

# Advanced Linux System Administration

## Subject 2. Command Line (Shell)



**Pablo Abad Fidalgo**  
**José Ángel Herrero Velasco**

Departamento de Ingeniería Informática y Electrónica

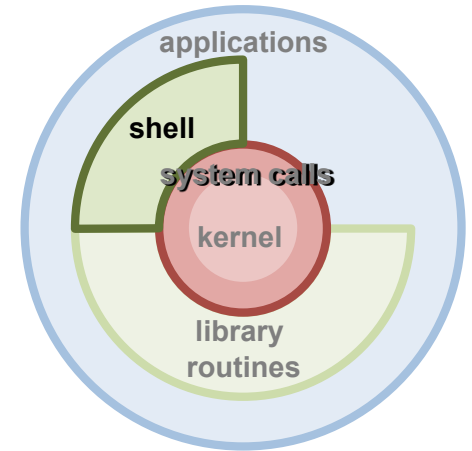
Este tema se publica bajo Licencia:

[Creative Commons BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/)

# Index

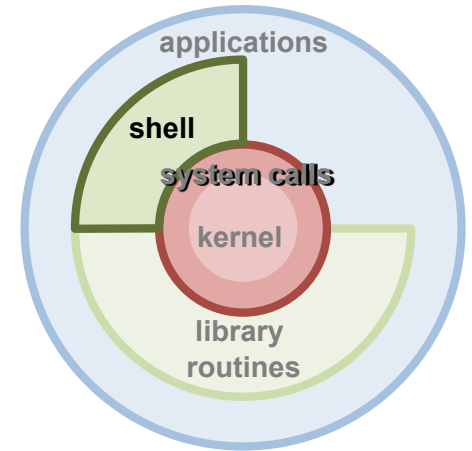
- **The shell.**
- **File System.**
- **“Shortcuts”.**
- **User management.**
- **Environment Variables.**
- **Redirection and Pipes.**
- **Shell Scripting.**
- **Process management.**
- **Advanced Administration commands.**

# The shell



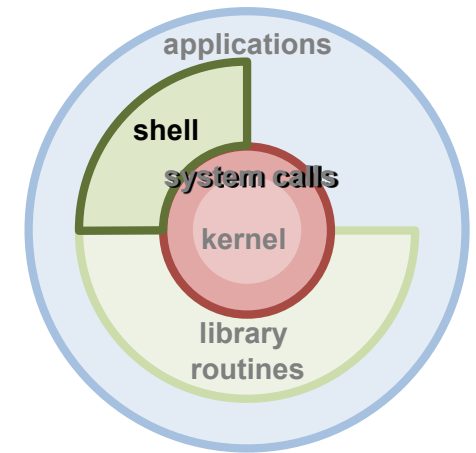
- **Interface** for system calls:
  - **POSIX** Compatibility (independent of the system).
  - Move from user mode to supervisor mode: TRAP.
  - Usually from C language.
- **Command Interpreter:**
  - Same privileges as other program.
  - Multiple interpreters available: sh, csh, ksh, tcsh, **bash**...
  - Responds with the prompt: test@si:~\$ (normal account:\$, root account:#).
- **Session (login + passwd):**
  - Local Access: 6 consoles in text mode (Ctrl+Alt+F1...F6) and 1 graphic console (Ctrl+Alt+F7).
  - Remote access: through network (telnet, rlogin, ssh...).

# The shell



- **Shell Types:**
  - Bourne shell “**sh**” (/bin/sh): old UNIX syntax (SysV).
  - C shell “**csh**” (/bin/csh): C-like syntax (BSD).
  - Bourne Again shell “**bash**” (/bin/bash): Similar to its antecessor, but extended with many features from csh.
  - Tcsh “**tcsh**” (/bin/tcsh): improved version of the original C shell.
  - In general, differences are not relevant for day-to-day use.
- **Shell Goal:** interactive dialog between user and system:
  - Through a huge amount of orders/commands and applications:
    - Change execution mode (background/foreground).
    - Input/Output redirection.
    - Command Pipes and redirection.
    - Scripting.
    - 100% Customizable.

# The shell

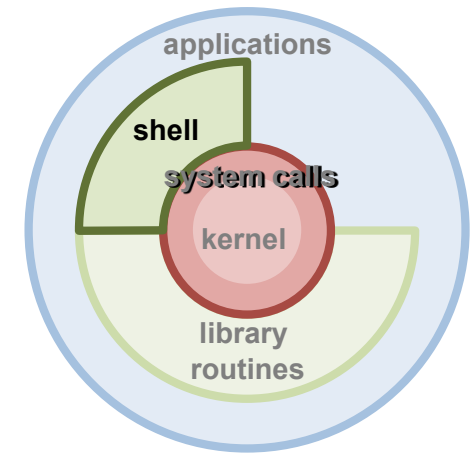


- **Command structure:**

```
user@machine:~$ command -<options> [arguments]
```

- **Options:** command pieces that modify the initial behavior.
  - **Arguments:** file name or any other kind of data needed by the command.
- 
- **Man command** (formats and displays manual pages):
    - First command to learn. Displays on screen information about a command, programming function, configuration file, etc.
    - Syntax: `$ man -<options> [command]`:
      - **-a:** display all the manual pages that match “command”, not just the first one.
      - **-K:** search for the specified string in all man pages.

# The shell



- **Man** command: sections:

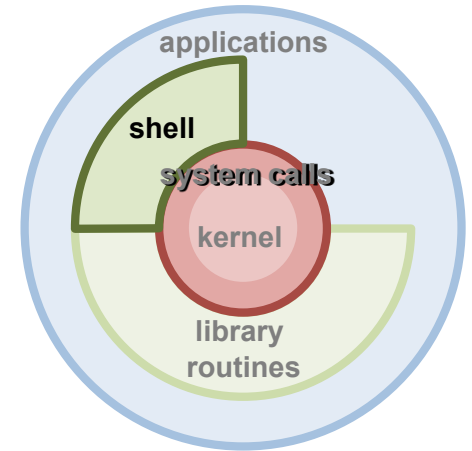
- Manual organized in sections. `/usr/share/man`:

- Usually, searching is performed in a specific order through all the sections, and only the first matching is displayed.
    - If the command specifies the section, search is only performed in that section.

- Manual sections:

1. User commands.
2. System Calls (functions provided by the kernel).
3. C Library functions (system library functions).
4. Devices and special files (usually found under `/dev`).
5. File formats and conventions. Example `/etc/passwd`.
6. Games.
7. Miscellanea: `man(7)`, `groff(7)`.
8. System administration tools and Daemons.

# The shell

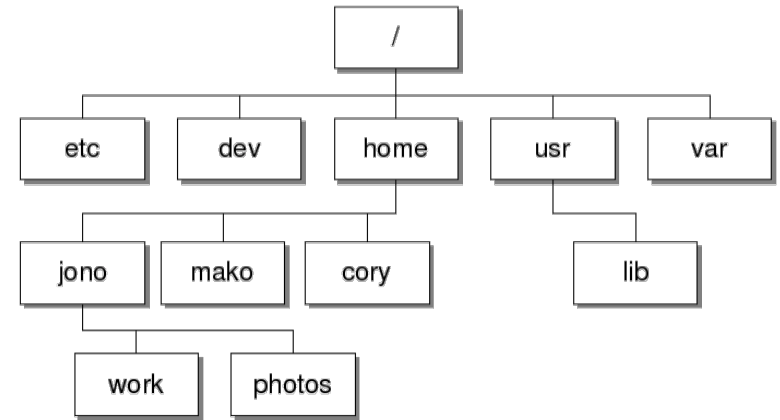


- **Man** command: configuration:
  - Through the file `/etc/manpath.config`:
    - Can make use of a different configuration file: `$ man -C new_file`.
  - The command **manpath** indicates the routes to look for the manuals:
    - Can also be modified, through `$ man -M path` or modifying the environment variable `$MANPATH`.
  - The section order for searching can also be modified: `$MANSECT`.
  - The application employed to display manual pages can also be chosen: `$PAGER` (by default: `less`).
  - Also the language can be selected: `$LANG`.

# Index

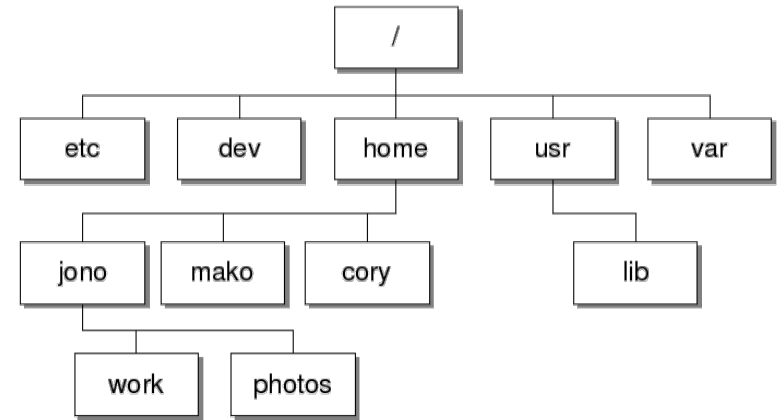
- The shell.
- **File System.**
- “Shortcuts”.
- User management.
- Environment Variables.
- Redirection and Pipes.
- Shell Scripting.
- Process management.
- Advanced Administration commands.

# File System



- Definition:
  - Logic structures and their corresponding methods employed by the Operating System to organize the files in the disk.
- Tree-like **Hierarchical** structure:
  - Efficient management of information (group related info into folders).
  - Folders separated by /
  - File access (path):
    - Absolute: `cd /home/pepe.`
    - Relative to current path (with “.” o “..”): `cd ../../../usr/local.`
- Files starting with “.” are “hidden”.
- Security: protection of files against unauthorized accesses.

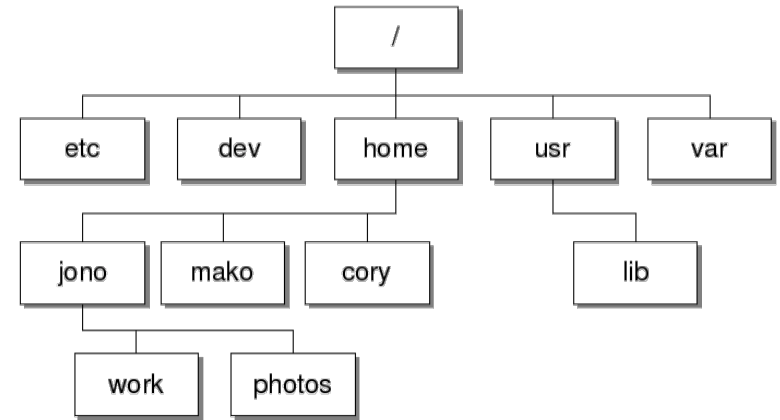
# File System



- Unit Mounting:
  - A storage device (usb, cd, etc.) can be associated with a particular position in the directory tree.
- Same treatment to files and I/O devices:
  - Same program can employ files and/or devices indifferently.
- Different locations of the file tree can be linked (**ln** command).
- Definition of a folder/file **path**:
  - Directories to be traversed, starting from root directory, in order to reach that folder/file.

# File System

|                   |                                                            |
|-------------------|------------------------------------------------------------|
| <b>/</b>          | Root directory.                                            |
| <b>/bin</b>       | Core operating system commands.                            |
| <b>/boot</b>      | Kernel and files needed to load the kernel.                |
| <b>/dev</b>       | Device entries for disks, printers, pseudo-terminals, etc. |
| <b>/etc</b>       | Critical startup and configuration files.                  |
| <b>/mnt</b>       | Temporary mount points, mounts for removable media.        |
| <b>/lib</b>       | Libraries, shared libraries and parts of the C compiler.   |
| <b>/home</b>      | Default home directories for users.                        |
| <b>/opt</b>       | Optional software packages (not consistently used).        |
| <b>/root</b>      | Home directory for the superuser.                          |
| <b>/sbin</b>      | Command needed for minimal system operability.             |
| <b>/proc</b>      | Information about all running processes.                   |
| <b>/tmp</b>       | Temporary files.                                           |
| <b>/usr</b>       | Hierarchy of secondary files and commands.                 |
| <b>/usr/local</b> | Software installed by users.                               |
| <b>/var</b>       | System specific data and configuration files.              |



# File System (Commands)

- Large amount of shell command to interact with FS.
- For a detailed description, take a look at the APPENDIX or consult system man pages.
- Navigating through the file system:
  - Command **pwd**: displays current.
  - Command **cd**: change to a different directory.
  - Command **mkdir**: create a new folder.
- File Manipulation:
  - Command **ls**: list folder contents in alphabetical order.
  - Command **cp**: copy files.
  - Command **mv**: move files (or rename).
  - Command **rm**: remove files or folders.

# File System (Commands)

- File Manipulation (cont.):
  - Command **ln**: create a link between two files.
  - Command **whereis**: locate the path of a cmd's binary/src code/manual.
  - Commands **locate**/**find**: locate a file in the directory tree.
- File Contents:
  - Commands **cat**/**more**/**less**: show the contents of a file.
  - Command **wc**: count the number of bytes/words/lines in a file.
  - Commands **head**/**tail**: display in stdout the first/last lines of a file.
  - Command **grep**: display the lines of a file that match a text pattern.
  - Command **tar**: add the contents of a file tree to a single file.
  - Command **cut**: remove specific sections of each line of a file.
  - Command **sort**: arrange file lines in specific order (alphabetical).
  - Command **vi**: text editor in the terminal (present in every UNIX system).

# Index

- The shell.
- File System.
- **“Shortcuts”.**
- User management.
- Environment Variables.
- Redirection and Pipes.
- Shell Scripting.
- Process management.
- Advanced Administration commands.

# Shortcuts

- Some simple “tricks” that might make your life a bit easier...
- Commands/filenames/paths can be **autocompleted**:
  - Tab (in bash).
  - If it cannot be completely resolved, a list with all the alternatives is displayed.
- Moving the cursor through the command line (prompt):
  - [Ctrl+a]: go to the beginning of the command. [Ctrl+e]: move to the command end.
  - Cursor Left/Right: move through the command line (char by char).
  - [Ctrl left/right]: move word by word.
- Navigating through the command history:
  - Easiest way: Cursor Up/Down.

# Shortcuts

- Command **history**:

- The commands employed in a shell session are stored. With this command we can review command executed, repeat or edit previous commands:
  - **!!**: execute again the last command of the list (previous command).
  - **!letters**: execute again the last command executed starting with the letters indicated.
  - **!number**: execute the command in the list with that number.
- List size can be configured (\$HISTSIZE in bash) (set).

- Employing **regular expressions**:

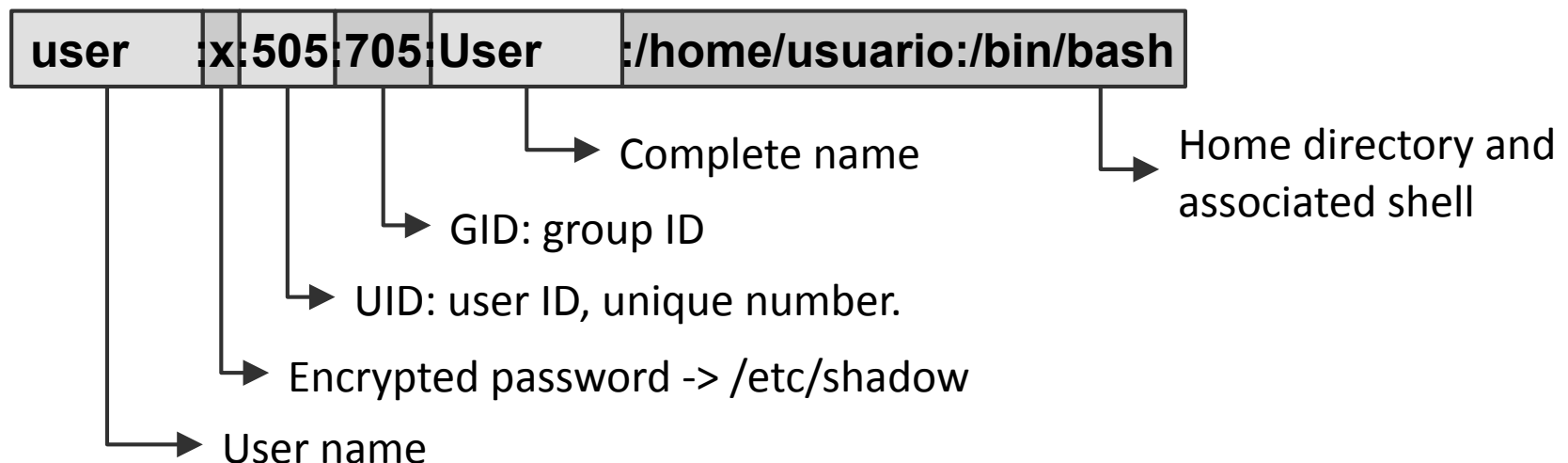
- Some characters cannot be employed in filenames, having a special purpose:
  - **"\*"**: replace all characters: `$ ls -l pa* // $ rm -fr /*` (oops!!).
  - **"?"**: replace a single character: `$ rm pepe?` (remove pepea, pepeb, pepec, etc.).
  - **"["**: replace a single numerical character: `$ rm pepe[12]` (remove pepe1 and pepe2).
  - **"{"**: for expansion: `$rm p{e,i}pe` (removes pepe and pipe).
  - **"~"**: designates \$HOME directory.
- What if I need to search the character \* in a file? (`\ o ""`).

# Index

- The shell.
- File System.
- “Shortcuts”.
- **User management.**
- Environment Variables.
- Redirection and Pipes.
- Shell Scripting.
- Process management.
- Advanced Administration commands.

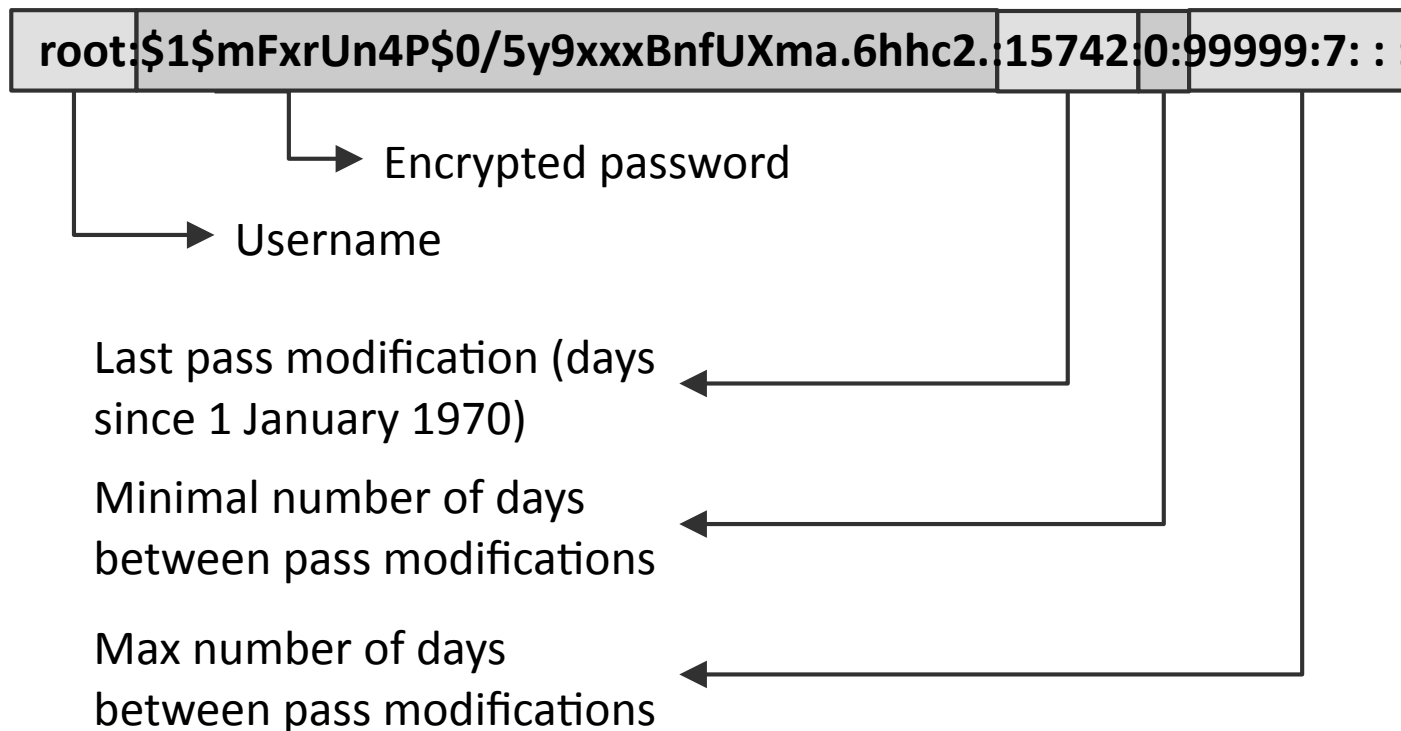
# User Management

- In UNIX, users are organized in groups.
- The files `/etc/passwd` and `/etc/group` contain information about all the users and groups of the OS:
  - As well as system login, these files include basic user configuration (home directory, shell).
  - Group management: useful to control access to certain parts of the system.
  - For each user, passwd file contains a line with the following format:



# User Management

- The file `/etc/shadow` manages user passwords:
  - For each user, the file shadow contains a line with the following format:



# User Management

- Based on users and groups, UNIX implements a protection mechanism for the File System based on permissions.
- Each file (and folder) has a single owner and access permissions.
- The different permissions are:
  - Read (r): allows read access to the contents (list directory files).
  - Write (w): allows content modification (create/remove/move files).
  - Execute (x): execute a file (no specific extension is required (windows exe)).
- File permissions can be configured according to three types:
  - User: file owner.
  - Group: rest of the users from the same group as the owner.
  - Rest: rest of system users.

# User Management

- Conventional users only have write permissions in their \$HOME directory: /home/<usuario>:
  - Also in temporary directories (such as /tmp).
- Superusers (system administrators) have unlimited access to the whole file system (Warning!!).
- Information about file/directory permissions with [ls -l]:

| permissions | link | owner | group | date        | time  | file  |
|-------------|------|-------|-------|-------------|-------|-------|
| -rwxr--r--  | 1    | pablo | users | 2014 Apr 13 | 14:11 | file1 |

Diagram illustrating the components of the `ls -l` output line:

- permissions:** -rwxr--r-- (indicated by a box with columns for user, group, and rest).
- link:** 1
- owner:** pablo (circled, with "owner" written below).
- group:** users (circled, with "group" written below).
- date:** 2014 Apr 13
- time:** 14:11
- file:** file1

# User Management (Commands)

- Detailed description in the APPENDIX.
- Basic user management:
  - Command **whoami**: displays username.
  - Command **who**: shows users connected to the system.
  - Command **passwd**: change user password.
  - Command **finger**: shows the status of a user in the system.
  - Command **write**: sends a txt message to other user's terminal.
- File Permission management:
  - Command **chmod**: modify file or directory permissions.
  - Command **chown/chgrp**: modify UID/GID of a file.
  - command **umask**: modifies default permissions assigned to new files.

# Index

- The shell.
- File System.
- “Shortcuts”.
- User management.
- **Environment Variables.**
- Redirection and Pipes.
- Shell Scripting.
- Process management.
- Advanced Administration commands.

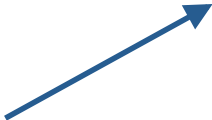
# Environment Variables

- Group of shell session variables with a pre-defined value. Their value is obtained this way: `$ echo $VARIABLE`.
- Allow the configuration of certain aspects in the cmd interpreter.
- Two kinds:
  - **User variables:** internal to our shell session:
    - Can be listed with command **env**.
  - **System variables:** common to every shell and other programs and users:
    - Can be listed with command **set**.
- Environment variables can be modified:
  - csh-like shells (csh, tcsh, zsh): **setenv/unsetenv**:
    - Example: `$ setenv PATH /usr/local/bin:/bin:/usr/bin`.
  - sh-like shells (sh, ksh, **bash**): **export**:
    - Example: `$ export PATH=/usr/local/bin:/bin:/usr/bin`.
  - After leaving a session, all modifications are lost.

# Environment Variables

- Shell configuration files:
  - Objective: give a value to environment variables. Allows the permanent modification of shell aspect and behavior (changes are not lost).
  - Bash loading sequence (last file overwrites the rest):
    - `/etc/bashrc` → `/etc/profile` → `$HOME/.bashrc` → `$HOME/.bash_profile`.
    - Different for each kind of shell.
  - File example(bash):

The alias command allows command re-definition (more friendly shell).



```
# .bashrc
# User specific aliases and functions
alias rm='rm -i'
alias cls="clear"
alias cd..="cd .."

# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi

alias ls="ls --color -shaF"
```

# Environment Variables

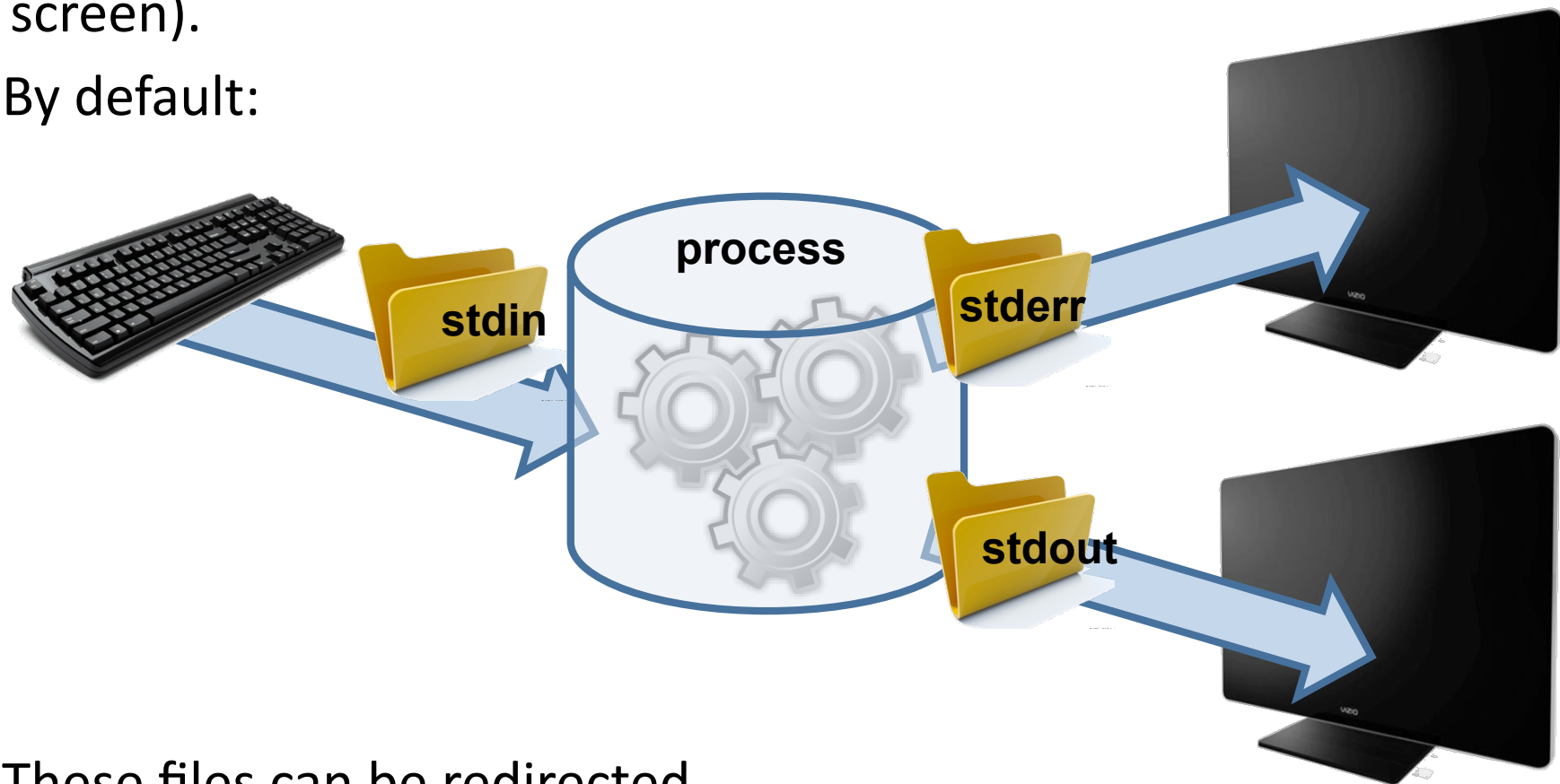
- Some important internal variables:
  - **\$PATH**: indicates which are the directories where binaries can be found. Before executing a command, the shell searches in those directories.
  - **\$HOME**: root directory of current user.
  - **\$TERM**: kind of terminal we are employing to connect to the system.
  - **\$SHELL**: user shell. Ex. /bin/bash.
  - **\$TZ**: time zone. Has an influence on the timing format returned by date command. Any change in our files adjusts to the time zone specified by that variable.
  - **\$\*\*\*\***: in the man page of each shell we have the complete repertory of its environment variables.

# Index

- The shell.
- File System.
- “Shortcuts”.
- User management.
- Environment Variables.
- **Redirection and Pipes.**
- Shell Scripting.
- Process management.
- Advanced Administration commands.

# Redirection and Pipes

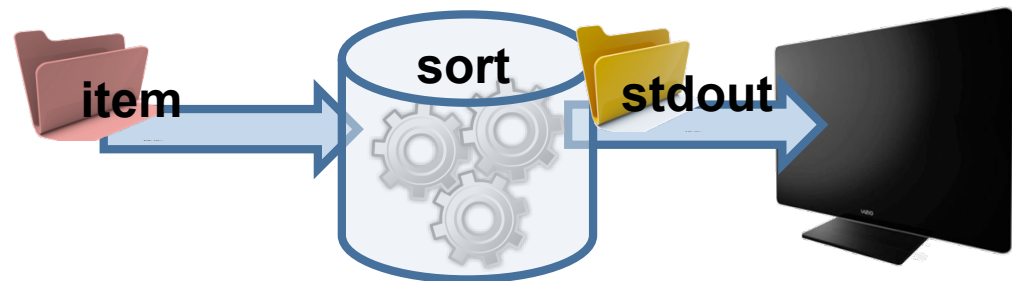
- In linux, always three files (remember, devices treated as files) opened by default: **stdin** (keyboard), **stdout** (screen) y **stderr** (also screen).
- By default:



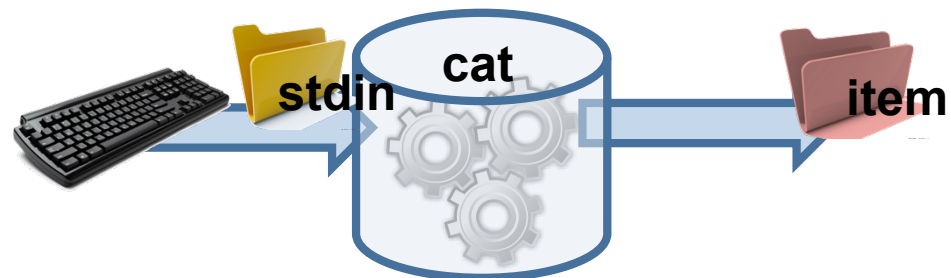
- These files can be redirected.

# Redirection and Pipes

- **Definition:** redirection consists of the capture of a file/command/program output in order to send it as input to another file/command/program.
- Standard **input** redirection: do not use keyboard as input:
  - Syntax: `$ sort < item`.

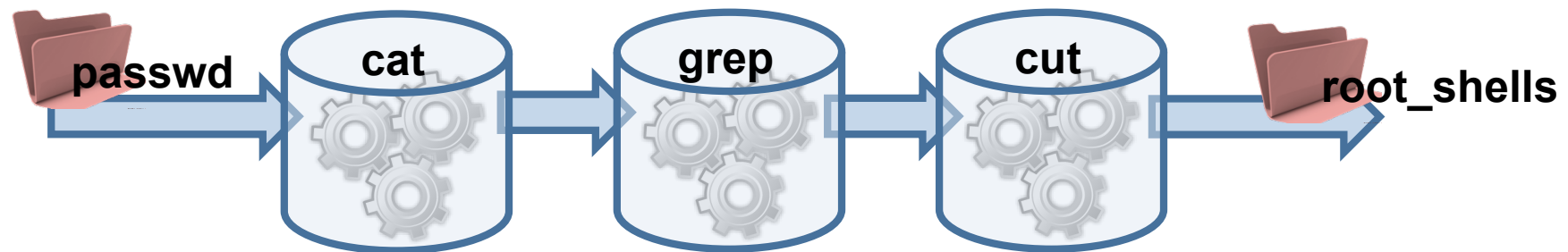


- Standard **output** redirection: output to a file (instead of the screen):
  - Syntax: `$ cat > item` (without overwriting item content: `$ cat >> item`).



# Redirection and Pipes

- **Pipes:** allows two or more commands to be linked, where the output of a command is redirected to be the input of the following one:
  - Example: `cat < /etc/passwd | grep root | cut -d : -f 7 > root_shells`.



- **Concatenation:** concatenate command in the same line:
  - Example: `ls -l; cd ..; ls -l` (also this way: `ls -l && cd .. && ls -l`).
  - **Nested** execution: `(ls -l; cd ..); ls -l` Difference with previous one?

# Index

- The shell.
- File System.
- “Shortcuts”.
- User management.
- Environment Variables.
- Redirection and Pipes.
- **Shell Scripting.**
- Process management.
- Advanced Administration commands.

# Shell Scripting

- Group shell commands to perform complex tasks in a single step.
- The simplest structure: text file with a command per line, but usually much more complex:
  - Conditional sentences, loops, functions...
- A script can create a sub-shell regardless of the one that executed it:
  - \$ bash macro.
- Its structure depends on the shell we are employing.
- Example:

```
#!/bin/sh
echo "Today is:"
date
echo "have a nice day"
```

- First line indicates which kind of shell executes the rest of the script.

# Shell Scripting

```
# run ./script [name]
#!/bin/sh
echo -n "Your surname? "
read surname
echo "Hola, $1 $surname"
```

- **Execution:**
  - If the file does not have execution permissions: `$ bash script`.
  - Other way, modify permissions: `chmod a+x script; ./script`.
- **Input/Output:**
  - Read from keyboard with command **read**.
  - Write in screen with **echo/printf**:
    - `echo -n` suppress newline.
- **Command line arguments:**
  - Become variables whose names are numbers:
    - `$1 - 9`: command line parameters, number indicates its position.
    - `$0`: macro name(script name).
    - `$#`: number of command line argument.
    - `$?: $$`: PID associated with the macro.
    - `$*`: string containing all the arguments passed (beginning with `$1`).

# Shell Scripting

- **Control Flow:**

- Sequence **if then else (elif)**:

- /bin/sh: if [<condition>] && [<condition>]; then.
    - /bin/bash: if [[<condition> && <condition>]]; then.

- Bash comparison operators

| String | Numeric    | True if          |
|--------|------------|------------------|
| x = y  | x -eq y    | x is equal to y  |
| x != y | x -ne y    | x not equal to y |
| x < y  | x -lt y    | x is less than y |
| ...    | le, gt, ge | ...              |

- ... and Bash file evaluation operators:

| Operator | True if                            |
|----------|------------------------------------|
| -d file  | File exists and is a directory.    |
| -e file  | File exists.                       |
| -f file  | File exists and is a regular file. |
| -r file  | You have read permission on file.  |
| -s file  | File exists and is not empty.      |
| -w file  | You have write permission on file. |

- Sequence **case**:

```
rental=$1
case $rental in
    "car") echo "rent car";;
    "moto") echo "rent moto";;
    "bus") echo "rent bus";;
esac
```

```
if [ who | grep -s pepe > /dev/null ]
then
    echo "pepe is in the system"
else
    echo "not present"
fi
```

# Shell Scripting

```
for archivo in `ls`  
do  
    touch ${archivo}  
    echo "archivo ${archivo} update"  
done
```

- **Loops:**

- Sequence **for**:

- List of arguments: for files in fich1.sh fich2.sh fich3.sh; do.
    - Pattern matching expansion: for files in \*.sh; do.
    - Command outputs the list: for files in `ls`; do.

- **Variables:**

- All bash variables are string valued:

- Declaration: a = pepe, Utilization: echo "\$a".

- Arithmetic with variables??: (( )):

- Operation in (( )) is arithmetic, otherwise only concatenated strings.

- Arrays:

- Declaration: list = (aa bb cc dd). Utilization: echo "\${list[num]}" (begins at zero).

```
a=1  
b=$((2))  
c=$a+$b  
d=$a$b  
e=$(( $a+$b ))  
echo "$c $d $e"
```

# Shell Scripting

- **Regular expressions:**

- Employed to match a text string to a pattern (semi-generic pattern).
- Pattern: built through a mix of literal and special characters.
- Examples:

- “pepito” matches “pepito”.
- “p([a-z]+)o”:
  - pepito / pablo / po / p1o.
- “p(\d\*)o”:
  - pepito / p10o / po / pablo.
- “^p(\w\*)o\$”:
  - pepito / hola pepito / pasa julio.

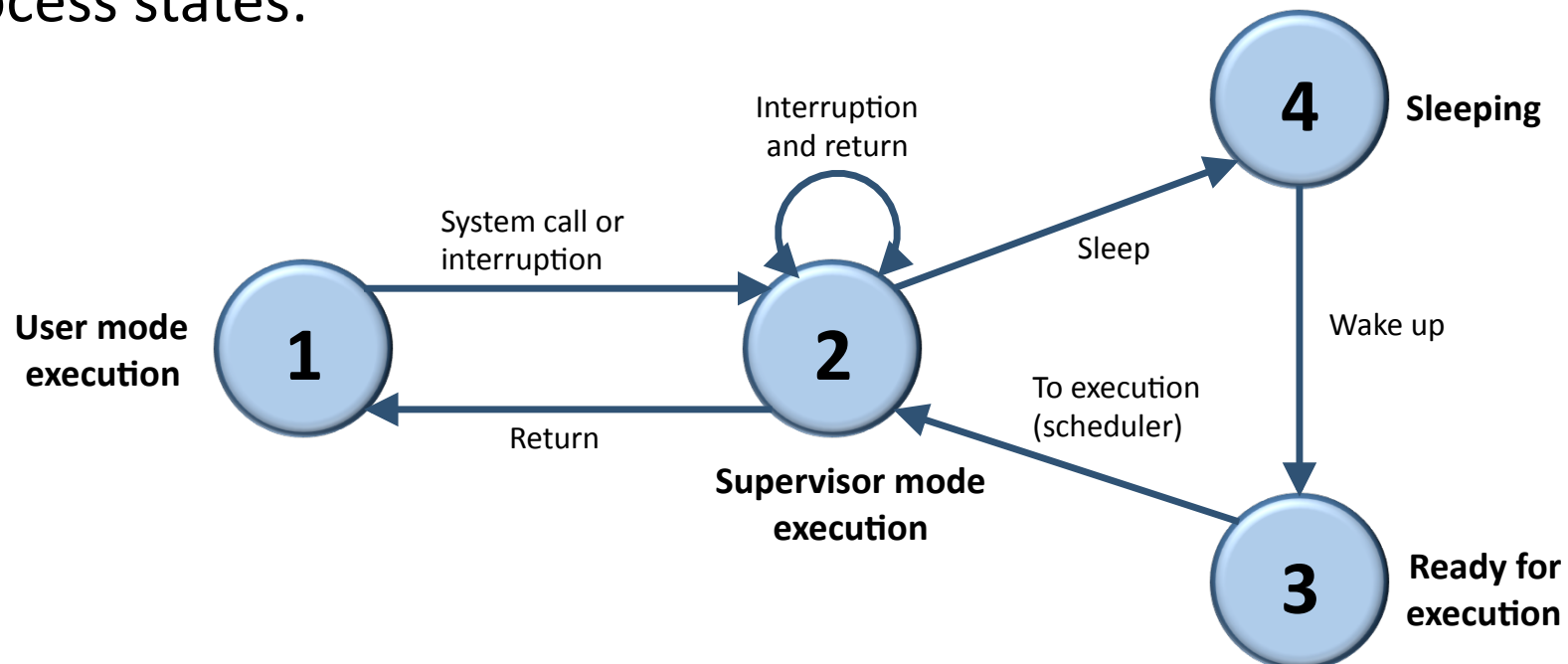
| Symbol  | What it matches or does                                |
|---------|--------------------------------------------------------|
| .       | Matches any character.                                 |
| [chars] | Matches any character from a given set.                |
| ^       | Matches the beginning of a line.                       |
| \$      | Matches the end of a line.                             |
| \w      | Matches any word character ([A-Za-z0-9_]).             |
| \s      | Matches any whitespace character (space, tab, return). |
| \d      | Matches any digit.                                     |
|         | Matches either element to its left or to its right.    |
| (expr)  | Limits scope, group elements, capture matches.         |
| ?       | Allows zero or one match of the preceding element.     |
| *       | Allows zero, one or many matches of preceding element. |
| +       | Allows one or more matches of preceding element.       |
| {n}     | Matches n instances of preceding element.              |

# Index

- The shell.
- File System.
- “Shortcuts”.
- User management.
- Environment Variables.
- Redirection and Pipes.
- Shell Scripting.
- **Process management.**
- Advanced Administration commands.

# Process Management

- **Process:** sequence of instructions and data stored in memory able to perform some specific task.
- Unique ID (numerical) in the system: **PID**.
- Three main memory segments: code/data/stack.
- Process states:



# Process Management

- Processes have a hierarchy similar to the file system (tree). Root process: **init**:
  - Each process (excluding init) has a father process.
  - The kernel (root) has absolute control of every system process.
- A process can be identified by its PID:
  - Only its owner can interact with that process (UID).
- The shell is a process, dependent on the terminal:
  - Foreground process: blocks shell utilization until it finishes execution:
    - `$ ls -R / >/dev/null.`
  - Background process: does not block shell:
    - `$ ls -R / >/dev/null &.`
  - Processes can be moved between foreground and background:
    - **[Ctrl+z]**: foreground process stopped (suspend execution).
    - **bg** moves process to background mode and **fg** moves it back to foreground.

# Process Management

- /proc: pseudo file system associated with the processes:
  - Employed as interface to the data structures in the kernel associated with each process.
  - Content example (one folder for each process):

```
[ root si /tmp ] ls /proc/
1      211    2428  2490  2600  41     7      bus      execdomains  kallsyms    misc        scsi         timer_stats
1076   212    2439  2497  2603  42     741    cgroups   fb           kcore       modules     self         tty
1153   213    2440  2512  2605  4769   742    cmdline  filesystems  key-users   mounts      slabinfo     uptime
1620   2318   2459  2521  2618  4772   774    cpuinfo   fs           kmsg        mpt         stat         version
1687   2329   2465  2532  2691  5       775    crypto    ide          kpagecount  mtrr        swaps        vmallocinfo
173    2339   2468  2566  2719  5280   958    devices  interrupts   kpageflags  net         sys          vmstat
2      2397   2470  2594  3      5282   acpi     diskstats   iomem       loadavg     pagetypeinfo sysrq-trigger zoneinfo
2099   2410   2483  2596  39     5387   asound   dma         ioports     locks       partitions  sysvipc
210    2420   2489  2598  4      6      buddyinfo driver        irq         meminfo     sched_debug  timer_list
```

- In each folder...:

```
[ root si /tmp ] ls /proc/2719
attr      clear_refs      cpuset  exe      io      maps      mounts      oom_adj  root      smaps  status
auxv      cmdline         cwd     fd       limits  mem       mountstats  oom_score sched    stat   task
cgroup    coredump_filter environ  fdinfo   loginuid mountinfo  net         pagemap  sessionid statm  wchan
```

- **fd**: files opened by the process.
- **maps**: physical memory range associated with the process.
- **stat**: current process status: PID, PPID, utime, etc.

# Process Management

- See Appendix for detailed description.
- Process Management commands:
  - Command **top**: process monitoring in real time.
  - Command **ps**: reports information about active processes.
  - Command **kill**: send signals to a process.
  - Command **pstree**: hierarchical relations among processes.

# Index

- The shell.
- File System.
- “Shortcuts”.
- User management.
- Environment Variables.
- Redirection and Pipes.
- Shell Scripting.
- Process management.
- **Advanced Administration commands.**

# Advanced Commands

- Command **sed**: perform text modifications in an input file:
  - Line by line analysis.
  - Syntax: `sed -<opts> '[instruction]' [file]:`
    - Option **-i**: in place, the file passed as argument is overwritten.
  - Some useful instructions:
    - **i**: insert line before current one.
    - **p**: print current line in stdout.
    - **s**: replace string in current line.
  - Examples:
    - `sed -i 's/Pepe/Manolo/g' *.txt` replace pepe by manolo in every .txt file.
    - `sed '/cadena/ s/vieja/nueva/g' file > salida` only replace in lines containing the string (flag g: perform the change in every matching).
    - `sed '2,3 p' *` print lines 2 and 3 of every file.
    - `sed -i '/cadena/d' archivo` remove string from file named archivo.

# Advanced Commands

- Command **xargs**: run the same command over a list of arguments separated by a space (or different lines):
  - Syntax: [Commands...] | xargs -<options> [command]:
    - Option **-i**: replace string.
    - Option **-n**: group the items.
  - Example:
    - `$ ls *.c | xargs -i gcc -c {}.`
    - `$ ps -ef | grep "pepito" | awk '{print $2}' | xargs renice +10.`
  - **-i** and **-n** may cause conflict when used together, last one “wins”:
    - `$ echo a b c d e f | xargs -n3 -i echo before {} after:`
      - before a b c d e f after.
    - `$ echo a b c d e f | xargs -i -n3 echo before {} after:`
      - before {} after a b c.
      - before {} after d e f.

# Advanced Commands

- **Awk** programming language: oriented to file processing:
  - Line-oriented (file is analyzed line-by-line).
  - Basic format of an awk program is: **pattern { action }**:
    - Pattern determines when to perform action.
    - If pattern condition returns true, in that line action is performed.
    - If pattern is left empty, action is performed in every line.
  - Awk variables:
    - \$N: this var contains the N field of the line (default field separator: space).
    - \$0: variable containing the whole line.
    - FS: determines a different field separator (option -F).
    - NF: contains the number of fields in a line.
    - NR: contains the line number.
  - Examples:
    - `awk -F: '{if($2=="") print $1": no password!"}' < /etc/passwd.`
    - `awk '{ if(NR>100) print NR, $0}' < fichero.`

# **APPENDIX**

# File System (Commands)

## Navigating through the file system

- Command **pwd**: display the path of the current folder.
- Command **cd**: command to move to a different location.
  - Usually, each session starts at the home directory of the user.
  - Syntax: `$ cd [directory]`:
    - The destination folder can be expressed as an absolute path (from root: `cd /home/pepe/`) or as a relative path (from current folder: `cd ../usr/bin`).
    - If no destination is specified, the command moves to the `$HOME` dir of the user.
- Command **mkdir**: create a new directory:
  - Syntax: `$ mkdir -<options> directory`:
    - Option **-m**: establish the permissions of the created folder.

# File System (Commands)

## File Manipulation

- Command **ls**: one of the most employed. List the content of a directory alphabetically:
  - Syntax: `$ ls -<options> [file...]`:
    - If executed without arguments, list files and folder of current directory.
    - Option **-a**: include hidden files (starting with `.`) to be listed.
    - Option **-l**: detailed view (permissions, links, owner, group, size, modification date).
    - Option **-r**: opposite order for the list.
    - Option **-t**: order the list by modification date.
    - Option **-S**: order the list according to file size.
    - Option **-s**: show the size of each file.
    - Option **-A**: list all files except `."` y `."`
    - Option **-R**: list the content of every folder recursively.
    - Option **-color** = [none/auto/always]: use colors for different file types.
  - Combined example: `$ ls -lart` What does this command do?

# File System (Commands)

## File Manipulation

- Command **cp**: copy files:
  - Syntax: `$ cp -<options> [arch_1]...[arch_n] [destination-dir]:`
    - Option **-f** (forced): overwrite destination file with the same name.
    - Option **-i** (interactive): opposite to **-f**, ask before overwriting.
    - Option **-p**: maintain permissions, user and group.
    - Option **-R**: copy directories recursively.
    - Option **-a**: equivalent to **-pR**.
    - Option **-u**: do not perform the copy if in destination folder there is a file with the same name and it is more recently modified.
    - Option **-v** (verbose): display information about the copy process.
- Command **mv**: move files (not copy) and/or rename:
  - Syntax: `$ mv -<options> [source_1]...[source_n] [destination]:`
    - If the last argument is a directory, each source file is moved to that directory.
    - If source and destination are files, file is renamed.

# File System (Commands)

## File Manipulation

- Command **rm**: remove files and folders:
  - Syntax: `$ rm -<ops> [file]...`:
    - Warning: use with care.
    - The argument [file] can be a file, a folder or a regular expression.
    - Option **-f** (forced): without error messages, without requesting confirmation.
    - Option **-r** (recursive): remove folder content recursively.
- Command **ln**: links between files:
  - Two types, static or symbolic.
  - Syntax: `$ ln -<ops> [src] [dst]`:
    - Option **-d**: allows superuser to perform static links to folders.
    - Option **-s**: create a symbolic link.
  - Example: `$ ln -s /etc/passwd /home/usuario/claves`.
  - Running `ls -l` in a folder with symbolic links:
    - `lrwxrwxrwx 1 usuario usuario 11 Apr 8 13:33 claves -> /etc/passwd`.

# File System (Commands)

## File Manipulation

- Command **whereis**: find the path of a binary/source code/manual of a command:
  - Syntax: `$ whereis -<options> [file]...:`
    - Option **-b**: look only for the binary file.
    - Option **-m**: look only for the man page.
    - Option **-s**: look only for the source code.
- Command **locate**: command for file searching:
  - Performed through an indexed database (speed). One file with a list of every file in the file system.
  - `/var/lib/mlocate/mlocate.db`.
  - Usually, the OS runs a command periodically to update this database.
  - Syntax: `$ locate -<options> [pattern]`.

# File System (Commands)

## File Manipulation

- Command **find**: powerful command for file searching:
  - Basic for administration. Allows filtering searches and running actions on the result.
  - Syntax: `$ find <starting_point> -<filters> -<action>`.
  - Filters:
    - **-atime n**: only search for files opened n days ago (+n: more than n days ago).
    - **-mtime n**: file modified n days ago (+n...).
    - **-newer file**: files modified after file.
    - **-size n**: files with n-blocks size (block = 512 bytes) (+n...).
    - **-type c**: type of file (f = text, d = directory, etc.).
    - **-fstype type**: file type \*.type.
    - **-name nam** name = nam.
    - **-perm p**: with permission p.
    - **-user usr**: owner usr.

# File System (Commands)

## File Manipulation

- Command **find** (continued):
  - The search filters can be combined:
    - To force precedence: `\( ... \)`.
    - Condition AND: `-atime + 60 -mtime + 120`.
    - Condition OR: `-atime + 7 -o -mtime + 120`.
    - Condition NOT: `! -name gold.dat`.
  - Actions on the files found:
    - Action **-print**: display all the files found.
    - Action **-ls**: display with extended format.
    - Action **-exec cmd\;**: run a command on every file (without asking)
    - Action **-xdev**: only search in current file system.
  - Some examples:
    - `$ find /home -size +2048 \( -mtime +30 -o -atime +120 \) -exec ls {} \;`
    - `$ find /home -fstype f -name core -exec rm -f {} \;`
    - `$ find /home/pepito -name '*.c' -exec mv {} /home/pepito/src \;`

# File Content (Commands)

- Command **cat**: display the content of a file in a single step:
  - Not useful with large files.
- Command **more**: show the content progressively (paging):
  - Number of paging lines same as terminal size.
- Command **less**: evolution of the more command:
  - Interactive, with its own commands (launched through a key or a key combo):
    - **Space bar**: advance a number of lines equal to the terminal.
    - **Cursors**: move fw/bw line by line.
    - **G/g**: go to the beginning/end of the text.
    - **/pattern**: enter a string to search in the file.
    - **n/N**: move to the next/previous result of the work searched.
    - **AvPag/RePag**.
    - **q**: exit the program.

# File Content (Commands)

- Command **wc**: count the words in a file:
  - Syntax: `$ wc -<opts> [file...]`:
    - Option **-c**: count bytes.
    - Option **-l**: count lines.
    - Option **-w**: count words.
- Command **head**: display the first part of a file:
  - Syntax: `$ head -<options> [file]`:
    - Option **-c N**: display the first N bytes.
    - Option **-n N**: display the first N lines (10 by default).
- Commando **tail**: display the last part of a file:
  - Syntax: `$ tail -<options> [file]`:
    - Options **-c** and **-n**: same as head.
    - Option **-nf**: display the last part of a file as it grows. Very useful to control log files that grow over time.

# File Content (Commands)

- Command **grep**: display those lines matching a pattern:
  - Syntax: `$ grep -<opts> PATTERN [files...]`:
    - Option **-c**: display the number of lines matching the pattern.
    - Option **-H**: display the name of the file on every match.
    - Option **-r**: search recursively inside the folders of the current directory.
  - When the patterns contains special characters ( space, -, etc.), `""` can be employed.
  - Regular expressions can also be used:
    - Example: search for lines with words starting with a: `grep a* file`.
- Command **tar**: add the content of a whole directory tree to a single file:
  - Not compressed, only packaged.
  - tar: `$ tar -cvf fichero.tar /path/`.
  - untar: `$ tar -xvf fichero.tar`.
  - working with **gzip** (compressor): `$ tar -czvf fichero.tar.gz /path/`.

# File Content (Commands)

- Command **cut**: remove sections from each line of a file:
  - Syntax: `$ cut -<opts> [files...]`:
    - Option **-c N**: select the Nth character of each line (-N: from the beginning of the line to N).
    - Option **-b N**: select the Nth byte of each line (M-N: from byte M to N).
    - Option **-f N**: select the Nth field. Default delimiter: TAB.
- Command **sort**: sort the lines of a text file:
  - Syntax: `$ sort -<opts> [file]`:
    - Option **-d**: alphabetic order.
    - Option **-n**: numeric order.
    - Option **-b**: ignore blank spaces at the beginning of the line.

# File Content (Commands)

- Command **vi**: text editor (terminal) included in every UNIX system:
  - A bit difficult for beginners. With practice, much faster than any graphic editor (in some cases it might be the only option).
  - Some improved versions available, such as **vim**, which are more friendly.
  - **Command mode**: exit, save, copy, search, etc.
  - **Edition mode**: text insertion.
  - From command to edition: [i], [a], [o], [O]...
  - From edition to command: [Esc].
  - **Moving through the text**:
    - [h],[l],[j],[k]: cursor; left, right, up, down (in vim cursors work...).
    - [G]: go to the last line ([5G]: go to the 5th line).
    - [0][\$]: go to the beginning (zero)/end of the line.
  - Entering the edition mode:
    - [a][i]: append or insert.
    - [o][O]:

# File Content (Commands)

- Command **vi** (continued):
  - **Entering edition mode:**
    - [a][i]: append or insert.
    - [o][O]: open above/below a line.
  - **Edition (managing the buffer):**
    - [x]: remove a character ([xx] remove a line, [4xx] remove 4 lines, [xw] remove a word).
    - [d]: cut ([dd] cut a line...).
    - [y]: copy ([yy]...).
    - [p]: paste.
    - [r]: replace.
    - [u]: undo.
    - [Ctrl+r]: redo.
    - [.]: repeat the last command.
  - **Search:**
    - Similar to less ([/pattern], [?pattern], [n], [N]).

# File Content (Commands)

- Command **vi** (continued):
  - **Replace:**
    - [%s/old/new/g]: replace old string by new string throughout the whole text.
  - **Exit:**
    - [:w]: save changes, without exiting.
    - [:q]: exit (fails if unsaved changes are found).
    - [:q!]: forced exit, unsaved changes are lost.
    - [:wq]: save and exit.

# User Management (Commands)

- Command **whoami**: displays the name of the user running the command.
- Command **who**: displays the users logged in the system.
- Command **passwd**: change user's password:
  - Syntax: `$ passwd [user]`:
    - If no user is specified, it makes use of the one using the command.
- Command **finger**: shows the status of a user in a system:
  - Syntax: `$ finger user@system`:
    - Shows user information, session time, inactivity time, mail, .plan file.
- Command **write**: send text messages to the terminal of a connected user:
  - Syntax: `$ write user [tty]`.
  - Complementary command **wall**: write to all (every user connected).
  - Command **talk** *user@machine*: establish complete communication (~IRC).

# User Management (Commands)

- Command **chmod**: modify permissions of a file/folder:
  - Syntax: `$ chmod [ugo] [+ -] [rwx] [file or directory]`:
    - Option **-R**: recursive.
    - Example: limit the access to \$HOME to every user.
    - `$ chmod -R g-rwx, o-rwx $HOME`.
    - Permissions can be coded in octal/binary: `chmod -R 700 $HOME`.
- Command **chown/chgrp**: modify the UID/GID of a file:
  - Syntax: `$ chown [-R] new_user file`
- Command **umask**: modify the permissions assigned to new files by default:
  - Syntax: `$ umask [inhibition code]`:
    - Establishes which bits are 0 when creating the file.
    - Example: `$umask 022` -> permissions for the created files: 644 (rw-r--r--).
- To run a file, execution permissions activated (x). Extensions (.exe) NOT necessary.