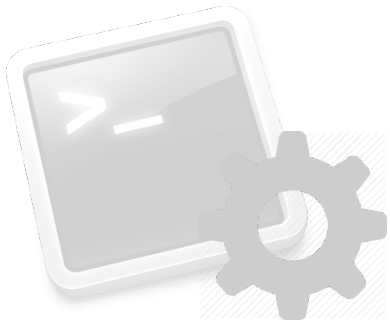
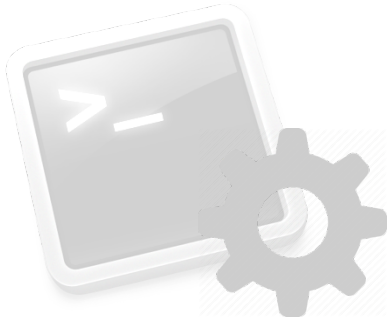
File Systems

(Getting started)



Index (Getting started)

- **Introduction**
  - Devices
  - Basic aspects about File Systems
- **Partitions, Mount/Umount**
- **FAT File System**
- **EXT File System**
  - inodes and blocks
  - block groups (ext2)
  - journaling (ext3)
  - Extents y B-Trees (ext4)
- **Virtual File System**
- **Administration**

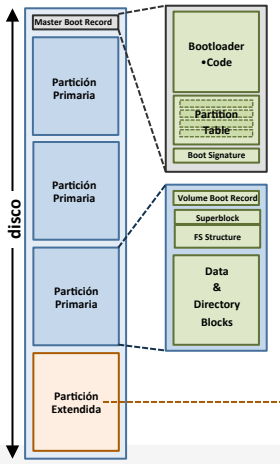


Introduction

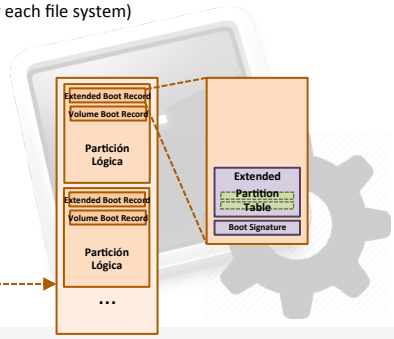
- **Tasks** of system administrator on the file system:
  - Guarantee user access to local and remote File Systems.
  - Supervision and management of storage capacity.
  - Protect information against corruption, hw failures and user errors through periodic backups.
  - Guarantee data confidentiality
  - Check File systems and repair possible corruptions.
  - Connect and configure new disks.

Introduction

- **Remember:**



- **Superblock**:
  - Keeps information about file system (version, boot file location, number of blocks, first block of / directory,…)
  - Read during file system mounting.
- **FS Structure**:
  - Mapping structure of files/directories into blocks (different for each file system)



Introduction

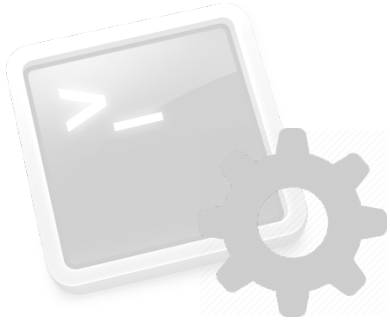
- Previous concept: **Device**
  - Name assigned to a device, physical (disk, tape, sound card,…) or logic (terminal, network port, …).
  - **Device file**: file for app-HW interactions (through the kernel).
    - Consistent way to access different devices (same group of commands
      - $ cat /dev/dsp > my_recording [talk] $cat my_recording > /dev/dsp
      - cat /dev/mouse
    - Every device file can be located in directory **/dev**
      - Standard devices [stdin, stdout, stderr] and Memory: [mem] (and virtual memory: kmem)
      - Specials: [null] (garbage), [zero] (zero generator), [random] (random number generator)…
      - Virtual terminals [ttyX] , Parallel and serial ports [lpX, ttySX], Optical devices [cdrom]
      - IDE devices [hdXX], USB/SCSI/SATA devices [sdXX], RAID devices [mdX] (or mapper/XXXX)
  - **Device driver**: kernel routines which define how to perform communication between kernel and HW (Interruptions, DMA, …)

Introduction

- Distribution of linux files
  - /dev: special files for device management.
  - /etc: Configuration files. Do not place here runnable files.
  - /lib: dynamic libraries to run /bin and /sbin binaries.
  - /proc: virtual file system for processes, with information about kernel state tables.
  - /sbin: binaries employed only by root. Needed for booting & mounting /usr.
  - /bin: base system binaries
  - /var: variable data files. Including spool directories and files, tracing and administrative files and temporary files.
  - /usr: additional tools accessible by any user.

Index (Getting started)

- Introduction
  - Devices
  - Basic aspects about File Systems
- **Partitions, Mount/Umount**
- FAT File System
- EXT File System
  - inodes and blocks
  - block groups (ext2)
  - journaling (ext3)
  - Extents y B-Trees (ext4)
- Virtual File System
- Administration

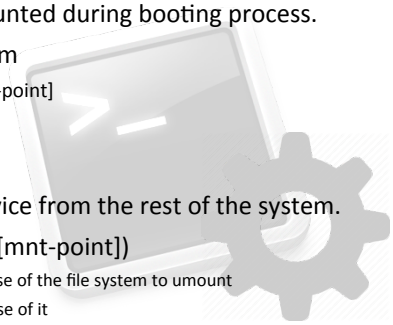


Partitions, mount/umount

- **Disk Partition**:
  - Logical storage unit which allows to treat a single physical device as multiple ones, allowing a different File system on each partition.
  - High utility for administration tasks:
    - Protect directories that tend to grow indefinitely:
      - /var/spool against "mailbombing"
      - /tmp against careless users/apps
    - Divide software and users
      - Easier upgrading, avoid users blocking the system
  - In recent kernels, the system creates alias for each partition
    - Can be employed whenever needed (Loader configuration, mounting, etc.)
    - In /dev/disk/{disk-by-uuiid}, links to the corresponding /dev/sdXX

Partitions, mount/umount

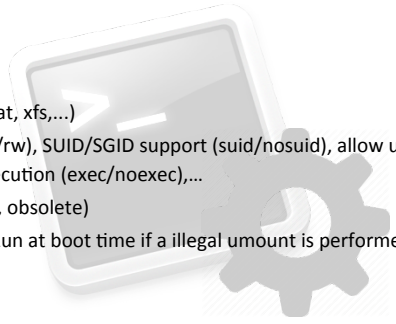
- **Mount/Umount**
  - **mounting** process provides access to the content of a disk from the file system (making use of device file)
  - Can be done for any storage device (USB, CDROM, cinta,…)
  - At least one partition (system) is mounted during booting process.
  - Command **mount**: mount a file system
    - 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/
  - **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
    - Command fuser shows the processes making use of it



Partitions

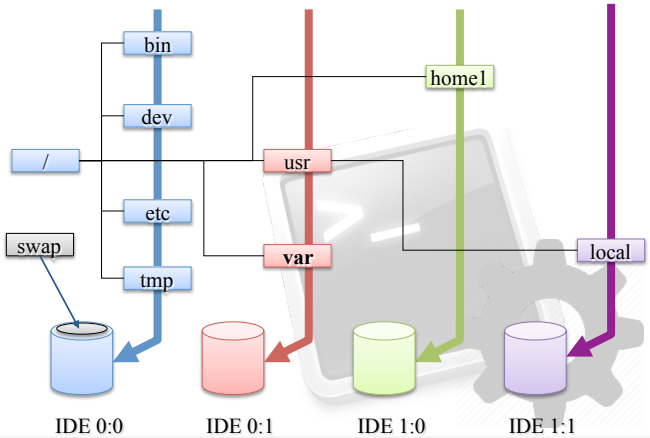
```
# /etc/fstab: static file system information.
#
#<file sys>  <mount point>   <type>      <options>          <dump>  <pass>
proc         /proc          proc        defaults           0       0
/dev/sda1    /              ext3        errors=remount-ro  0       1
/dev/sda5    none           swap   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

- **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
    - **<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. Run at boot time if a illegal umount is performed for that device (power button)



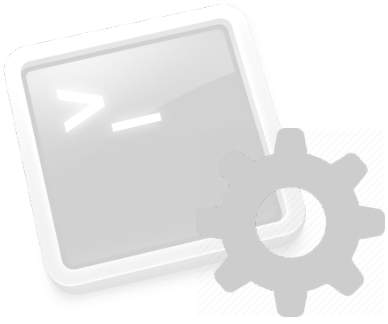
Partitions, mount/umount

- **Example: File system in multiple partitions/disks**



Index (Getting started)

- Introduction
  - Devices
  - Basic aspects about File Systems
- Partitions, Mount/Umount
- **FAT File System**
- EXT File System
  - inodes and blocks
  - block groups (ext2)
  - journaling (ext3)
  - Extents y B-Trees (ext4)
- Virtual File System
- Administration

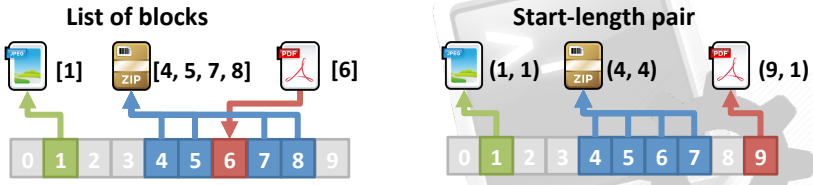


FAT File System

- Which are the main requirements for a file system?
  - Labeled files (with name)
  - File organization as a linked hierarchy (tree-like) of directories.
  - Meta-data for every file (generation time, permission, etc.)
- How is this implemented?
  - Disk performs sequential storage (blocks), does not know about hierarchies.
  - File system.
- FAT: File Allocation Tables
  - File system created in 1977 and popularized thanks to MS-DOS.
  - Still popular today (FAT32): USB, mem cards, EFI boot partition.

FAT File System

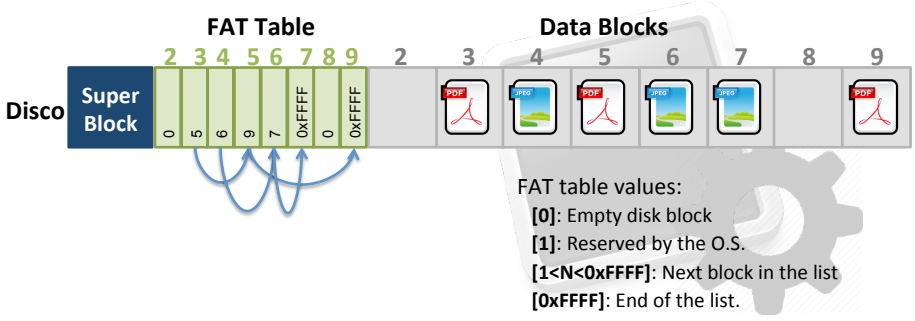
- File storage
  - A file has two main components:
    - Data: one or more disk blocks with binary information.
    - Metadata: Name, size, permission, directory, **block mapping**, …
  - Any file is stored in at least 1 disk block.
    - How can I map files in multiple blocks?



  - FAT employs list of blocks, which are stored as a linked list.

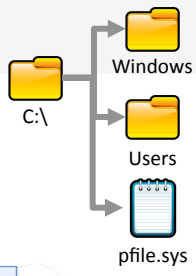
FAT File System

- File Allocation Table
  - Linked list structure that holds information about the blocks occupied by each file.
  - It also determines whether a block is in use or not.

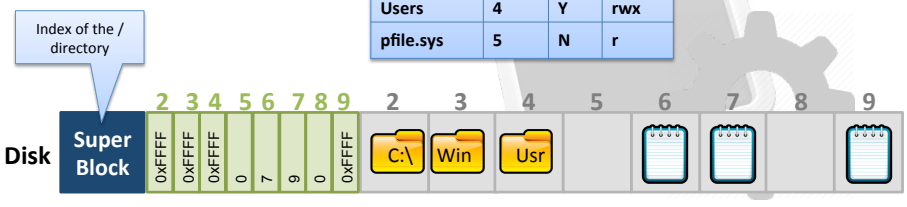


FAT File System

- Directories
  - Treated as special files.
  - It is a file containing a list with the elements in the directory.

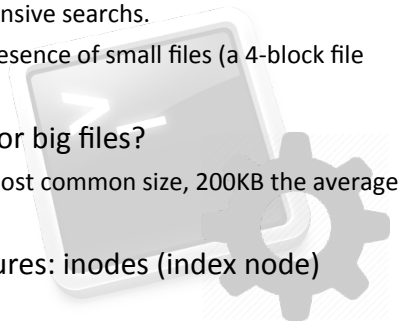


Name	Index	Dir?	Perm
.	2	Y	rwx
Windows	3	Y	rwx
Users	4	Y	rwx
pfile.sys	5	N	r



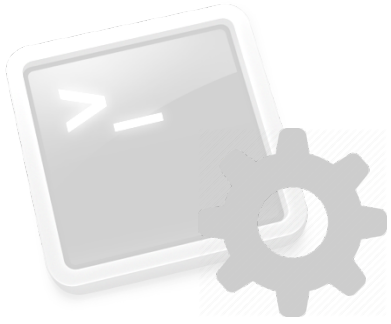
FAT File System

- Problems/Limitations:
  - Upper limit, FAT32 supports a maximum disk size of 2TB
  - Locating free blocks requires scanning the whole FAT table.
  - Prone to file fragmentation (poor locality in blocks from the same file). Metadata fragmentation-> very expensive searchs.
  - Linked lists are not efficient in the presence of small files (a 4-block file requires 4 readings of the FAT).
- Which is the common case, small or big files?
  - Seems to be small ones: 2KB is the most common size, 200KB the average size.
- Make use of more efficient structures: inodes (index node)
  - Employed in the linux file system.



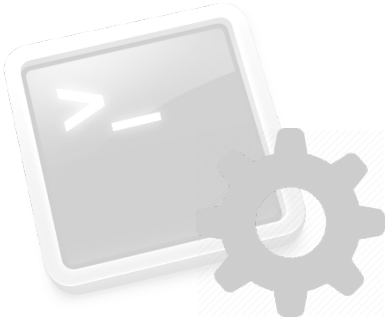
Index (Getting started)

- Introduction
  - Devices
  - Basic aspects about File Systems
- Partitions, Mount/Umount
- FAT File System
- **EXT File System**
  - inodes and blocks
  - block groups (ext2)
  - journaling (ext3)
  - Extents y B-Trees (ext4)
- Virtual File System
- Administration



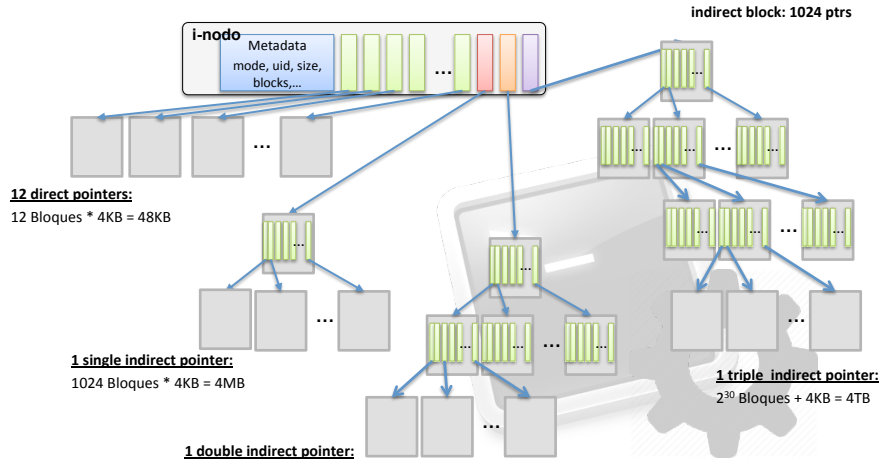
EXT File System

- The i-nodes
  - Basic building element of the file system.
  - Each file (or directory) has associated at least one i-node.
  - By default, they consume a 10% of disk storage (can be configured at FS creation time).

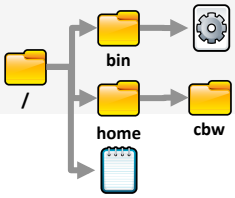


EXT File System

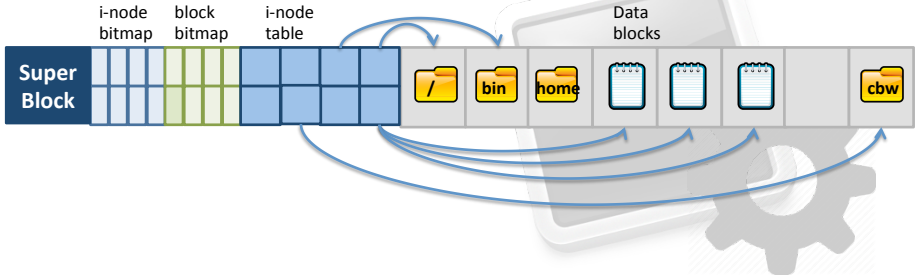
- The i-nodes



EXT File System

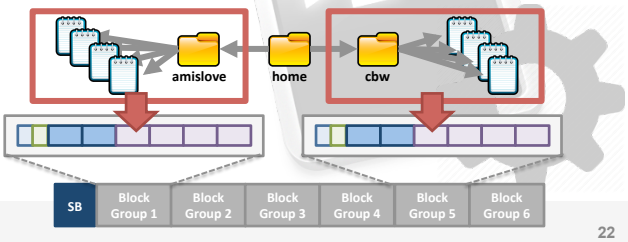


- EXT File System structure
  - i-node bitmap: bit map of occupied/free inodes.
  - block bitmap: bit map of occupied/free blocks.
  - I-node table: each input is a single i-node.



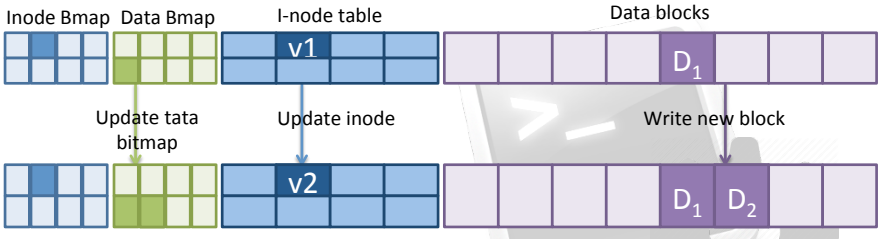
EXT File System(ext2)

- Problems/Limitations of EXT:
  - Less fragmentation of metadata, but data fragmentation still present.
  - i-nodes and their associated date can be far away in the disk.
- 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.



EXT File System(ext3)

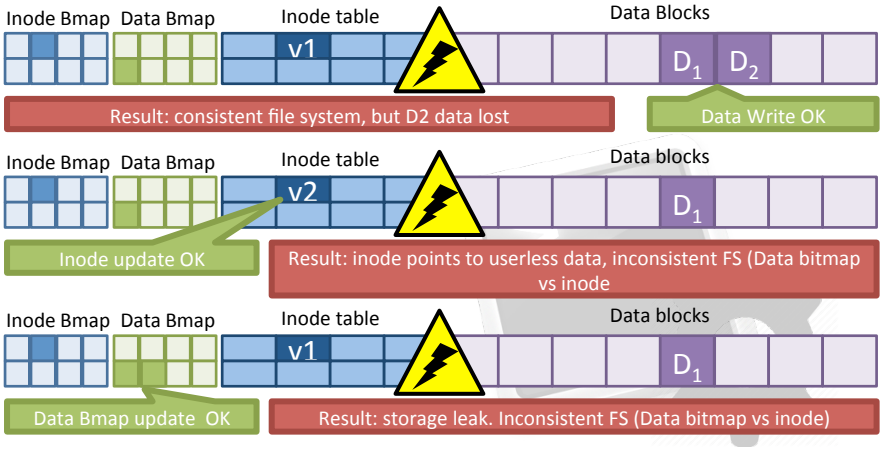
- Consistency of the file system:
  - Some operations require multiple and independent write operations in the file system.
  - Example: Add a block to an existing file (size increase).



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

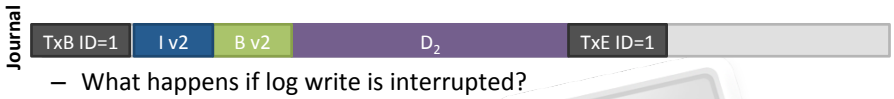
EXT File System (ext3)

- Consistency of the file system:



EXT File System(ext3)

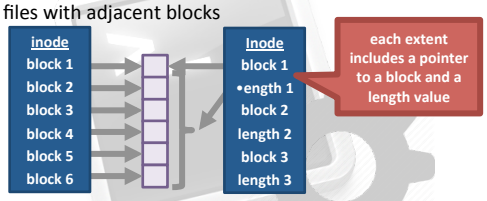
- Journaling:
  - Atomic pre-writing (at the same time) disk data.
  - Disk writes are pre-annotated in a log. Each input: journal.



  - 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)

EXT File System(ext4)

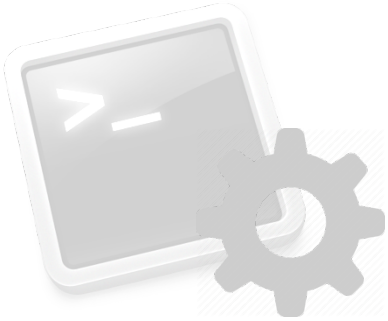
- Pointers vs Extents:
  - Inode pointers are not efficient for big files
    - Example: a 100MB file requires 25600 pointers.
    - Cannot be avoided if no contiguous blocks, but what happens in the presence of locality?
  - Current file systems try to minimize data fragmentation
    - Less searches,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



- Btrees:
  - Improved directory encoding to speed up file search.

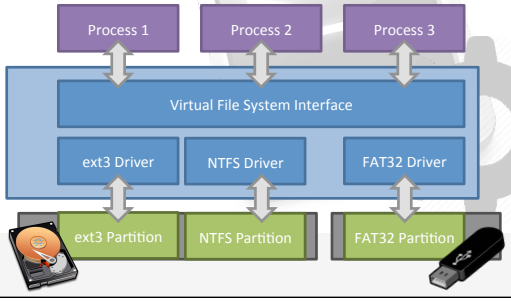
Index (Getting started)

- Introduction
  - Devices
  - Basic aspects about File Systems
- Partitions, Mount/Umount
- FAT File System
- EXT File System
  - inodes and blocks
  - block groups (ext2)
  - journaling (ext3)
  - Extents y B-Trees (ext4)
- **Virtual File System**
- Administration



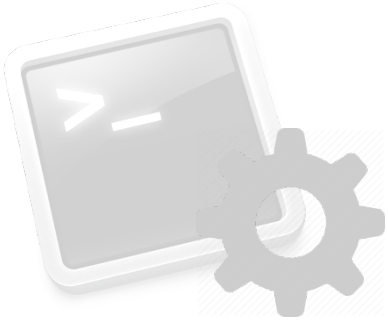
Virtual File System

- 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 a 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.



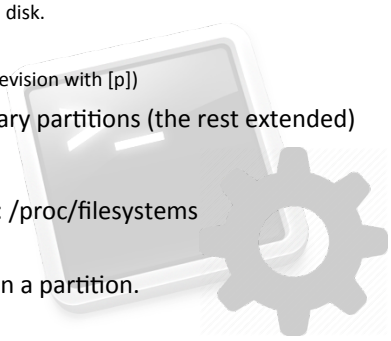
Index (Getting started)

- Introduction
  - Devices
  - Basic aspects about File Systems
- Partitions, Mount/Umount
- FAT File System
- EXT File System
  - inodes and blocks
  - block groups (ext2)
  - journaling (ext3)
  - Extents y B-Trees (ext4)
- Virtual File System
- **Administration**



Administration

- **Adding** a new disk:
  - Command **fdisk**: manipulation of the partition table
    - Syntax: fdisk /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])
  - BIOS limitations for a PC: only 4 primary partitions (the rest extended)
- **Formatting** the new disk:
  - File systems supported by the kernel: /proc/filesystems
    - Most recommended in linux is: ext3/ext4
  - Command **mkfs**: builds a file system in a partition.
    - Syntax: mkfs [-V –t fs-tipo] /dev/sda3



Administration

- **Checking** the file system:
  - Command **fsck**: 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 **badblocks**: detect and exclude broked disk sectors
    - Physical error, replace the disk immediately.
  - S.M.A.R.T
    - Utilities to access fiability/usage information about the disk (requires firmware support).
    - smartmontools

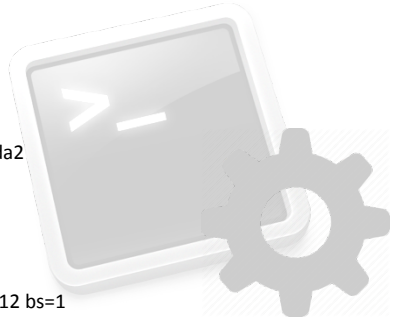
Administration

- **Resizing** the file system:
  - Command **resize2fs**:
    - Supports ext4 and requires kernel >= 2.6
    - Adjacent partitions must allow it.

    - First make room with fdisk, then resize (increase) with resize2fs.
    - It is also useful to reduce the file system size
      - Combined with fdisk 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
  - Command **parted**: manipulation of partition table and FS
    - Syntax: parted /dev/sdX
    - Can copy, move change file systems, very powerful
    - Dangerous if commands are not executed correctly!!

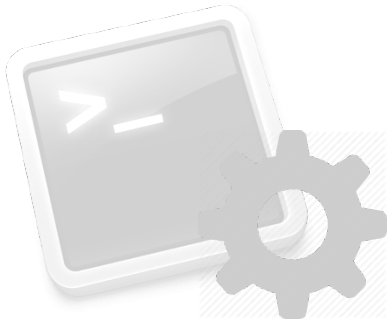
Administration

- Modify file system parameters:
  - Command **tune2fs:** Adjust configurable parameters of the FS
    - [-e] policy in the presence of error
    - [-j] add journaling
- Other tools: **dd**
  - Images of the file system:
    - dd if=/dev/sda1 | gzip > imagen_disco.gz
    - gzip –dc imagen_disco.gz | dd of=/dev/sda2
  - Copy of the file system:
    - dd if=/dev/sda1 of=/dev/sda2
  - Backup of the partition table:
    - dd if=/dev/sda1 of=backup_part count=512 bs=1



File Systems

(Advanced)

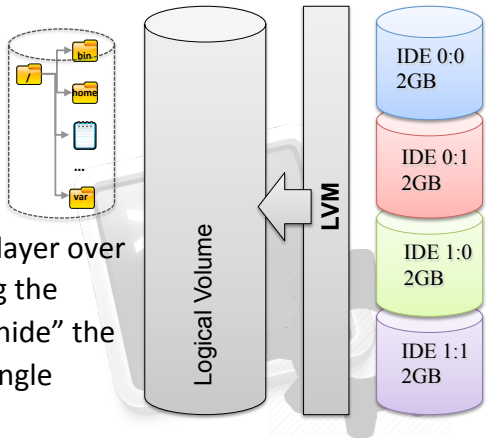


Index

- Logical Volume Manager (LVM)
- Redundant Array of Inexpensive Disks (RAID)
- Backup

Logical Volume Manager (LVM)

- My File System has a size of 4GB, but I only have 2GB Disks. Is there any solution?



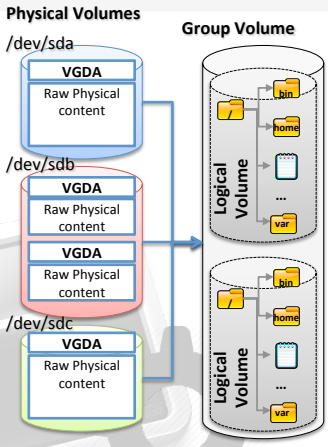
- **LVM:** creates an abstraction layer over the physical storage, allowing the creation of logical volumes("hide" the underlying HW, exposing a single Volume to the SW).

Logical Volume Manager (LVM)

- LVM Advantages
  - **Flexible management of disk storage**: avoid the limitations imposed by disk physical size. A File System can be extended through multiple disks.
  - **Re-sizeable Storage**: logical volumes can be extended/reduced in a simple way. Some operation do not require File System umounting.
  - **On-line Data movement**: data con be moved between disks while these disks are in use. I can replace a disk without interrupting system service.
  - **Taking "Snapshots"**: eases the process to take snapshots to devices (backup)

Logical Volume Manager (LVM)

- LVM Hierarchy:
  - Physical Volumes (PV)
    - Lowest level of LVM hierarchy
    - Complete disk or partition
    - Contains VGDA (Volume Group Descriptor Area) and the raw physical content.
  - Group Volumes (VG)
    - equivalent to “super-disks”
    - Built with one or more PVs
      - more PVs can be added to the GV without modifying the previous ones
  - Logical Volumes(LV)
    - Equivalent to “super-partitions”
    - File Systems are created on a Logical Volume



Logical Volume Manager (LVM)

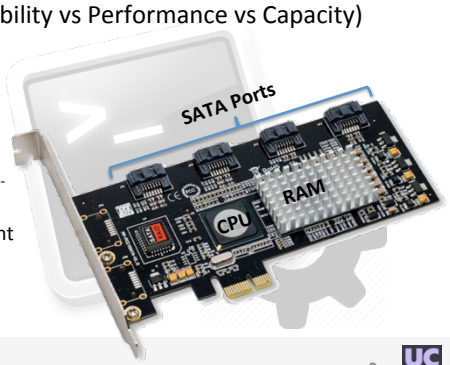
- LVM Administration:
  - Command **pvcreate**: creation of a Physical Volume.
    - Syntax: pvcreate [partition] (It is necessary to previously create a partition with fdisk).
  - Command **vgcreate**: creation of a Group Volume from multiple PVs.
    - Syntax: vgcreate [name-vol] [PVs]
      - Example: vgcreate vg01 /dev/sdb /dev/sdc1 (group disk sdb and partition sdc1 in a GV in /dev/vg01).
  - Command **lvcreate**: creation of a Logical Volume
    - Syntax: lvcreate [GV] –L[size] –n[name-vl]
      - Example: lvcreate vg01 –L1000M –nvol1 (after this we can create the FS with mkfs)
  - Need more storage?
    - add a new Physical Volume to the Group Volume (**vgextend**)
    - Extend the Logical Volume to the larger Group Volume (**lvextend**)
    - Re-size the File System (resize2fs).
      - Can do this online !!! (…In contrast, reductions must be done offline)
    - We can also reduce VG and LV (**vgreduce**, **lvreduce**)

Index

- Logical Volume Manager (LVM)
- **Redundant Array of Inexpensive Disks (RAID)**
- Backup

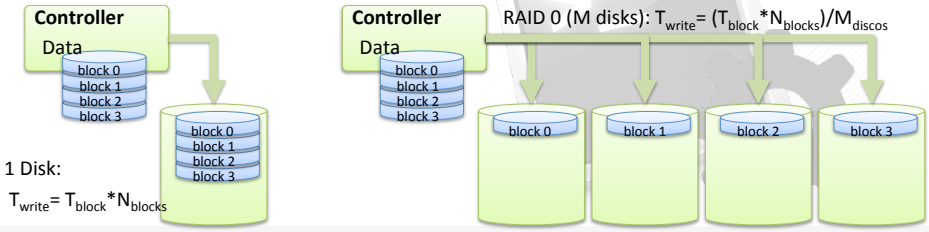
RAID (Redundant Array of Inexpensive Disks)

- Mechanism to provide reliability and performance in disks.
  - Make use of multiple disks to create the illusion of a disk with larger capacity, faster access and fault tolerant.
  - Transparent to the user and the OS..
  - Different configuration options (Reliability vs Performance vs Capacity) denoted as levels [RAID0 … RAID7].
  - Can be implemented via HW or SW
    - HW Implementation: High efficiency but also high cost.
      - RAID Controller: CPU +dedicated sw, RAM + non-voletile memory.
    - SW Implementation: Efficient management of simplest RAID configs (0,1).



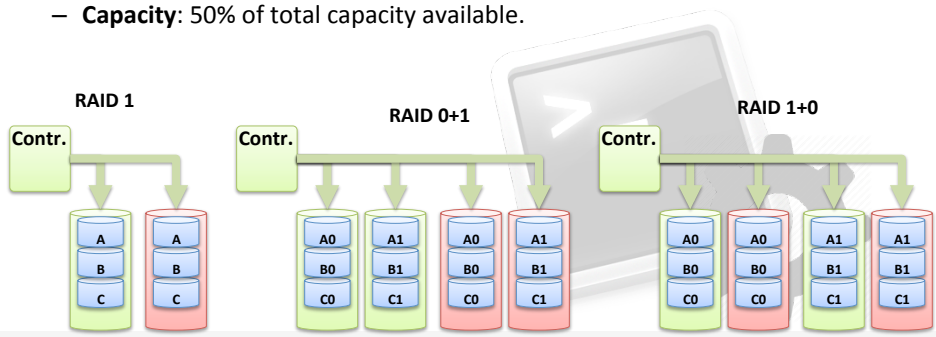
RAID (Redundant Array of Inexpensive Disks)

- **RAID 0** (striping) :
  - Data are divided into segments (strips) and distributed among multiple disks.
    - parallel access to disks.
  - **Performance**: improves read/write latency
    - Speed increases as the number of disks grows (also depends on data size).
  - **Reliability**: no fault tolerance.
  - **Capacity**: 100% storage utilized (no redundancy).



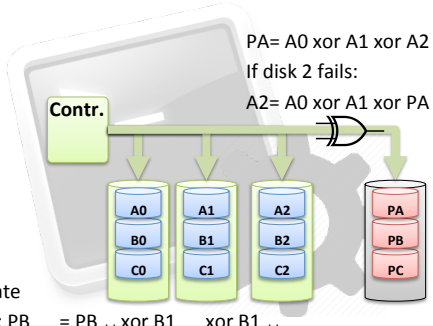
RAID (Redundant Array of Inexpensive Disks)

- **RAID 1** (mirroring) :
  - Employ a secondary disk to copy all data being modified
  - **Performance**: low performance caused by writes (everything replicated)
  - **Reliability**: High redundancy, one disk can fail.
  - **Capacity**: 50% of total capacity available.



RAID (Redundant Array of Inexpensive Disks)

- **RAID 4** (striping + parity) :
  - One disk stores information about the parity of the rest.
  - Block-level division (1 strip= 1 block). Can access disks individually.
  - **Performance**: High performance for reads. Bottleneck for writes.
  - **Reliability**: Tolerance to 1 faulty disk.
  - **Capacity**: Only 1 disk is not available.



¿How to calculate new parity after a write event?

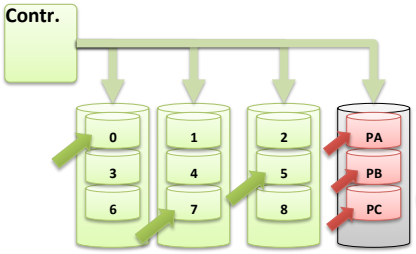
(Example: write in block B1)

Option1: Read the rest of blocks (B0, B2) and recalculate

Option2: Read the content of B1 and PB and calculate: $PB_{new} = PB_{old}$ xor $B1_{new}$ xor $B1_{old}$

RAID (Redundant Array of Inexpensive Disks)

- Write problem in **RAID 4**:
  - Need to write in positions 0, 5, 7



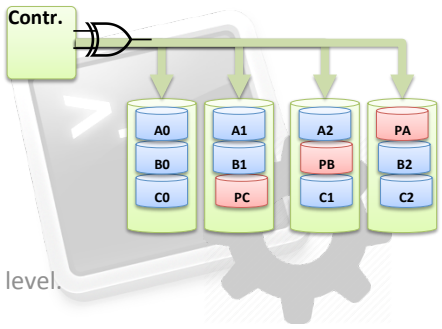
Write process:

1. Read the blocks 0,5,7 and PA,PB,PC.
2. Calculate the new value of PA, PB, PC
3. Write new data blocks
4. Write new parity blocks

Serialized (writes in the same disk)-> Low performance

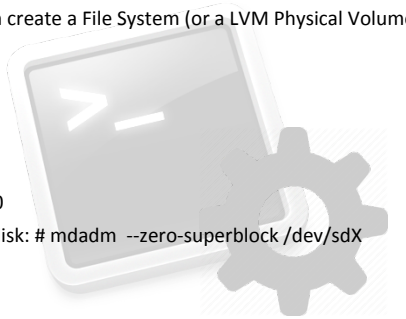
RAID (Redundant Array of Inexpensive Disks)

- **RAID 5** (striping + distributed parity) :
  - Parity information is distributed among all the disks.
  - Similarly to RAID 4, block-level division (1 strip= 1 block).
  - **Performance**: Eliminate the writes bottleneck.
  - **Reliability**: Tolerates 1 faulty disk.
  - **Capacity**: only 1 disk lost.
- RAID 6 (striping + double parity)
  - RAID 4 + double parity distribution
  - Tolerates two faulty disks.
- RAID 2, RAID 3
  - Parity control at a lower (than block) level.
  - Rarely employed.



RAID (Redundant Array of Inexpensive Disks)

- RAID Administration, command **mdadm** :
  - **Creation** of a RAID device:
    - **# mdadm --create /dev/md0 --verbose --level=0 --raid-devices=2 /dev/sdb /dev/sdc2**
    - It is necessary the previous partitioning of disks (fdisk)
    - Creation process can be monitorized: # cat /proc/mdstat
    - Created a RAID in /dev/md0. On it we can create a File System (or a LVM Physical Volume).
  - **Monitorization** of RAID system:
    - # cat /proc/mdstat
    - # mdadm --monitor [options] /dev/md0
  - **Elimination** (deactivation) of RAID:
    - "Stop" device: # mdadm --stop /dev/md0
    - Clean previous information from a RAID disk: # mdadm --zero-superblock /dev/sdX

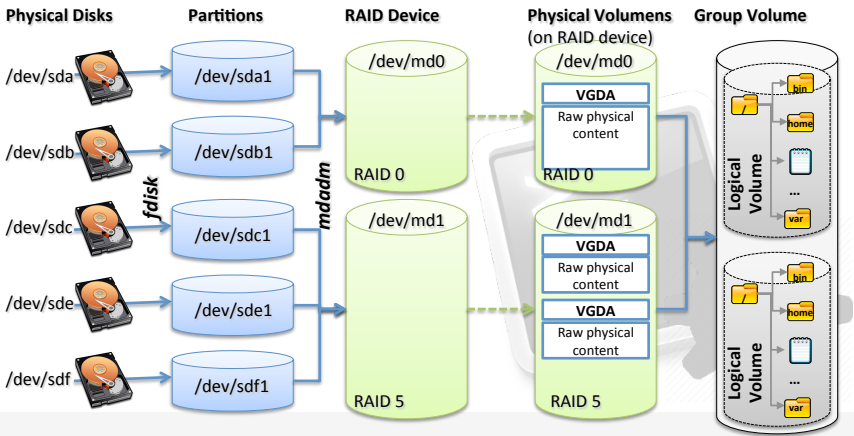


RAID (Redundant Array of Inexpensive Disks)

- Procedure for a **disk failure**:
  - Assume a RAID5 system, still operative with a significant performance degradation.
  - Broken disk can be automatically **restored**:
    1. Eliminate broken disk from RAID: # mdadm /dev/md0 –r /dev/sdc1
    2. Physically replace with another one (identical)
    3. Create the partitions as in the original: # fdisk /dev/sdc
    4. Add it to the RAID device: # mdadm /dev/md0 –a /dev/sdc1
    5. Monitorize the reconstruction process: # cat /proc/mdstat
  - We can simulate a disk failure:
    - # mdadm /dev/md0 –f /dev/sdc1
    - All the process log information in /var/log/messages

RAID (Redundant Array of Inexpensive Disks)

- Combination **RAID + LVM**
  - RAID must be implemented below LVM

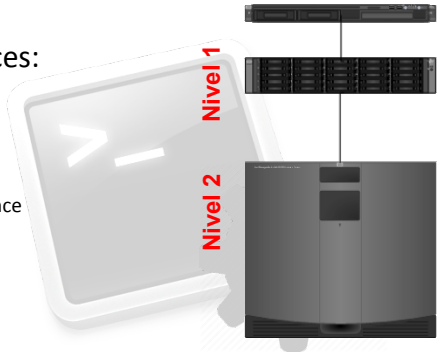


Index

- Logical Volume Manager (LVM)
- Redundant Array of Inexpensive Disks (RAID)
- **Backup**

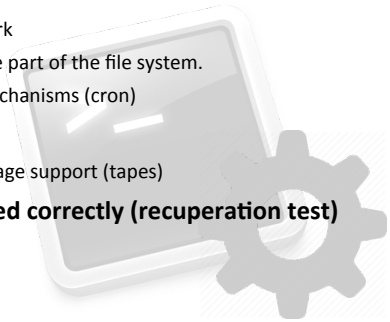
Backup

- RAID+journaling not enough to provide 100% availability.
- Essential: backup copies
  - Solution for multiple unexpected events, both HW and SW.
  - Mainly “the users”.
- Performed with dedicated resources:
  - Hard Disks
    - Exclusively dedicated to backup
    - SAN Servers
    - Disk hierarchy with decreasing performance
  - Tapes (or other magnetic support
    - LTO (Linear Tape-Open) (LTO-6 Ultrium):
      - 2.5TB capacity, 160MB/s transference.
    - Others: SAIT, AIT



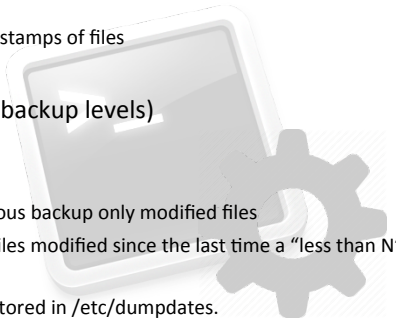
Backup

- **Backup Policy**: configured according to our requirements
  - **What** do we need to store?
    - Data from users/apps/system
    - Select the critical parts of the system
  - **When** do we want to backup?
    - Do not overload systems with useless work
    - Depends on the kind of utilization and the part of the file system.
    - Employ programming/automatization mechanisms (cron)
  - **Where** do we want to backup?
    - Efficient labeling and organization of storage support (tapes)
  - **Check always that the backup finished correctly (recuperation test)**



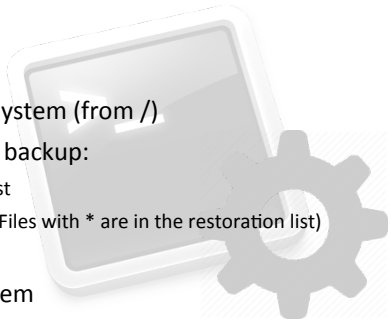
Backup

- Basic system tool: **dump/restore**
  - Present in most UNIX/Linux systems
  - Many advanced tools employ this as starting point.
  - Designed to work at File System level
    - Can copy any kind of files (even devices)
    - Preserves permissions, property and timestamps of files
    - "sparse" files managed correctly.
  - backups are performed incremental (backup levels)
    - Only available for the whole File Systems.
    - Level 0: (FULL) Copy all files from scratch.
    - Level 1: (INCREMENTAL) Add to the previous backup only modified files
    - Level N: Add to the previous backup the files modified since the last time a "less than N" backup was performed.
    - The information about backup history is stored in /etc/dumpdates.



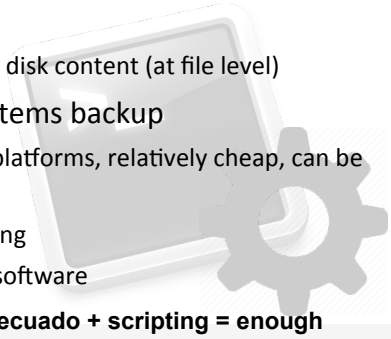
Backup

- Creation of backups with **dump** command
  - Syntax: dump -<level> <options> -f [destination] [File system]
    - Level: int from 0 (FULL) to 9
    - Option –f: destination of backup file. Can be a device file (tape)
    - Option –u: update the file /etc/dumpdates after the backup.
    - Example: # dump -0u -f /dev/tape /
- Recovery with **restore** command
  - restore –C: Compare the stored File system (from /)
  - restore –i: interactive operation with backup:
    - add/delete: files/dirs to the restoration list
    - cd/ls/pwd: move through the backup FS (Files with * are in the restoration list)
    - extract: restore the files from the list
  - restore –r: restore the whole file system
    - # restore –r –f <backup_file> <destination>



Backup

- Alternative tools(rudimentary):
  - Command tar (package):
    - Can understand devices without file system
    - Can be completed with compression tools (bzip, zip)
  - Command dd
    - # dd if=/dev/sda2 of=/dev/tape
  - Command cp –a: optimal to replicate disk content (at file level)
- Advanced tools for distributed systems backup
  - Data Protector (HP): many different platforms, relatively cheap, can be integrated with HP OpenView
  - Legato/Tivoli (IBM): expensive licensing
  - Bacula: GNU alternative to non-free software



rdump + rrestore + HW adecuado + scripting = enough