by using a text editor such as `vi` or `nano`.

> If you run `vi`, you can exit the editor by typing `:` and then `q!<enter>`.

5.4 Deleting

The command rm is used to delete a file whose name will be the one that we will we pass it as an argument. For example, to delete the file named `delete_me.txt`, we will execute the command:

```
rm delete_me.txt
```

If the file we want to delete does not exist, it will show us an error. As we have seen with other commands, we can delete several files. So, if we execute `rm tom dick harry`, it will delete those three files from the current working directory.

The `rmdir` command is used to remove a directory. The directory must be empty. If not, it will show a message saying that it has contents. So, if we want to delete an empty folder that is in our current working directory we would run `rmdir the_directory`.

The `rm` command can also be used to delete entire directories. If the directory contains files and you want to delete both the directory and its contents, you will have to use the rm command with the -R option:

```
rm -r non_empty_directory
```

Be careful when using this option, as you may delete important information.

`sudo rm -fr /` won't delete the french language pack

5.5 Copying and moving

To copy files, we use the command `cp`. It copies the file that is passed to it as a parameter (source) to another one whose name is passed as a second parameter (destination). For example, if we execute the command:

```
cp file1.txt copyfile1.txt
```

it will copy the content of the file `file1.txt` into `copyfile1.txt`, both of them being in the current working directory.

On the other hand, if instead of a file, we indicate as destination a directory, what will happen is that it will create a copy of the source file in the specified destination. For example:

```
cp file2.txt /home/student
```

will create a file with the same name as the original in the directory `/home/student`.

In the `cp` command, as in the `rm` command, we have the option `R` to copy entire dire ctories with all its contents. Thus, if we execute the command:

```
 cp -R /home/student/directory1 /tmp,
```

it will copy the entire contents of the `directory1` folder to the indicated destination, and then copy the entire contents of the directory1 folder to the indicated destination, which in this case is `/tmp`.

If we want to move files from one directory to another, we can use the command `mv`. For example, we want to move the file `moveme.txt`, which is in the current directory, to the directory `/tmp`, we would execute:

```
mv moveme.txt /tmp
```

The mv command has another utility, which is to rename a file. To do this, we only have to change the name of the file we want to rename and the new name as arguments, as follows:

```
mv oldname.txt newname.txt
```

`mv` detects if the second argument is a directory. In that case, the file will be moved instead of renamed.

5.6 Visualizing

We can display text files from the console. It is also possible to display information from other types of files such as images, executables, etc. using different commands, but we are not going to deal with them in this section.

The most common orders for displaying text files are:

- `cat`: shows the contents of a text file.
- `head`: shows the first lines of a text file. For example, `head -n 3 file` will show the first three lines of the file.
- `tail`: shows the last lines of a text files. For example, `tail -n 4 file` will show the last four lines of the file
- `less`: display the content of a file but allows the user to navigate the content using page up/page down, search for content, etc.

We also have the `info` command to show us information about files. For example the character encoding, the type of the data it contains, and specific information in some cases like images or videos.

5.7 Linking

To understand how linking works, we need to understand first how we can find the data of a file in a ext4 file system.

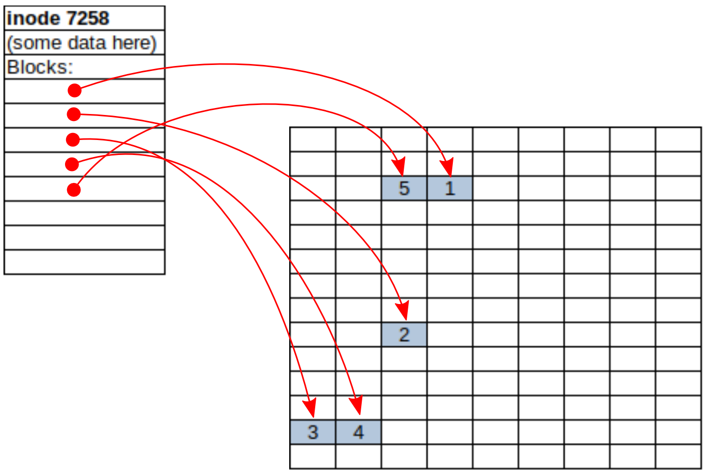
> We will see here the linking is a feature of ext4 file systems. Other file systems, like ntfs, can have different ways of linking files.

To access a file, we usually do it through a directory. The directories are composed of directory entries or dentrys. Each directory entry stores the file name and the inode containing the file data. For example, a directory dentries could be:

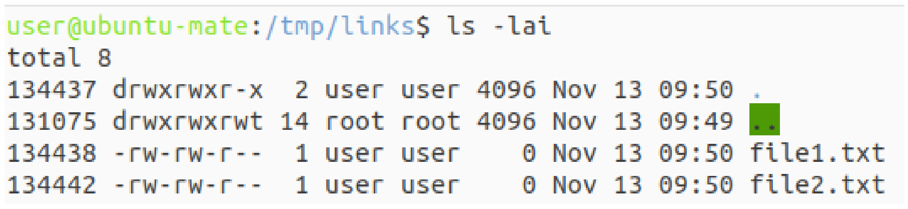
Directory	
name	***inode***
file1.txt	897
image.bmp	1225
example	7258
…	

In the above image we can see three dentrys: a file named `file1.txt` whose data is in inode 897, a file named `image.bmp` with inode 1225 and `example` with inode 7258.

The inode contains information about the file and the blocks where the file is stored on the disk. For example, the inode 7258 could be something like this:



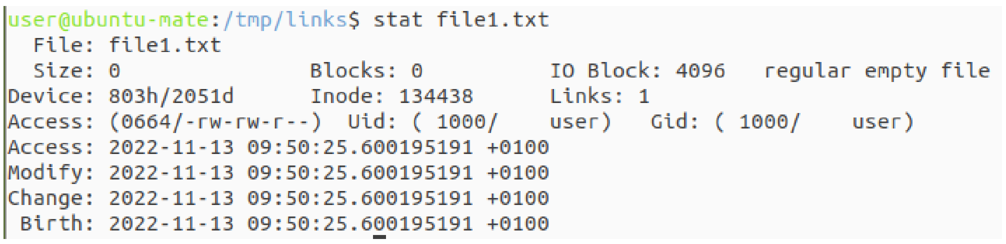
The file will use five blocks. The blocks doesn't need to be contiguous or ordered, because the correct order is stored in the inode. Note that the file name is not stored in the inode, but in the dentry. We can view the inodes of the files in a directoy using the `-i` parameter with the `ls` command:



```
user@ubuntu-mate:/tmp/links$ ls -lai
total 8
134437 drwxrwxr-x  2 user user 4096 Nov 13 09:50 .
131075 drwxrwxrwt 14 root root 4096 Nov 13 09:49 ..
134438 -rw-rw-r--  1 user user    0 Nov 13 09:50 file1.txt
134442 -rw-rw-r--  1 user user    0 Nov 13 09:50 file2.txt
```

The file `file1.txt` has the inode 134438, and `file2.txt` has the inode 13442. The directories also have inodes to store the directory entries.

We can also view the inode of a file using the stat command:



```
user@ubuntu-mate:/tmp/links$ stat file1.txt
  File: file1.txt
  Size: 0               Blocks: 0          IO Block: 4096   regular empty file
Device: 803h/2051d      Inode: 134438      Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/    user)   Gid: ( 1000/    user)
Access: 2022-11-13 09:50:25.600195191 +0100
Modify: 2022-11-13 09:50:25.600195191 +0100
Change: 2022-11-13 09:50:25.600195191 +0100
 Birth: 2022-11-13 09:50:25.600195191 +0100
```

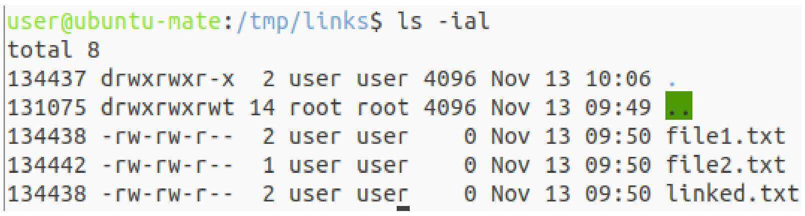
5.7.1 Hard links

A hard link is a dentry that points to the same inode of the linked file.

We can create hard links with `ln <original file> <linked file>` For example, if we execute:

```
ln file1.txt linked.txt
```

The new directory info will be:



```
user@ubuntu-mate:/tmp/links$ ls -ial
total 8
134437 drwxrwxr-x  2 user user 4096 Nov 13 10:06 .
131075 drwxrwxrwt 14 root root 4096 Nov 13 09:49 ..
134438 -rw-rw-r--  2 user user    0 Nov 13 09:50 file1.txt
134442 -rw-rw-r--  1 user user    0 Nov 13 09:50 file2.txt
134438 -rw-rw-r--  2 user user    0 Nov 13 09:50 linked.txt
```

As you can see, the inode of `file1.txt` and `linked.txt` is the same (134438). Both files point to the same blocks on the disk, so we will be modifying the same information if we access it through `file1.txt` or through `linked.txt`.

We can also see that the link count (the column before the first user) has increased. Each inode stores the number of hard links of that file. The data on the disk won't be deleted until the las file pointing to that inode is deleted.

Hard links don't need to be in the same directory, we can link any file that we want.

We can't hard link directories, only files.

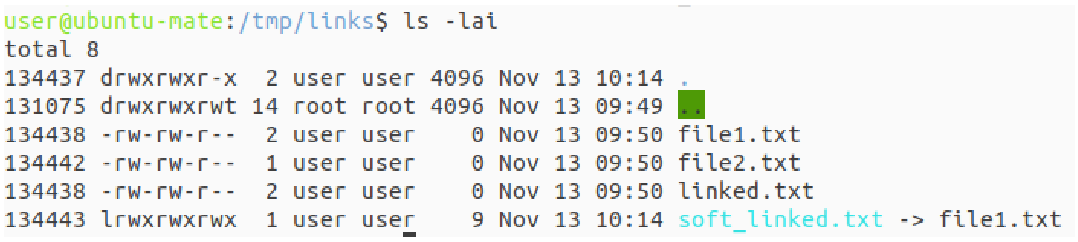
5.7.2 Soft links

Soft links work completely differently from hard links. When creating a soft link, a new "file" is created. This file stores the path to the linked file. When you try to access the link, the operating system will detect that you are accessing a linked file and redirect the request to the destination of the link.

To create a soft link we use the ln command with the `-s` parameter

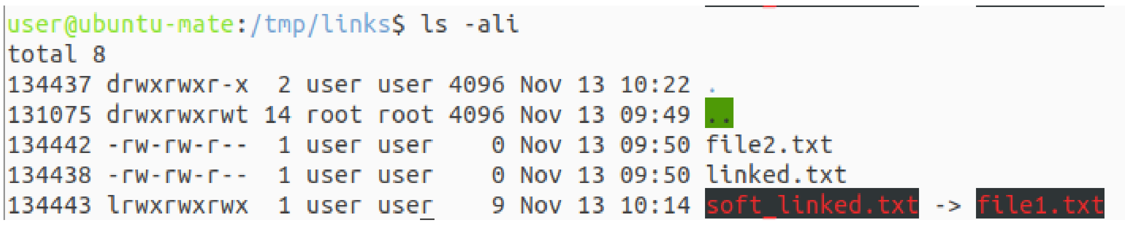
```
ln -s file1.txt soft_linked.txt
```

This will create a soft link to `file1.txt` in the same directory:



```
user@ubuntu-mate:/tmp/links$ ls -lai
total 8
134437 drwxrwxr-x  2 user user 4096 Nov 13 10:14 .
131075 drwxrwxrwt 14 root root 4096 Nov 13 09:49 ..
134438 -rw-rw-r--  2 user user    0 Nov 13 09:50 file1.txt
134442 -rw-rw-r--  1 user user    0 Nov 13 09:50 file2.txt
134438 -rw-rw-r--  2 user user    0 Nov 13 09:50 linked.txt
134443 lrwxrwxrwx  1 user user    9 Nov 13 10:14 soft_linked.txt -> file1.txt
```

As you can see, the soft link has a new inode. If we delete the link destination, the link will be "broken" and we won't be able to access the data anymore:



```
user@ubuntu-mate:/tmp/links$ ls -ali
total 8
134437 drwxrwxr-x  2 user user 4096 Nov 13 10:22 .
131075 drwxrwxrwt 14 root root 4096 Nov 13 09:49 ..
134442 -rw-rw-r--  1 user user    0 Nov 13 09:50 file2.txt
134438 -rw-rw-r--  1 user user    0 Nov 13 09:50 linked.txt
134443 lrwxrwxrwx  1 user user    9 Nov 13 10:14 soft_linked.txt -> file1.txt
```

Note that the same data is still available through `linked.txt` (it was a hard link of `file1.txt`) but the soft link points to `file1.txt` and that file doesn't exist anymore, so the link is broken.

We can soft link directories, unlike hard links.

6. Virtual File System and non-disk file systems

For accessing the files, Linux implements the Virtual File System (also known as VFS). It's a component of the kernel that handles all system calls related to files and file systems. VFS is a generic interface between the user and a particular file system. This means that we can access different file systems in a uniform way used the API provided by the VFS. The communication with the hardware and the I/O are handled by VFS.

That means that we can create file systems not supported by disk devices, as long as we provide the API calls that the VFS require. For example, we can create file systems that are stored in memory or accessed through the network.

From a functional point of view, file systems can be grouped into:

- disk file systems (ext3, ext4, xfs, fat, ntfs, etc.)
- network file systems (nfs, smbfs/cifs, ncp, etc.)
- virtual file systems (procfs, sysfs, sockfs, pipefs, etc.)

Network file systems are “disks” that some other computer makes available through the network. There are different network file systems like nfs or samba. Anyway, thanks to the VFS we can mount and use them as if the files were in our own computer. We will see this later in the course.

Virtual file systems are file systems where the data is not stored into a disk. For example we can create a RAM disk (a disk where the data is stored in computer's RAM) with this command:

```
sudo mount -t tmpfs -o size=512M myramdisk /tmp/ramdisk
```

where the device name will be myramdisk and it will be mounted in /tmp/ramdisk. It will take memory from the computer memory and the data will be lost when the computer is powered off, but it will be extremly fast.

7. Supplementary material

File systems:

- https://www.howtogeek.com/196051/htg-explains-what-is-a-file-system-and-why-are-there-so-many-of-them/

- https://kb.wisc.edu/helpdesk/page.php?id=11300

MBR and GPT

- https://www.softwaretestinghelp.com/mbr-vs-gpt/

Disk partitions

- https://docs.fedoraproject.org/en-US/fedora/latest/install-guide/appendixes/Disk_Partitions/

Linux inodes:

- https://www.youtube.com/watch?v=qXVbNlMG28I
- https://www.youtube.com/watch?v=6KjMlm8hhFA

Hard Links vs Soft links:

- https://www.youtube.com/watch?v=4-vye3QFTFo