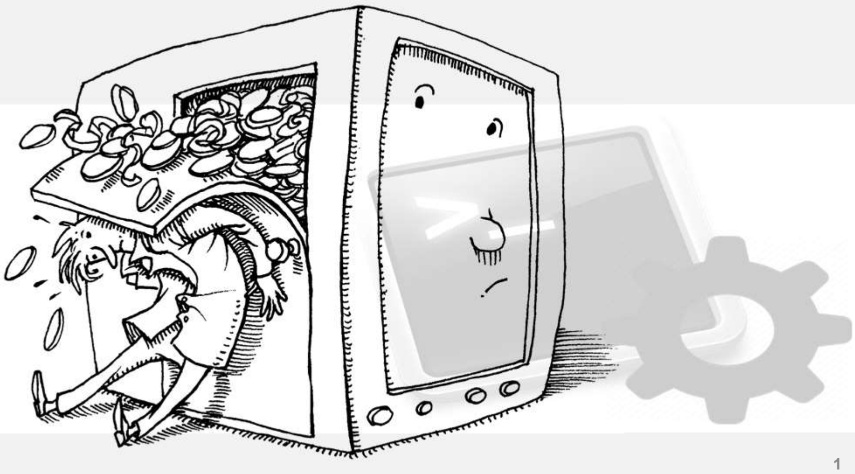
File Systems (Storage)



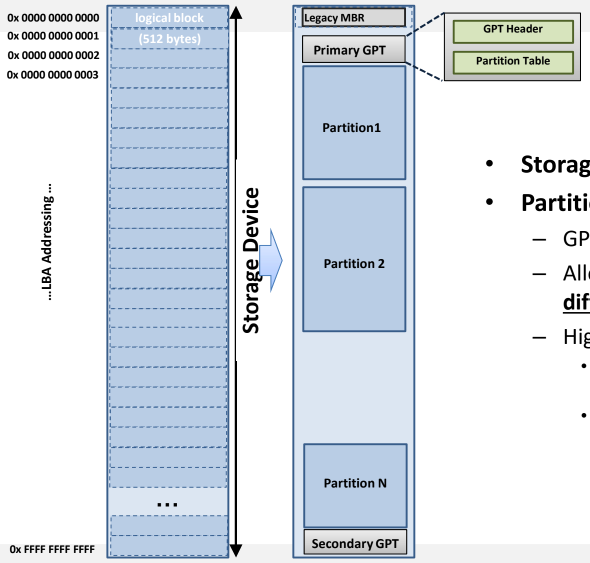
Index

- **Introduction**
  - File System essentials
  - EXT basic features
  - The Linux Filesystem: Logic structure & organization, File permissions.
- **Adding a new Storage Device**
  - Identify the Disk & Create Partitions
  - Create Filesystem
  - Create mountpoint
- **File System Consistency**
  - Journaling
  - Checking Filesystem integrity
- **Managing Filesystem**
  - Tunning Filesystem
  - Resizing Filesystem
- **File System Security (disk encryption)**

File System Definition

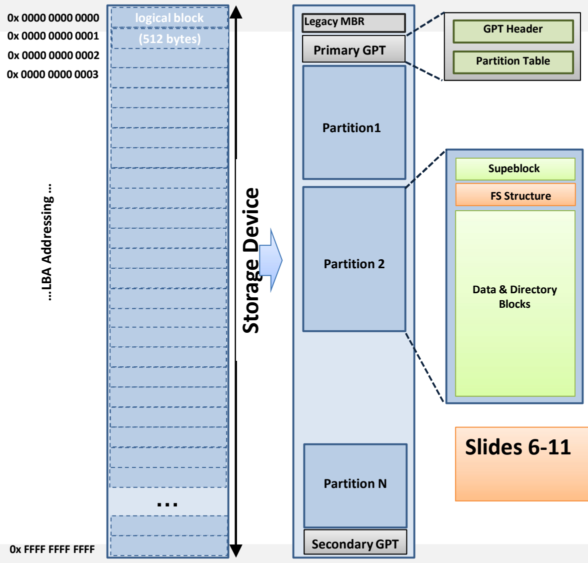
- Mechanism to represent and organize the system’s storage
  - From a linear array of sectors (disk) to the directory hierarchy (root filesystem).
- In Linux, many objects (not only files) are managed through a file system
  - I/O devices such as mouse, speakers, keyboard
  - Kernel data structures and some configuration parameters
  - Interprocess communication channels
  - etc.
- This chapter is focused on **disk-based filesystems**

Elements of a Storage System



- **Storage Device**: Hard disk, flash drive, SSD, etc.
- **Partition**: Fixed-size subsection of a storage device
  - GPT: Disk partition standard (more info in Booting chapter)
  - Allows to treat a single physical device as multiple ones, with **<u>a different File system on each partition.</u>**
  - High utility for administration tasks:
    - Protect directories that tend to grow indefinitely: /var/spool against “mailbombing” or /tmp against careless users/apps
    - Divide software and users: Easier upgrading, avoid users blocking the system

Elements of a Storage System


0x 0000 0000 0000

0x 0000 0000 0001

0x 0000 0000 0002

0x 0000 0000 0003

...LBA Addressing ...

0x FFFF FFFF FFFF

logical block (512 bytes)

...

Storage Device

Legacy MBR

Primary GPT

GPT Header

Partition Table

Partition1

Partition 2

Partition N

Secondary GPT

Supeblock

FS Structure

Data & Directory Blocks

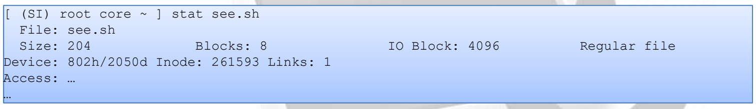
- **File System**: Mediation between the raw blocks and the standard interface expected by programs (paths, filenames, permissions, etc.)
  - Disk performs sequential storage (sector), does not know about hierarchies.
  - Main Requirements: Labeled files, file organization as a linked hierarchy (directories), Metadata for every file (generation time, permissions, etc.)
  - Superblock + FS Structure + Raw Data
- **Superblock**: Keeps information about file system (version, boot file location, number of blocks, first block of / directory,…)
  - Read during file system mounting.

Slides 6-11

- **FS Structure**: Mapping structure of files/directories into blocks (different for each file system)
- **Raw Data:** The actual file content

EXT File System

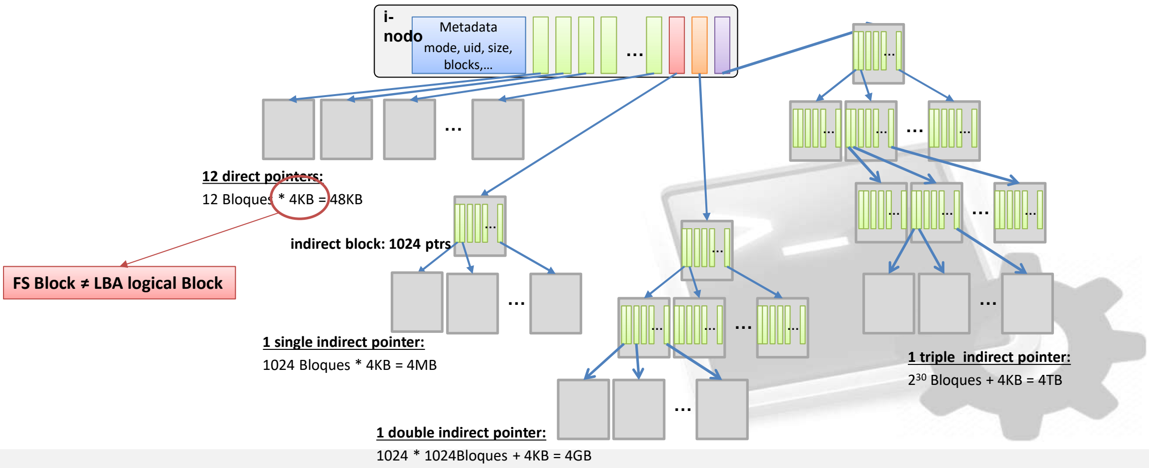
- ext: first of a series of linux-exclusive file systems
- The **i-nodes**
  - Basic building element of the ext file system. Stores **metadata** and the **data blocks** of a file (or dir)
  - Each file (or directory) has associated one i-node.
  - By default, they consume a 10% of disk storage (can be configured at FS creation time).
  - How to check i-node information in linux:
    - stat command: (second & third line)
    - ls -il



```
[ (SI) root core ~ ] stat see.sh
  File: see.sh
  Size: 204             Blocks: 8               IO Block: 4096          Regular file
Device: 802h/2050d Inode: 261593 Links: 1
Access: …
…
```

EXT File System

- The i-nodes



12 direct pointers:
12 Bloques * 4KB = 48KB

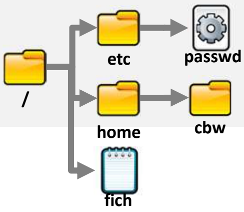
FS Block ≠ LBA logical Block

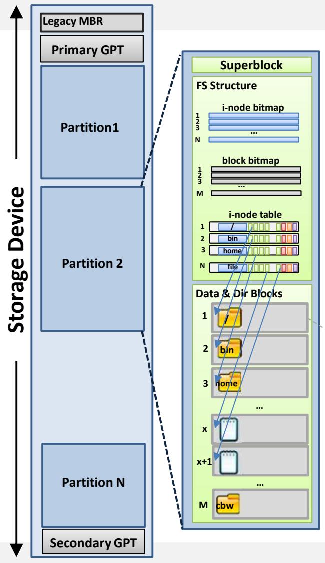
1 single indirect pointer:
1024 Bloques * 4KB = 4MB

1 double indirect pointer:
1024 * 1024Bloques + 4KB = 4GB

1 triple indirect pointer:
2^{30} Bloques + 4KB = 4TB

EXT File System

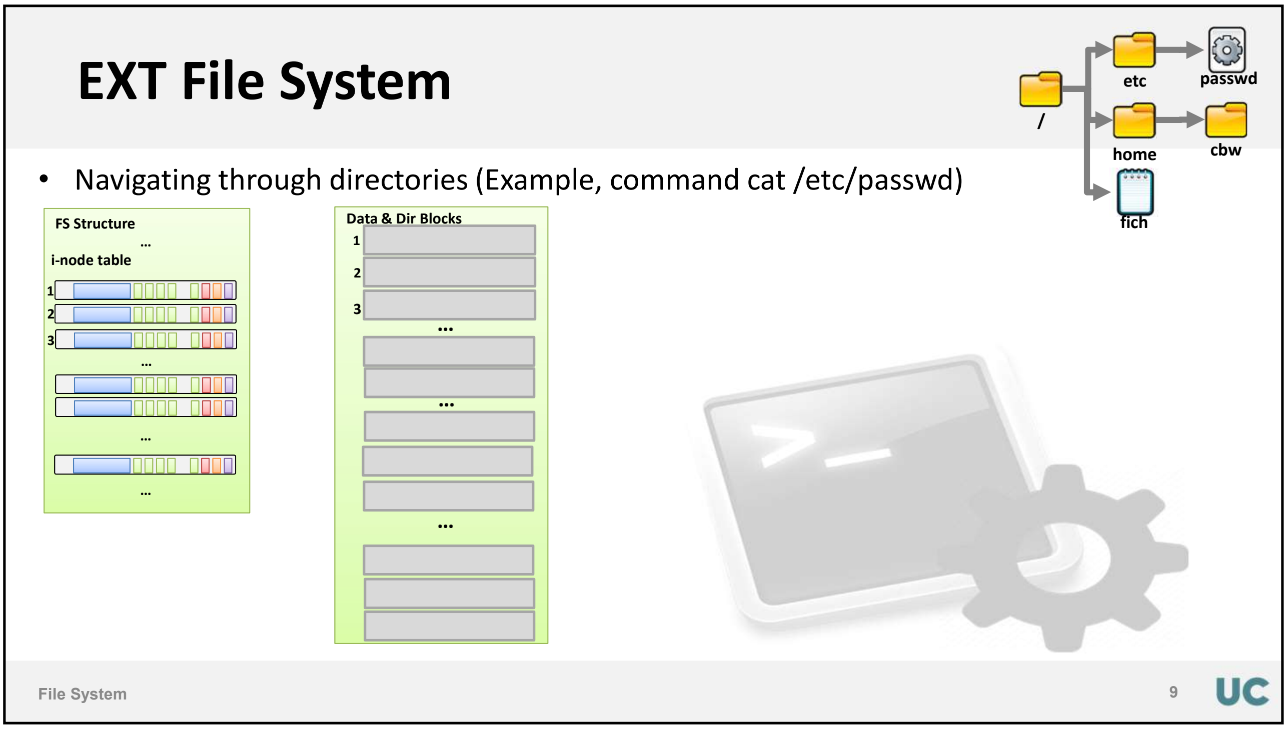




- EXT File System structure
  - **i-node bitmap**: bit map of occupied/free inodes.
  - **block bitmap**: bit map of occupied/free blocks.
  - **I-node table**: each input is a single i-node.

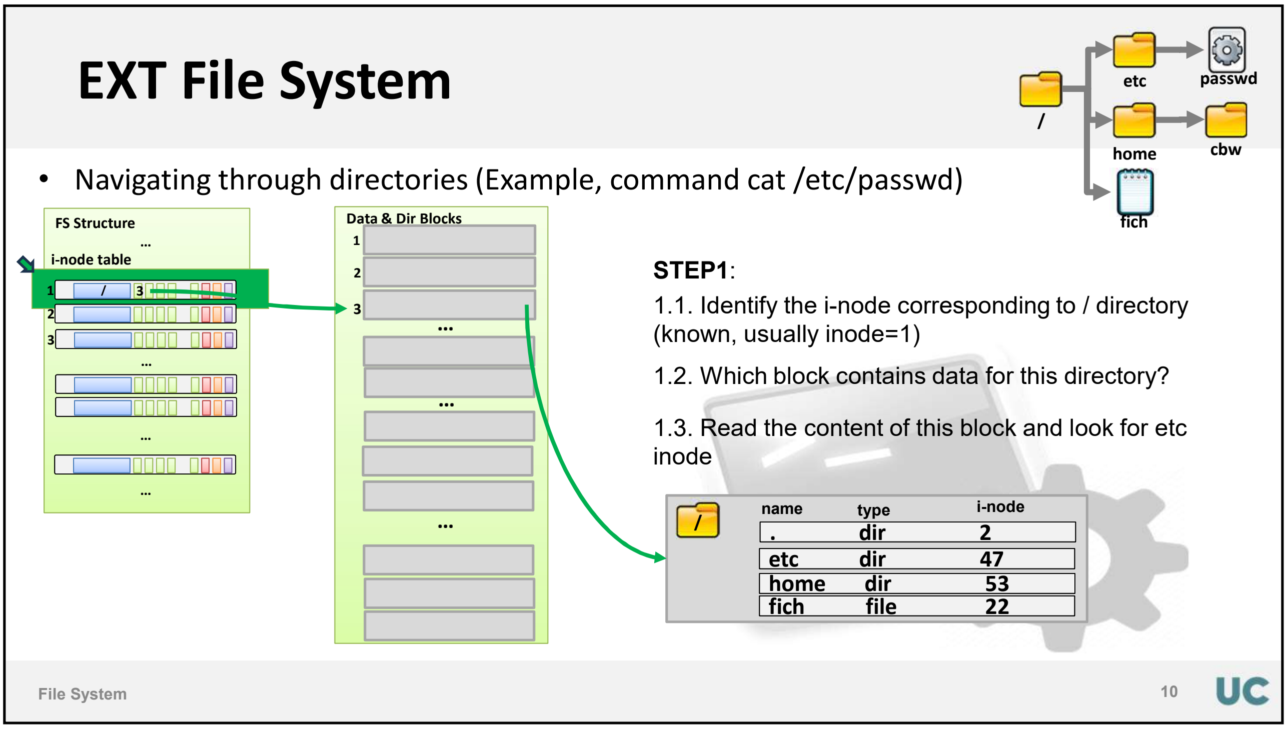
/

name	type	i-node
.	dir	2
etc	dir	47
home	dir	53
fich	file	22



EXT File System

- Navigating through directories (Example, command cat /etc/passwd)



EXT File System

- Navigating through directories (Example, command cat /etc/passwd)

STEP1:

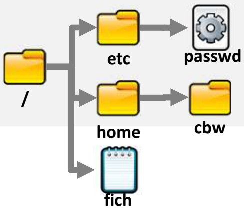
1.1. Identify the i-node corresponding to / directory (known, usually inode=1)

1.2. Which block contains data for this directory?

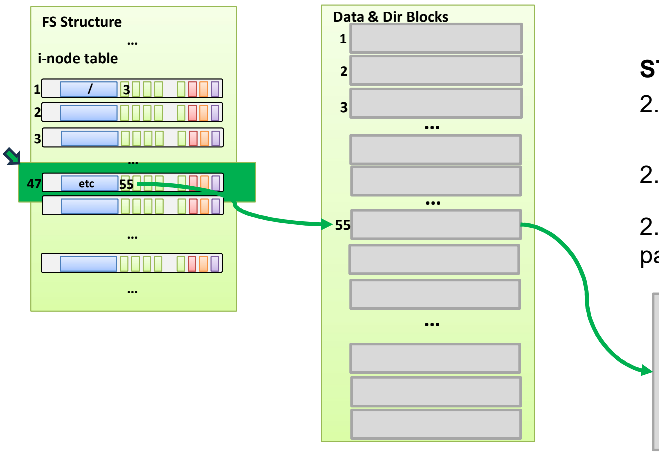
1.3. Read the content of this block and look for etc inode

name	type	i-node
.	dir	2
etc	dir	47
home	dir	53
fich	file	22

EXT File System



- Navigating through directories (Example, command cat /etc/passwd)



STEP2:

2.1. Read inode 47, corresponding to etc.

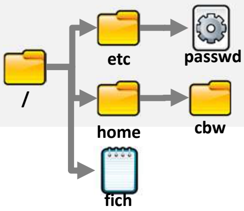
2.2. Which block contains data for this directory?

2.3. Read the content of this block and look for passwd inode

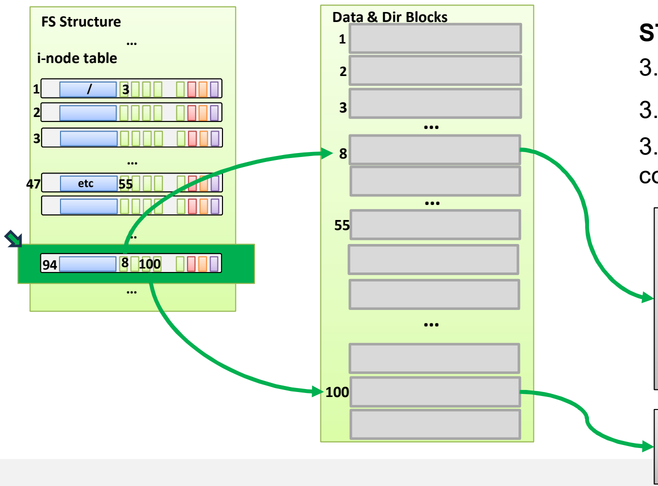
etc

name	type	i-node
.	dir	47
passwd	file	94
...		
ldap	dir	153
...		
fstab	file	102

EXT File System



- Navigating through directories (Example, command cat /etc/passwd)



STEP3:

3.1. Read inode 94, corresponding to passwd.

3.2. Which block contains data for this file?

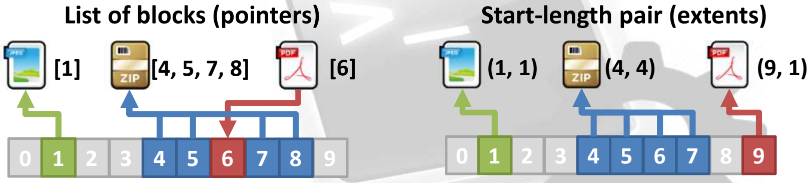
3.3. Read the content of these blocks, which correspond to the content of the file

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:
daemon:x:2:2:daemon:/sbin:
adm:x:3:4:adm:/var/adm:
lp:x:4:7:lp:/var/spool/lpd:
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
…
```

```
test:x:500:500:Usuario test:/home/test:/bin/bash
apache:x:48:48:Apache:/var/www:/bin/false
…
```

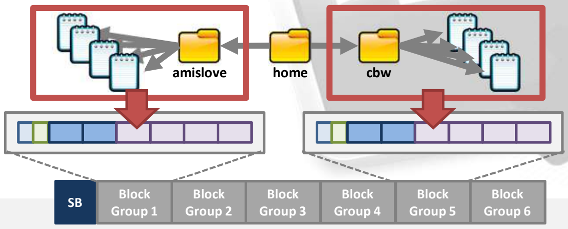
EXT File System

- Pointers (ext1-3) vs **Extents**:
  - Inode pointers are not efficient for big files
    - Example: a 100MB file requires 25600 pointers.
    - Cannot be avoided if no contiguous blocks, but what happens in the presence of locality?
  - Current file systems try to minimize data fragmentation
    - Less searches, better performance
    - Extents behave better in the case of files with adjacent blocks



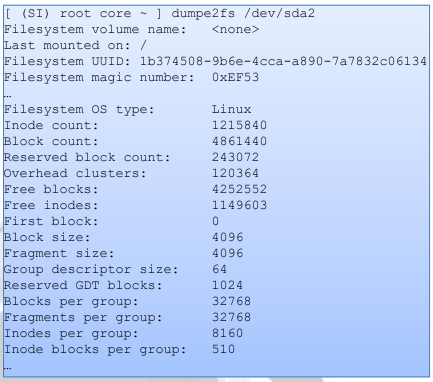
EXT File System(ext2)

- Problems/Limitations of EXT:
  - Data fragmentation: i-nodes and their associated data can be far away in the disk (performance problems).
- ext2 improves data-metadata locality:
  - Disk is divided into block groups (group size usually depends on disk physical properties: cylinder size).
  - Each group replicates FS structures: inode/data bitmap, inode table.



Exploring Filesystem Structures

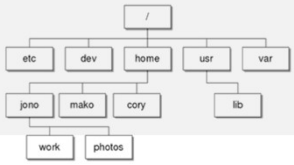
- Command **dumpe2fs:** dump filesystem information
  - – prints the super block and blocks group information for the filesystemq.
- Command **debugfs**: a debugger for FS metadata
  - – Interactive, used to examine and change the state of the file system (debugfs /dev/sda1)
    - You can use some equivalent shell commands to interact with the FS: cat, cd, chroot, ls, mkdir, pwd, rm, rmdir, stat
    - Inspect i-node content: *dump <inode num> out-file*
    - Recover an accidentally removed file (complex process*)*



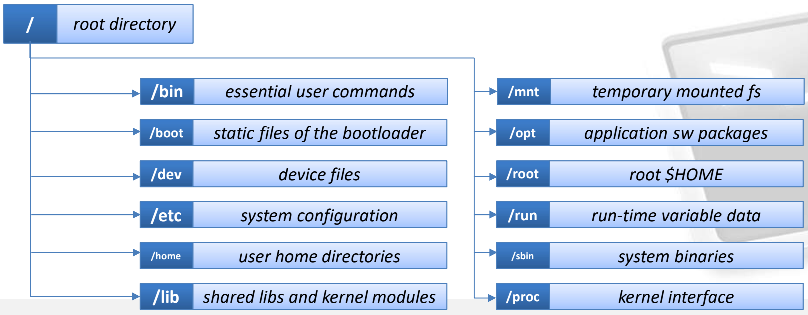
```
[ (SI) root core ~ ] dumpe2fs /dev/sda2
Filesystem volume name:   <none>
Last mounted on: /
Filesystem UUID: 1b374508-9b6e-4cca-a890-7a7832c06134
Filesystem magic number:  0xEF53
…
Filesystem OS type:       Linux
Inode count:              1215840
Block count:              4861440
Reserved block count:     243072
Overhead clusters:        120364
Free blocks:              4252552
Free inodes:              1149603
First block:              0
Block size:               4096
Fragment size:            4096
Group descriptor size:    64
Reserved GDT blocks:      1024
Blocks per group:         32768
Fragments per group:      32768
Inodes per group:         8160
Inode blocks per group:   510
…
```

Both commands (and more in this chapter) are part of the **e2fsprogs** package, which contains many utilities for ext FS management.

The Linux <u>root</u> Filesystem



- Minimal contents to **boot**, restore, recover and/or repair the system
  - – Any other FS must be mounted in a folder inside root filesystem (see mount slides).
- **Tree-like Hierarchical** structure.
  - – Folders separated by /
  - – File access (path): Absolute (cd /home/pep) or Relative to current path (with “.” o “..”): cd ../../../usr/local
  - – Files starting with “.” are hidden.



/ *root directory*

/bin *essential user commands*

/boot *static files of the bootloader*

/dev *device files*

/etc *system configuration*

/home *user home directories*

/lib *shared libs and kernel modules*

/mnt *temporary mounted fs*

/opt *application sw packages*

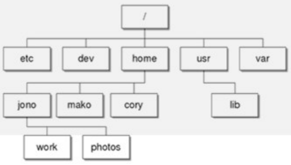
/root *root $HOME*

/run *run-time variable data*

/sbin *system binaries*

/proc *kernel interface*

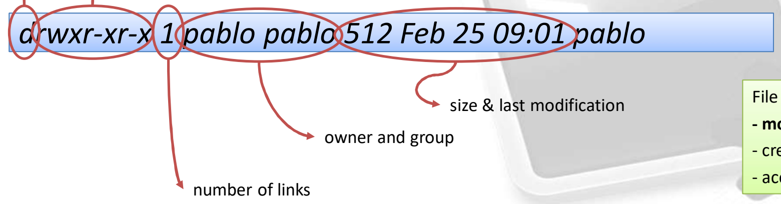
The Linux root Filesystem



- **File Attributes**
  - [ls -l] for a file, [ls –ld] for a directory

File type	Symbol
Regular file	-
Directory	d
Character device file	c
Block device file	b
Local domain socket	s
Named pipe	p
Symbolic link	l

- **Permission bits**. Three sets that define access for the owner, group and everyone else.
- Controlled with **chmod** command (umask for default permissions on file creation).
- **setuid/setgid bit**: [-rw**s**r-**s**r-x] when set on executable file, allow program to access files that are off-limits to the user that runs it. chmod 4xxx / chmod 2xxx
- **sticky bit**: [drwxr-xr-**t**] you are not allowed to delete or rename a file inside dir unless you are dir owner, file owner or superuser. chmod 1xxx



drwxr-xr-x 1 pablo pablo 512 Feb 25 09:01 pablo

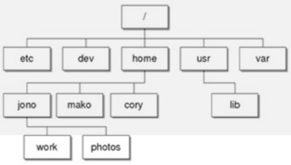
size & last modification

owner and group

number of links

File **timing** (3 flavours):
- **modification (--time=ctime)**
- creation (--time=creation)
- access (--time=atime)

The Linux root Filesystem

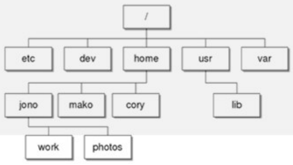


- **The Linux bonus flags**
  - supplemental flags to request special handling
  - Two commands to view and change file attributes: **lsattr** and **chattr**
  - Available flags in ext3 and ext4 (man chattr for a complete list):

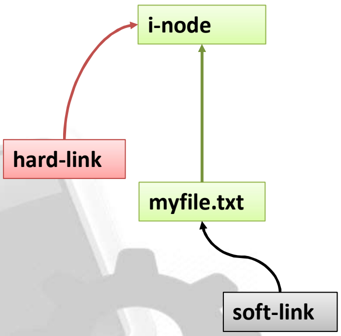
Flag	Meaning
A	Never update access time (performance)
a	Allow writing only in append mode
D	Force directory updates to be written synchronously
d	Do not backup (backup utilities)
i	Make file immutable and undeletable
j	Keep a journal for data changes as well as metadata
S	Force changes to be written synchronously (no buffering)

- **(FYI) Access control lists**
  - More powerful way of regulating file access.
  - Out of the scope of this course…

The Linux root Filesystem



- **Links**: Special files that point to other locations in the Filesytem.
- **Hard Link:** create an additional file with a link to an existing inode
  - Multiple files sharing the same i-node.
  - If you delete one file, the content is still available through the created hard links.
  - Only valid for files in the same FS.
- **Soft Link**: link to another name in the file system.
  - resolves the name of the file each time you access it through the symlink.
  - If the original copy is deleted, the symlink wont work
  - works with directories and within multiple FS.
- Command: ln
  - Syntax: ln [-s] <original-file> <created-link>



Index

- Introduction
  - File System essentials
  - EXT basic features
  - The Linux Filesystem: Logic structure & organization, File permissions.
- **Adding a new Storage Device**
  - Identify the Disk & Create Partitions
  - Create Filesystem
  - Create mountpoint
- File System Consistency
  - Checking Filesystem integrity
  - Journaling
- Managing Filesystem
  - Tunning Filesystem
  - Resizing Filesystem
- File System Security (disk encryption)

Block Device Naming

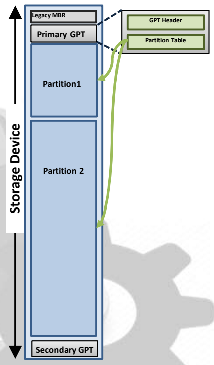
- Device file (/dev): communication between hw driver and OS.
- Linux Naming Convention (for storage devices):
  - In **SCSI devices (and PCI)** same naming convention, “sd” + a character (starting with”a”)
    - /dev/**sda, /dev/sdb, …**
  - In a disk, each **partition** is identified with a number.
    - /dev/**sda1**: first partition of the sda disk.
- Important: Persistent naming.
  - Associated sd names are allocated for each device when it is detected. If the order of device detection changes, associated sd names can change. This may result in device names like /dev/sda and /dev/sdb switching arround on each boot.
- Persistent naming solves these issues.
  - In /dev/disk/by-XXX, links to the corresponding /dev/sdXX
  - **XXX=uuid:** unique file system identifier generated by mkfs
  - XXX=id: (subsystem string) + unique number depending on HW serial num.
  - XXX=path: (subsystem string) + unique number depending on phys. path
  - XXX=partuuid: gpt partition UUID (defined in GPT header)

Weaker persistency: plugging device into a different port changes the name.

Legacy problem: not supported by MBR.

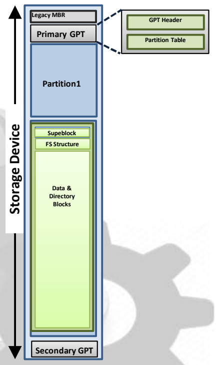
Disk Verification & Partitions

- After installing a new disk, check to make sure that the system acknowledges its existence.
  - Command **lsblk**: print a list of the disks that the system is aware of.
    - # lsblk -o +MODEL,SERIAL
- **Disk Partitioning**: GUID partition table (GPT)
  - <u>Warning: </u>some disk management utilities lack support for GPT disks.
  - Command **gdisk**: manipulation of the partition table
    - Syntax: gdisk /dev/sda (Includes a descritive menu of the available operations [m])
    - Think carefully what you are doing ([q] exit without saving changes)
    - [v]: look at the content of a unpartitioned disk.
    - [n]: new partition
    - **[w]: Write the new partition table** (Prior revision with [p])
    - gdisk -l /dev/sda to list current disk partitions



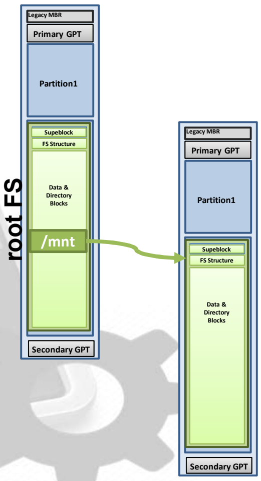
Filesystem creation

- One filesystem per partition
- File systems supported by the kernel: ls /lib/modules/$(uname -r)/kernel/fs
  - Most recommended in linux is: ext3/ext4
- File systems already being used: /proc/filesystem
- Command **mkfs**: builds a file system in a partition.
  - Syntax: mkfs [-V –t fs-type] /dev/sda3
  - default options available in **/etc/mke2fs.conf**



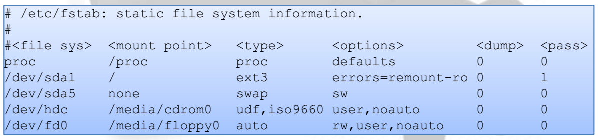
Filesystem mounting

- **mounting** process provides access to the content of a disk from the root FS
  - Can be done for any storage device (USB, CDROM, tape,…)
  - At least one partition (system) is mounted during booting process.
  - Command **mount**: (Syntax: mount -<options> [file-dev] [mnt-point])
    - Option –r: mounting in read-only mode
    - Option –t: kind of file system mounted
    - Example: mount –t ext3 /dev/hdc1 /home/
- Command **df** to list mounted filesystems
- **Umount** process disconnects the device from the rest of the system.
  - Command **umount** (syntax: umount [mnt-point])
    - Doing this requires that no process is making use of the file system to umount
  - Checking use if umount fails:
    - **fuser** -vm <mnt-dir> : lists which processes access the mount point
    - **lsof** <mnt-dir>: show each open file on the mount point.



Automatic mount/umount

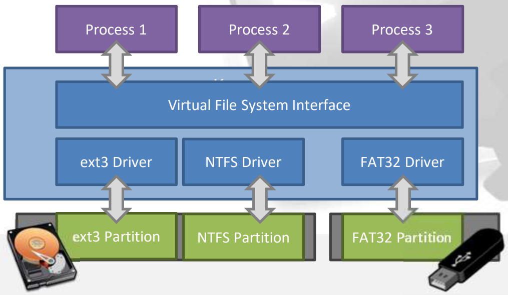
- Systems to mount/umount are read from file **/etc/fstab**.
- Done automatically during boot process (can be also performed at a different moment with command “mount –a”)
- File /etc/fstab:
  - **<file sys>**: Device file (any block device naming)
  - **<mount point>**: mount point (directory)
  - **<type>**: type of file system (ext3, ext4, vfat, xfs,...)
  - **<options>**: Read or Read/write mode (ro/rw), SUID/SGID support (suid/nosuid), allow user mounting (user/nouser), allow binary execution (exec/noexec),…
  - **<dump>**: dump frequency (backup utility, obsolete)
  - **<pass>**: order to run fsck on the device.



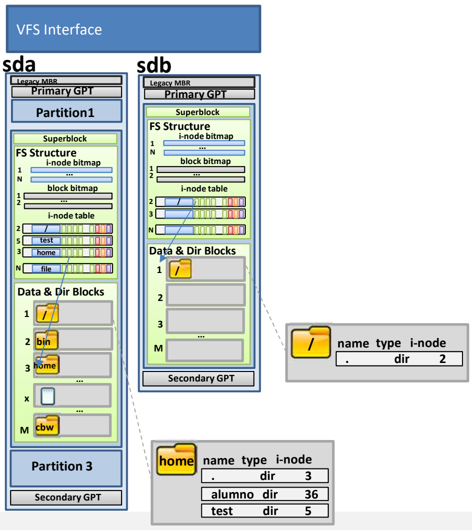
```
# /etc/fstab: static file system information.
#
#<file sys>  <mount point>   <type>        <options>          <dump>  <pass>
proc         /proc           proc          defaults           0       0
/dev/sda1    /               ext3          errors=remount-ro  0       1
/dev/sda5    none            swap          sw                 0       0
/dev/hdc     /media/cdrom0   udf,iso9660   user,noauto        0       0
/dev/fd0     /media/floppy0  auto          rw,user,noauto     0       0
```

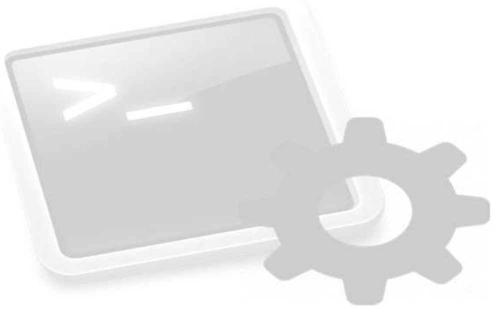
Managing multiple Filesystems

- Problem:
  - The OS can mount multiple partitions with different file systems.
  - Does a process need to use different APIs for each FS?
- Linux makes use of an interface known as Virtual File System (VFS)
  - Exposes a POSIX API to the processes.
  - re-sends requests to the specific driver of the underlying file system.

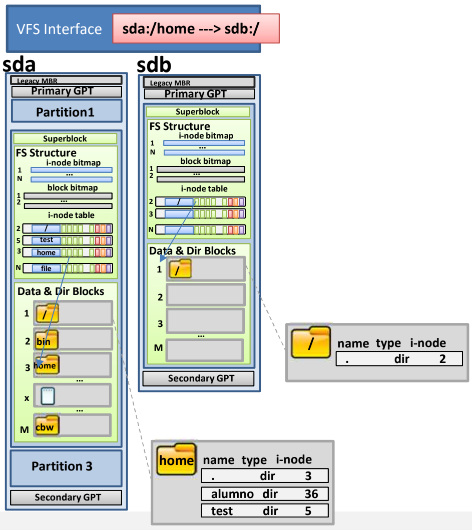


VFS & Mounting



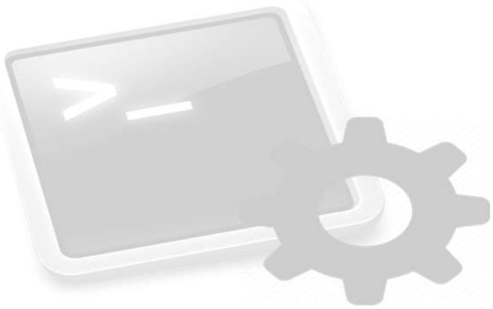


VFS & Mounting

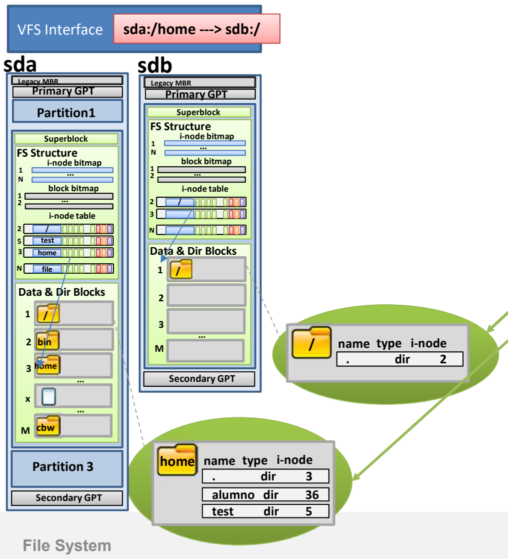


1.- [(SI)] mount /dev/sdb1 /home

- La asociación se define en estructuras internas del VFS
- El FS Structure de los dos sistemas de ficheros se mantiene intacto



VFS & Mounting



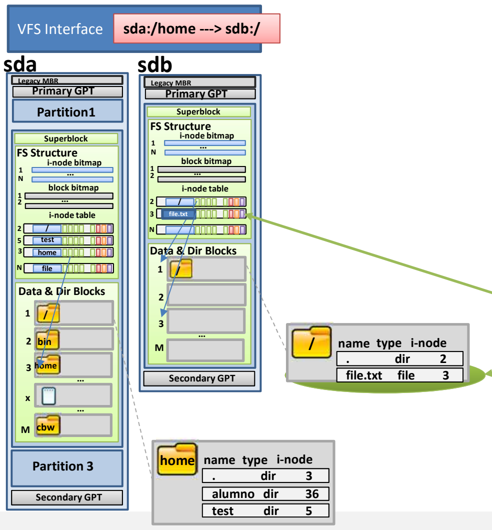
1.- [(SI)] mount /dev/sdb1 /home

- La asociación se define en estructuras internas del VFS
- El FS Structure de los dos sistemas de ficheros se mantiene intacto

2.- [(SI)] ls /home

- Se resuelve el path en el VFS, se muestra el contenido de la partición de sdb
- Los contenidos previos al montaje ya no son accesibles, pero siguen presentes en sda (y no se borrarán)

VFS & Mounting



1.- [(SI)] mount /dev/sdb1 /home

- La asociación se define en estructuras internas del VFS
- El FS Structure de los dos sistemas de ficheros se mantiene intacto

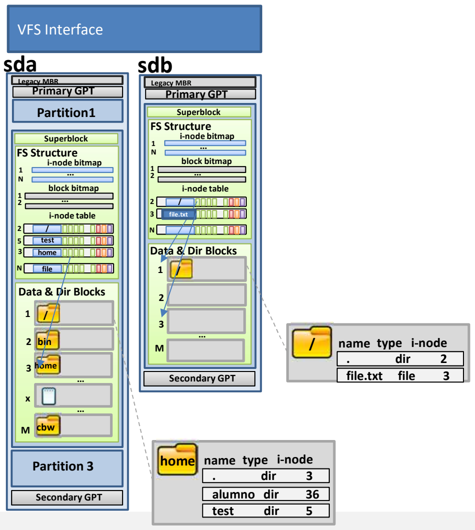
2.- [(SI)] ls /home

- Se resuelve el path en el VFS, se muestra el contenido de la partición de sdb
- Los contenidos previos al montaje ya no son accesibles, pero siguen presentes en sda (y no se borrarán)

3.- [(SI)] touch /home/file.txt

- De nuevo, VFS resuelve el path.
- Las estructuras que se actualizan (tabla inodos) son las de sdb.

VFS & Mounting



name	type	i-node
.	dir	2
file.txt	file	3

name	type	i-node
.	dir	3
alumno	dir	36
test	dir	5

1.- [(SI)] mount /dev/sdb1 /home

- La asociación se define en estructuras internas del VFS
- El FS Structure de los dos sistemas de ficheros se mantiene intacto

2.- [(SI)] ls /home

- Se resuelve el path en el VFS, se muestra el contenido de la partición de sdb
- Los contenidos previos al montaje ya no son accesibles, pero siguen presentes en sda (y no se borrarán)

3.- [(SI)] touch /home/file.txt

- De nuevo, VFS resuelve el path.
- Las estructuras que se actualizan (tabla inodos) son las de sdb.

4.- [(SI)] umount /home

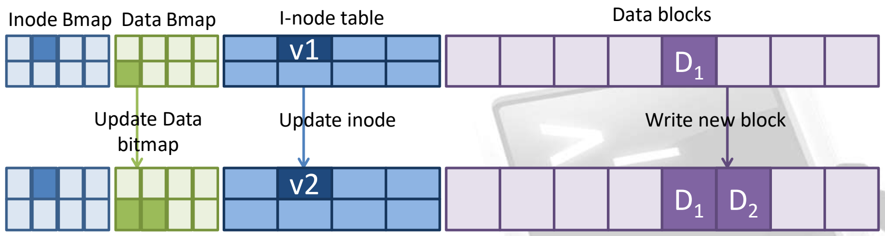
- Retornamos a la situación de origen.
- Contenido visible en /home: el de sda (alumno, test)
- El nuevo contenido (file.txt) permanece en sdb, pero para acceder al mismo debemos habilitar un nuevo punto de montaje.

Index

- Introduction
  - File System essentials
  - EXT basic features
  - The Linux Filesystem: Logic structure & organization, File permissions.
- Adding a new Storage Device
  - Identify the Disk & Create Partitions
  - Create Filesystem
  - Create mountpoint
- File System Consistency
  - Journaling
  - Checking Filesystem integrity
- Managing Filesystem
  - Tunning Filesystem
  - Resizing Filesystem
- File System Security (disk encryption)

EXT File System (ext3)

- Consistency of the file system:
  - Some operations require multiple and independent write operations in the file system.
  - Example: Add a block to an existing file (size increase).



Inode Bmap Data Bmap I-node table Data blocks

v1 D_1

Update Data bitmap

Update inode

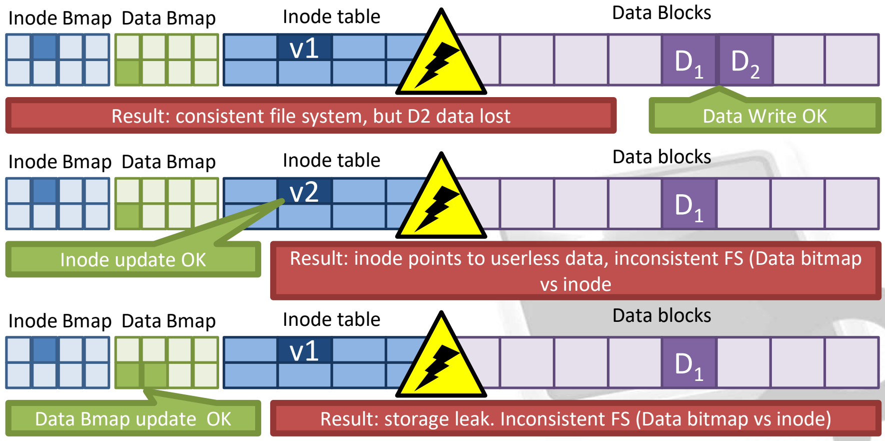
Write new block

v2 D_1 D_2

- Operations performed in random order
  - What happens if the process is interrupted at an intermediate point?

EXT File System (ext3)

- Consistency of the file system:



Inode Bmap Data Bmap Inode table Data Blocks

v1 D_1 D_2

Result: consistent file system, but D2 data lost

Data Write OK

Inode Bmap Data Bmap Inode table Data blocks

v2 D_1

Inode update OK

Result: inode points to userless data, inconsistent FS (Data bitmap vs inode

Inode Bmap Data Bmap Inode table Data blocks

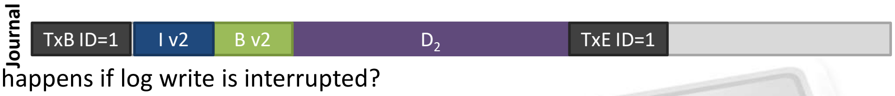
v1 D_1

Data Bmap update OK

Result: storage leak. Inconsistent FS (Data bitmap vs inode)

EXT File System (ext3)

- Journaling:
  - Atomic pre-writing (at the same time) disk data.
  - Disk writes are pre-annotated in a log. Each input: journal.



Journal

TxB ID=1 | I v2 | B v2 | D_2 | TxE ID=1

  - What happens if log write is interrupted?
    - Transaction is not completed (data lost) but the FS remains consistent.
  - What happens if journal is written correctly, but disk not?
    - Temporally, file system misses consistency.
    - The log has the information to restore it (during boot, unfinished journals are completed).
  - How do we improve performance?
    - Buffering sequential writes in memory, grouping them as a single log.
    - Performing journaling only to Metadata (Data Bitmap + Inode)

Debugging Filesystem

- Command **fsck:** checking the file system integrity
  - detection and correction (some cases) of corruption problems in the FS.
    - Compares the list of free blocks with the directions stored in the i-nodes.
    - It also verifies the list of free inodes in contrast to the inodes in directory inputs.
    - Important limitations against file corruption.
    - Should be performed without mounting the file system.
    - Periodically it is performed during boot process.
- Command **debugfs**: a debugger for FS metadata
  - Interactive, used to examine and change the state of the file system (debugfs /dev/sda1)
    - You can use some equivalent shell commands to interact with the FS: cat, cd, chroot, ls, mkdir, pwd, rm, rmdir, stat
    - Inspect i-node content: *dump <inode num> out-file*
    - Recover an accidentally removed file (complex process*)*

Index

- Introduction
  - File System essentials
  - EXT basic features
  - The Linux Filesystem: Logic structure & organization, File permissions.
- Adding a new Storage Device
  - Identify the Disk & Create Partitions
  - Create Filesystem
  - Create mountpoint
- File System Consistency
  - Journaling
  - Checking Filesystem integrity
- Managing Filesystem
  - Tunning Filesystem
  - Resizing Filesystem
- File System Security (disk encryption)

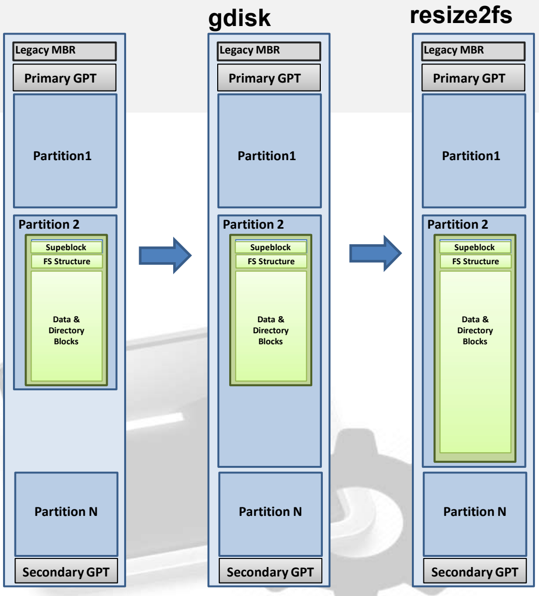
The dd command

- dd command is able to **convert** and copy files
  - This means it can read and/or write to special files, such as device files.
  - Syntax: dd [bs=BYTES] [count=N] [skip=N] [if=/path/to/FILE1] [of=/path/to/FILE2]
    - bs: read BYTES at a time
    - count: copy only N input blocks (of BYTES size each)
    - skip: skip N blocks at start of input
- Some examples of dd operation with FS:
  - copy the entire content of a harddisk: dd if=/dev/sda of=/dev/sdb
  - Copy partition content to a file, and restore it later: dd if=/dev/sda1 of=~/backupsda1.img
    - dd if=/dev/sda1 | gzip > imagen_disco.gz /// gzip –dc imagen_disco.gz | dd of=/dev/sda2

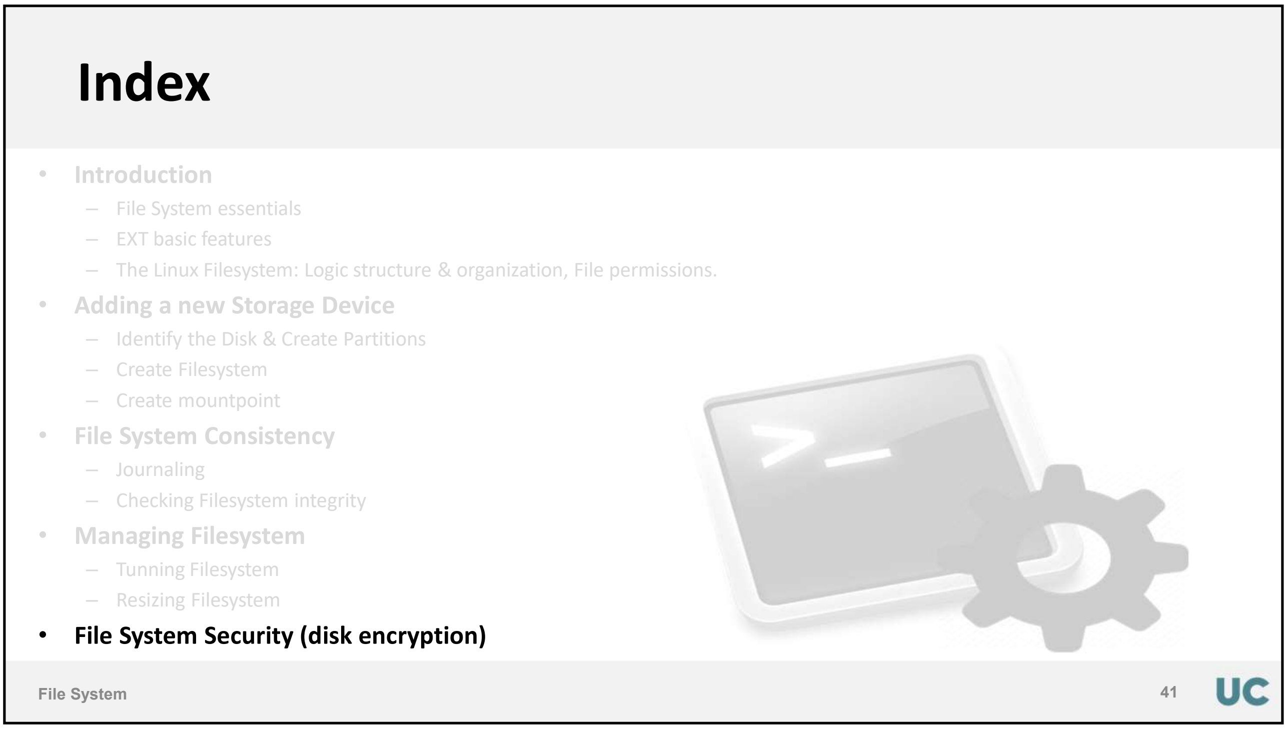
Tuning Filesystem

- Command **tune2fs**: Adjust configurable parameters of the FS:
  - [-l] List filesystem parameters
  - Control regularity of filesystem checks (fsck)
    - Time-based: [-i] interval between checks
    - Mount count based: [-c] maximum mount count
  - Control **Journaling**
    - [-j] adds journaling to the FS (from ext2 to ext3)
    - [-J] for journal options: size, location, device.
  - [-m] Control % reserved blocks (only allocated by privileged processes)

Resizing the filesystem

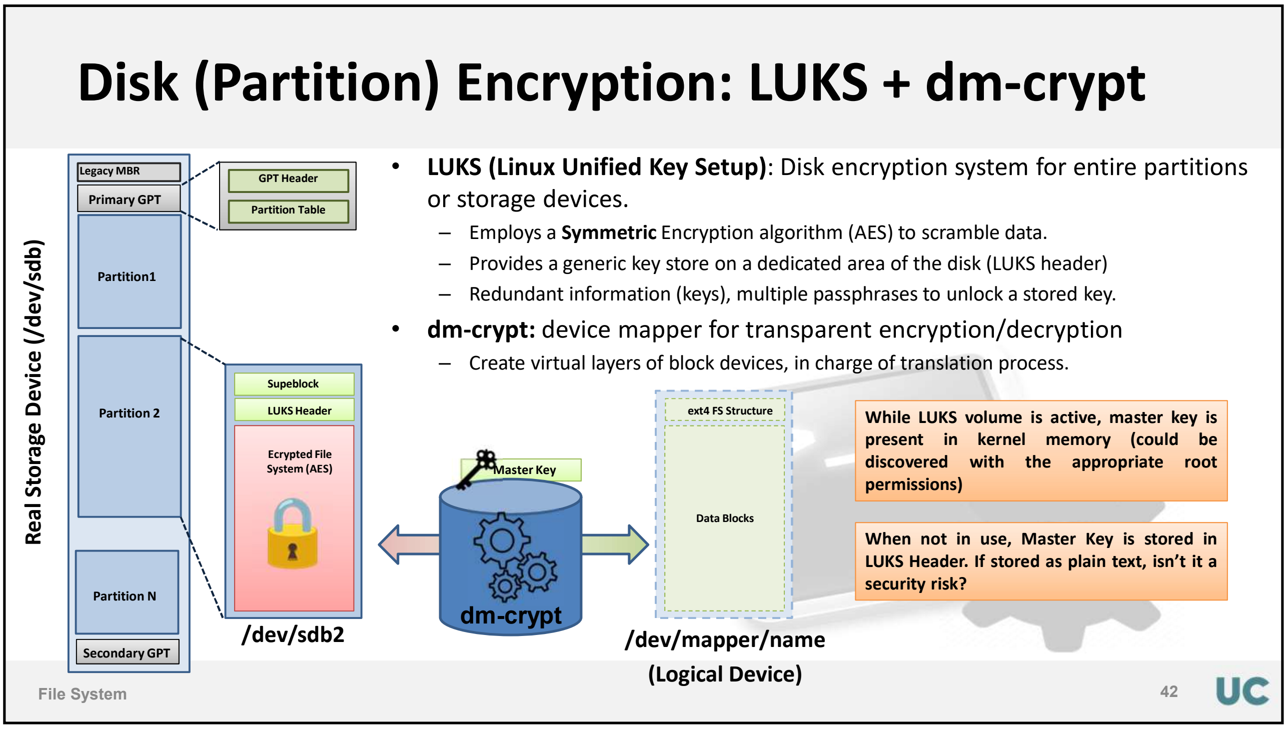


- **Resizing** the file system:
  - Command **resize2fs**:
    - Supports ext4 and requires kernel >= 2.6
    - Adjacent partitions must allow it.
- Steps:
  - First make room with **gdisk**,
  - Second: increase FS size with resize2fs.
- It is also useful to reduce the file system size
    - Combined with gdisk we can do anything: break, increase, etc.
    - Before working with partition table, make a backup dd if=/dev/sda of=part.bkp count=1 bs=1
    - **Dangerous**



Index

- Introduction
  - File System essentials
  - EXT basic features
  - The Linux Filesystem: Logic structure & organization, File permissions.
- Adding a new Storage Device
  - Identify the Disk & Create Partitions
  - Create Filesystem
  - Create mountpoint
- File System Consistency
  - Journaling
  - Checking Filesystem integrity
- Managing Filesystem
  - Tunning Filesystem
  - Resizing Filesystem
- **File System Security (disk encryption)**



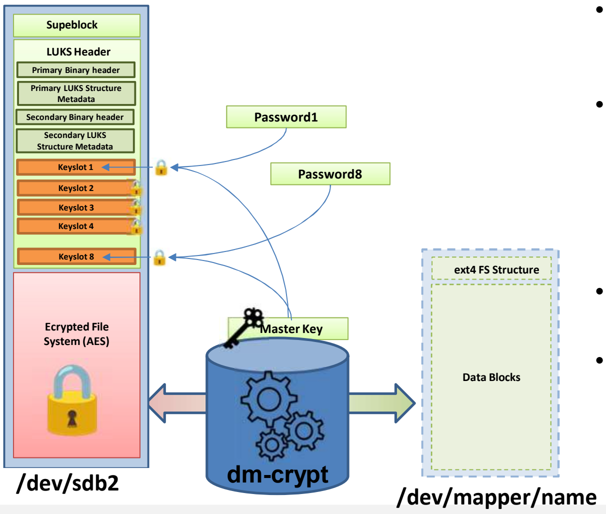
Disk (Partition) Encryption: LUKS + dm-crypt

- **LUKS (Linux Unified Key Setup)**: Disk encryption system for entire partitions or storage devices.
  - Employs a **Symmetric** Encryption algorithm (AES) to scramble data.
  - Provides a generic key store on a dedicated area of the disk (LUKS header)
  - Redundant information (keys), multiple passphrases to unlock a stored key.
- **dm-crypt:** device mapper for transparent encryption/decryption
  - Create virtual layers of block devices, in charge of translation process.

While LUKS volume is active, master key is present in kernel memory (could be discovered with the appropriate root permissions)

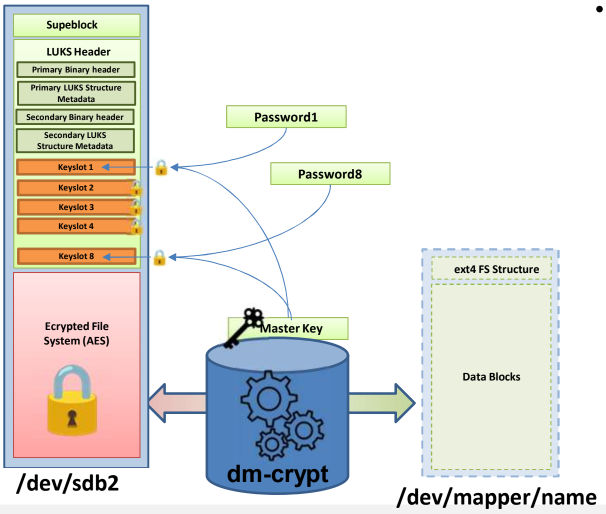
When not in use, Master Key is stored in LUKS Header. If stored as plain text, isn't it a security risk?

LUKS Header (Metadata)

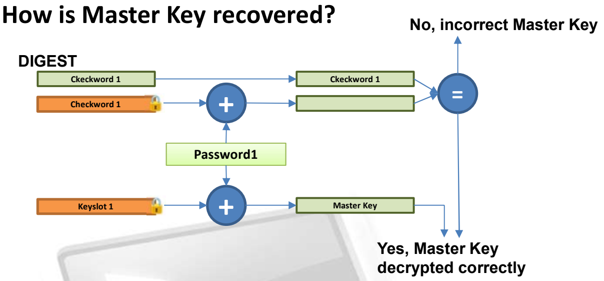


- **Binary Header**
  - Signature to detect LUKS device, basic information, header size, metadata checksum.
- **LUKS metadata**
  - Keyslots: describe the encrypted key storage areas.
  - Digest: text used to verify that keys decrypted from keyslots are correct.
  - Segments: describe areas on disk that contain encrypted data.
  - …
- Secondary header/metadata
  - Backup copies to avoid data corruption in LUKS hearder
- **Keyslost**
  - Portion of LUKS header that stores the encrypted Master Key
  - Mutiple passwords can be defined for a single encrypted disk

LUKS Header (Metadata)



- **How is Master Key recovered?**



Working with LUKS Volumes

- Command **cryptsetup**: Setup dm-crypt managed device-mapper mappings
  - Works with LUKS volumes
- LUKS Volume **creation** steps:
  1. **Create** the partition you want to encrypt (gdisk)
  2. **Encrypt** the partition using LUKS format (man cryptsetup)
     1. At this step you must provide a password used to protect your Master Key (Master Key is created internally)
  3. **Open** the encrypted partition
     1. Now you must use the password from previous step.
     2. A new device (/dev/mapper/<name>) is created, that’s your working device (logical)
  4. **Create** your File System inside the encrypted partition (mkfs)
  5. **Close** encrypted partition.
- **Using** an encrypted partition:
  1. Open the encrypted partition
  2. Mount the internal File System (in the logical device, under /dev/mapper/<name>) to work with it normally
     - If you want a permanent mount you need an entry in both /etc/fstab and /etc/cryptab
  3. Unmount and close the encrypted partition