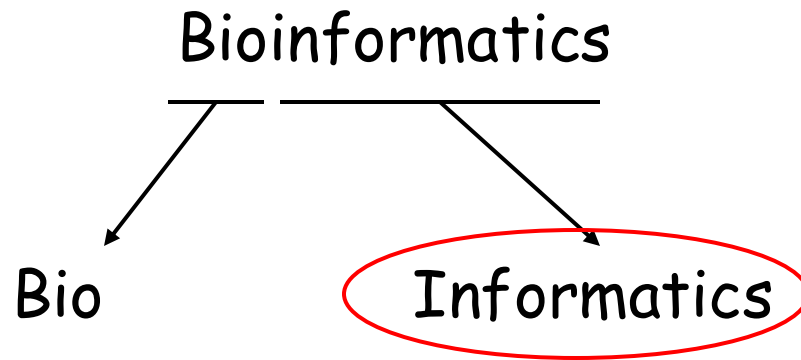


Computer: The Tool in



Why this class?

Some basic computer knowledge must be acquired in order to use computers in science:

Operative Systems (OS): what they are and their differences

Understanding the directory tree

Understanding the different kinds of files

Working with the command line

Understanding processes and programs

Unix / Linux has become the most used OS in protein crystallography and bioinformatics

Different Graphic User Interface (Windows)

Very powerful command line interface tools

Processor (AKA: C.P.U.): computer's brain

-Electron Current
(there is or there is not)

-Binary code

▪ $\left. \begin{matrix} 0 \\ 1 \end{matrix} \right\} \dots\dots\dots \text{Bit}$

▪ 01001110 Byte (b)

▪ $(1000 = 10^3) * \text{Byte} \dots\dots \text{Kilobyte (Kb)}$

$(1024 = 2^{10}) * \text{Byte} \dots\dots \text{Kibibyte (KiB)}$

▪ $(1000 = 10^3) * \text{Kb} \dots\dots \text{Megabyte (Mb)}$

$(1024 = 2^{10}) * \text{KiB} \dots\dots \text{Mebibyte (MiB)}$

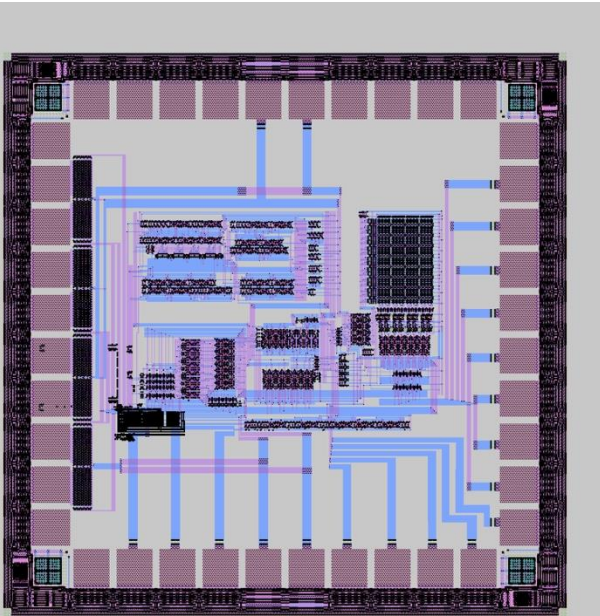
▪ $(1000 = 10^3) * \text{Mb} \dots\dots \text{Gigabyte (Gb)}$

$(1024 = 2^{10}) * \text{MiB} \dots\dots \text{Gibibyte (GiB)}$

▪ $(1000 = 10^3) * \text{Gb} \dots\dots \text{Terabyte (Tb)}$

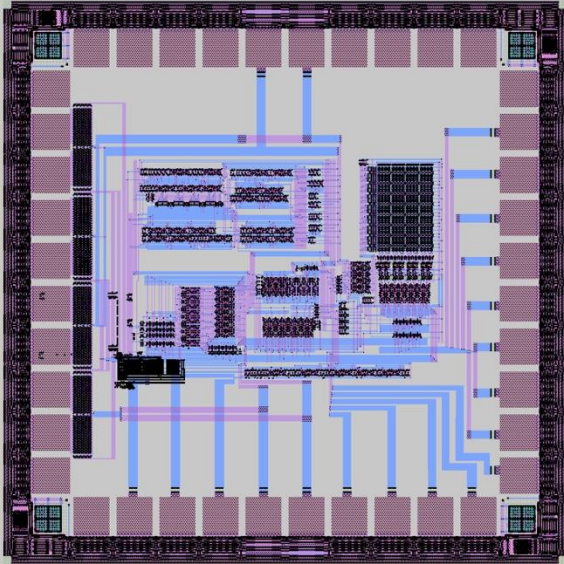
$(1024 = 2^{10}) * \text{GiB} \dots\dots \text{Tebibyte (TiB)}$

-Architecture: x86 (AKA: i386), x86_64,
~~IA64, PPC, SGI MIPS, SUN-SPARC, ARM ,~~
RISCV...



Processor (AKA: C.P.U.): computer's brain

multiprocessor



A processor can only execute one instruction at a time.

If multiple process are running at the same time the processor has to stop one task to do the other. If both process have the same priority will follow something like this:

1st - process 1 instruction 1
2nd - process 2 instruction 1
3rd - process 1 instruction 2
4th - process 2 instruction 2

...

As you can deduce, this makes your process take double time to complete.

You can have multiple physical processors in the same computer. But currently, all processors contain multiple processors inside them (AKA: cores).

This way you can execute multiple tasks simultaneously without having to share its time between them. Some processors contain multiple logical units which appear as different processors to the OS, but they are not. Depending on the task using them can make your task faster (In general: memory intensive, low number of calculations) or slower (In general high number of calculations low memory movements). But this is highly dependent on the program, the task, and lots of factors, the only way to know it is to test it.

Execute:

```
cat /proc/cpuinfo|less
```

look at:

```
processor : ?
```

```
physical id : ?
```

```
core id : ?
```

How many logical cores your computer have?

How many physical processors your computer have?

How many cores your computer have?

Execute:

```
top
```

press "1"

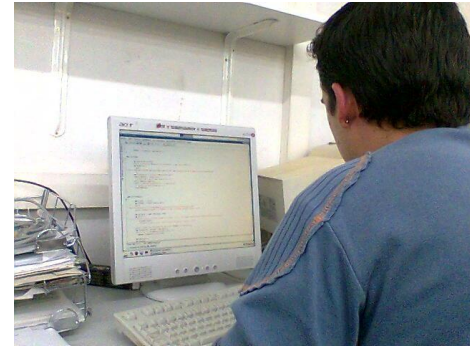
How many CPUs your operative system sees and is able to use?

How many is using?

Why processor understands me?

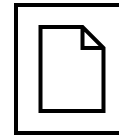
Programmer: Writes software with users in mind (or not)

```
main()  
{  
    printf "Hello world!!" ;  
}
```



Compiler: translates programmer's language to computer's language (0 and 1) generating **binary** programs executable by the processor

"R" = "01010010"



Operative System: allows interaction with the processor in a unified and easy way

"R" = "01010010"



Operative System, What is it and why we need it

An **Operative System** is a group of programs or software, devoted to allow interaction between users and computer, it manages its resources in an easy and efficient way. It starts to work when computer boots and manages hardware from lowest levels.

- **Device** access
- Input/Output (I/O)
- **Directory tree / File System**
- Process priority
- Users

Operative System : some parts (private ones)

-Kernel: The OS itself, it's the one who decide what to do, how to do it and when to do it, it controls devices and peripherals at the lowest hardware level. (linux, NT Kernel, XNU,)

Windows : C:\WINDOWS\system32\ntoskrnl.exe
Linux: /boot/vmlinuz-*

-Filesystem: Keeps hierarchy, easy access and consistency of stored data in the different physical or virtual devices (eg: fat32, ntfs, APFS, ext4, xfs, zfs, nfs, ...)

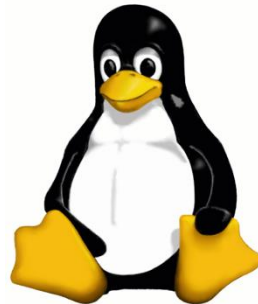
-shell(s): programs that provide the interface between the user and the operating system. Shells may be text-based, such as **bash** or **zsh** (in both MacOS and linux) or graphical, such as the **Windows Shell** or the **macOS Finder**.

-Various tools : group of programs that allow users and / or administrators interact with the OS

Operative System: shapes, kinds and colors



Linux



Windows



VMS



Macintosh

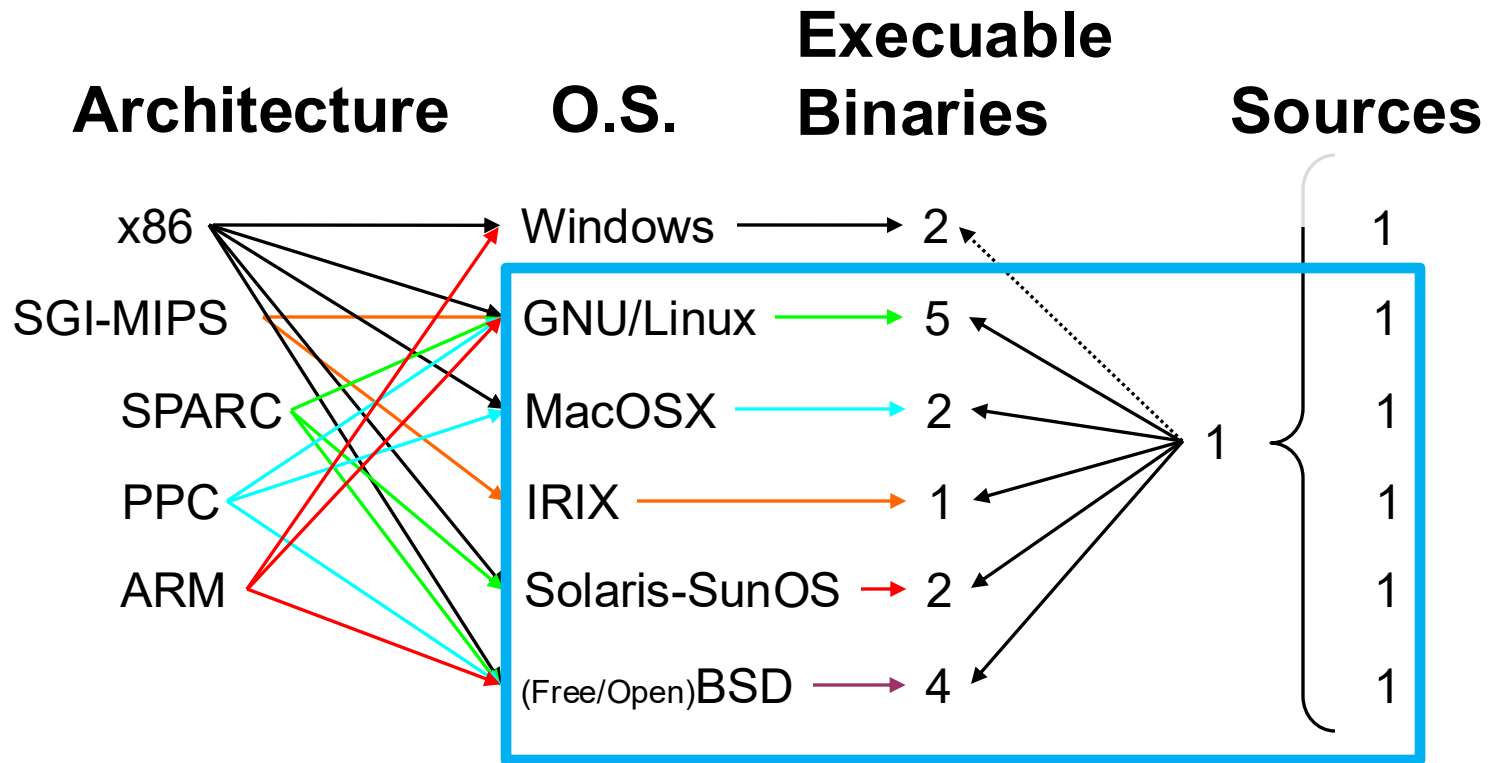


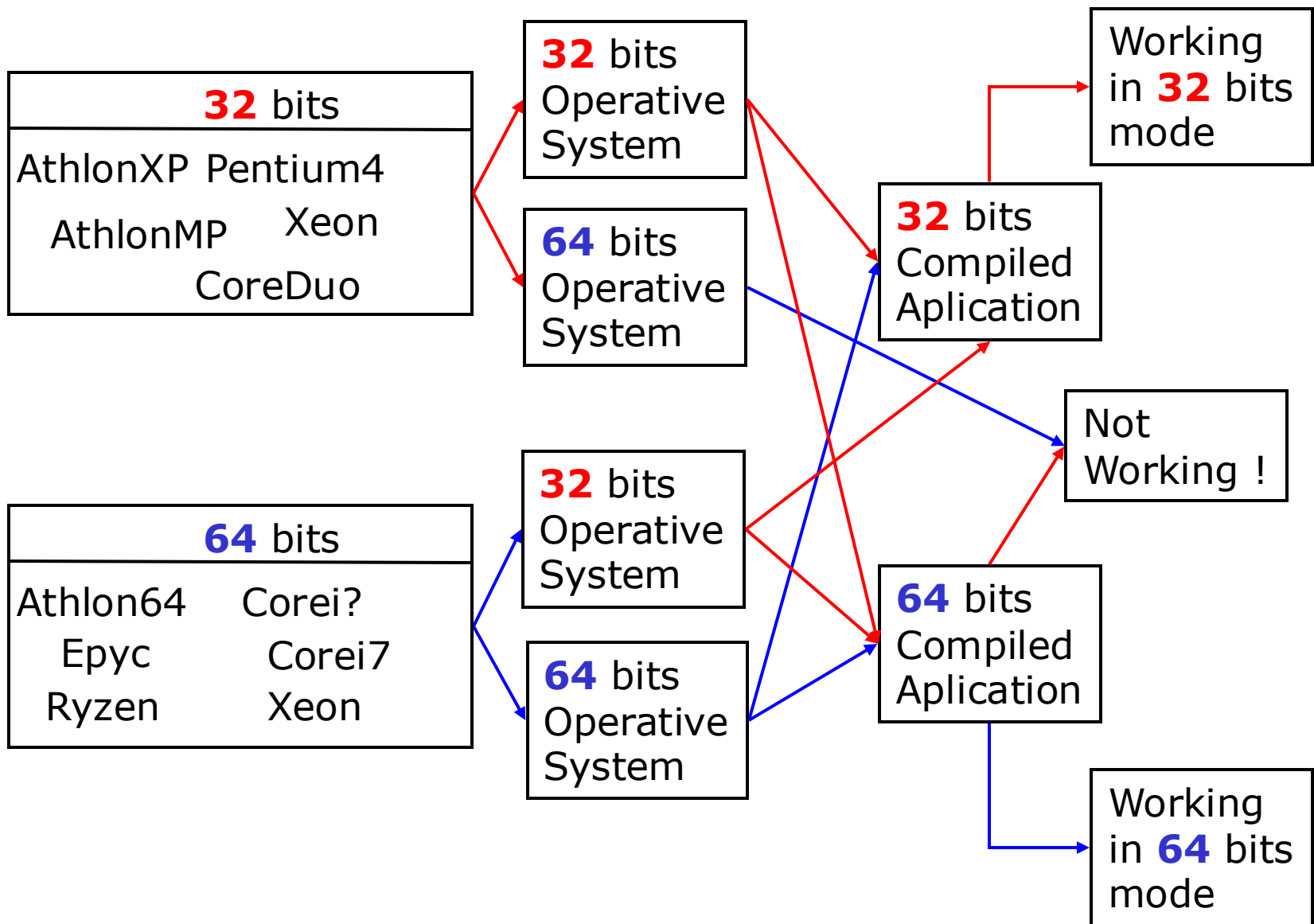
Mac OS

Symbian

symbian
OS

Plataforms





- According to function:

Types of Files

- Data files

binaryfile.doc



textfile.txt



They contain data that will be read by a program in order to use them

- Executables

winword.exe



TextEdit



notepad.exe

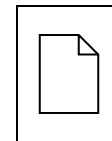


They contain instructions that the Operative System understands and can send to the processor in order to be executed.

- How data is stored in them:

- Text Files:

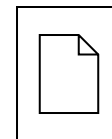
textfile.txt



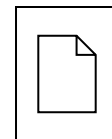
They contain only text, so can not be executed without the help of an external interpret

- Binary Files

binaryfile.doc



winword



They contain binary data, they can contain instructions understandable by the processor, so they can be executed without any external requirement (of course they still need the Operative system)

In a computer we find **storage units** (AKA Units or Volumes or disks or partitions or...) which store the data (the files) we use in a "none volatile" way, so when we turn off the power, the data remains.

Examples of storage units are: The Hard Disk, USB pendrive, DVD, Network drives, etc.

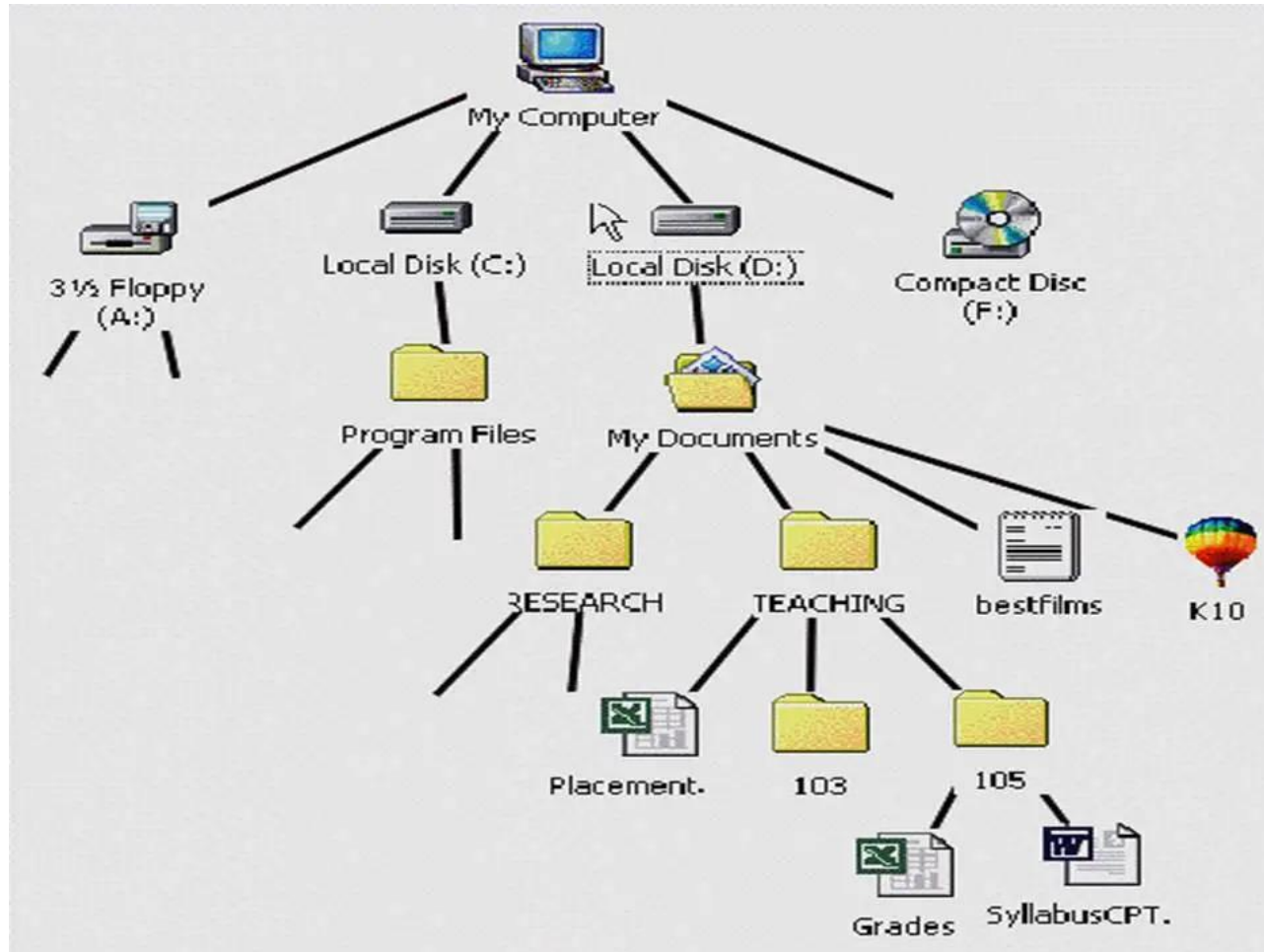
We organize the data inside these storage units, storing it in files, so the bytes that compound our data are stored in those files.

These files are organized in directories (AKA folders)

These folders are organized hierarchically in the **directory tree**

In Microsoft Windows:

Each storage unit receives a letter followed by ":" as a name.
It is **mounted** at the beginning of the directory tree.

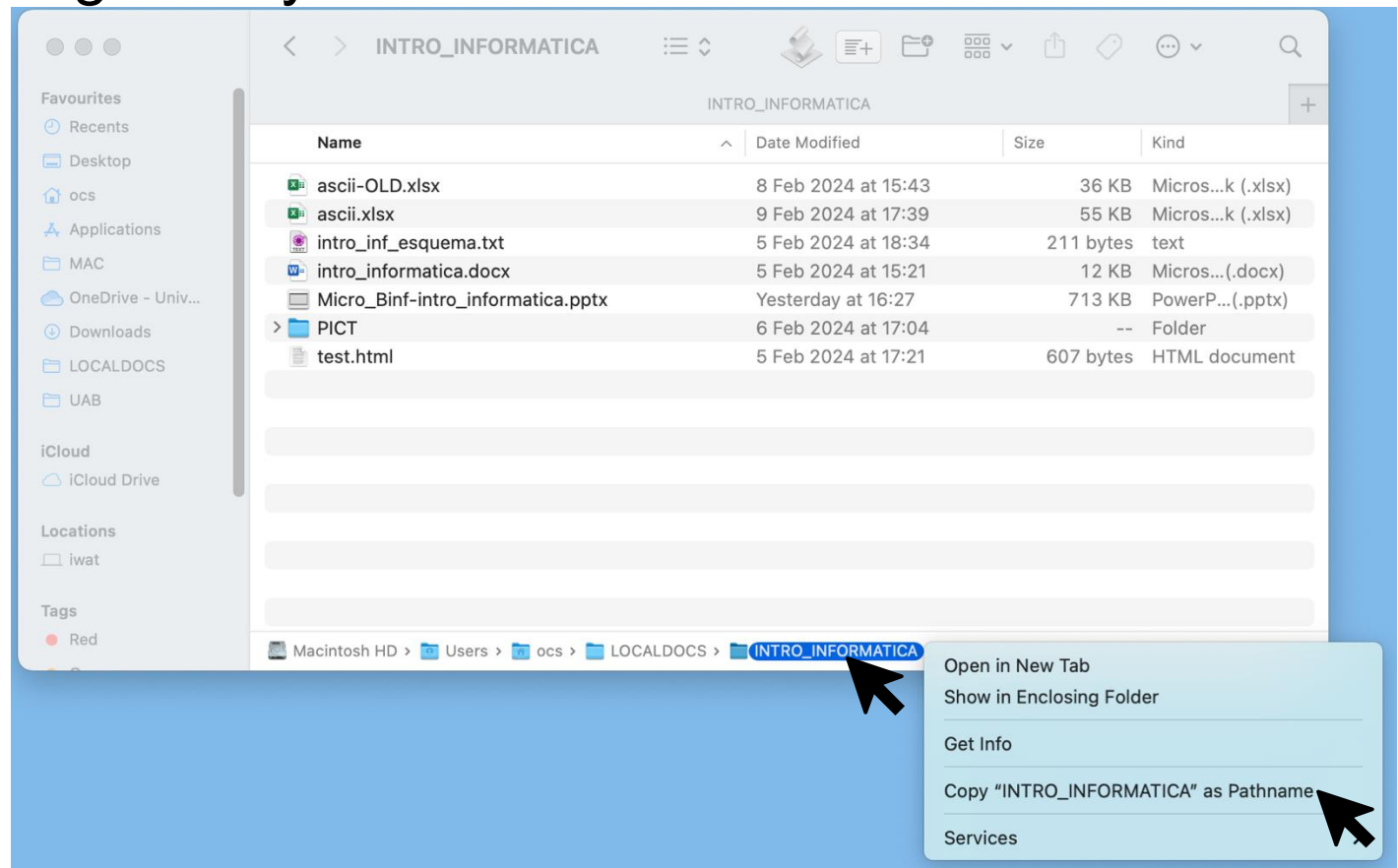


- ▼ Aquest ordinador
- ▼ Sistema (C:)
 - > SWINDOWS.~BT
 - ESD
 - > Fitxers de programa
 - > Fitxers de programa (x86)
 - > Heil Egregor
 - PerfLogs
 - Temp
- ▼ Usuaris
- ▼ 2055583
 - > .azcopy
 - .ms-ad
 - > Baixades
 - > Cerques

In MacOS:

Each storage Unit receives a given name. The one that contains the MacOS has the internal name of "/" and Becomes the start of the directory tree.

Any other storage unit you mount will be found as a folder inside this one:
/Volumes



This is what you would get:
/Users/ocs/LOCALDOCS/INTRO_INFORMATICA

In Linux:

The directory tree starts with the "root" "/" where a storage unit containing most of the necessary files to make the operative system work is mounted. Other storage units could be mounted in any folder of the directory tree. (execute "df -h" to see them).

```
ocs — ocs@rum:~ — ssh rumh — 80x24
~ — ocs@rum:~ — ssh rumh

[ocs@rum ~]$ df -h
```

Filesystem	Size	Used	Avail	Use%	Mounted on
devtmpfs	7.7G	0	7.7G	0%	/dev
tmpfs	7.7G	24K	7.7G	1%	/dev/shm
tmpfs	7.7G	1.7M	7.7G	1%	/run
tmpfs	7.7G	0	7.7G	0%	/sys/fs/cgroup
/dev/sda1	100G	21G	80G	21%	/
/dev/sda3	20G	4.2G	16G	21%	/var
/dev/sda5	1.7T	958G	721G	58%	/state/partition1
tmpfs	1.6G	12K	1.6G	1%	/run/user/42
10.2.2. [redacted]:/mnt/labdata/[redacted]	9.6T	7.5T	1.7T	82%	/home/[redacted]
tmpfs	1.6G	8.0K	1.6G	1%	/run/user/1132
tmpfs	1.6G	4.0K	1.6G	1%	/run/user/1154
10.2.2. [redacted]:/mnt/labdata/[redacted]	9.6T	7.5T	1.7T	82%	/home/[redacted]
tmpfs	1.6G	0	1.6G	0%	/run/user/0
zstore01:/ztank1/data1/data0	71T	22T	50T	31%	/share/Zdata0
10.2.2. [redacted]:/home	1.8T	838G	902G	49%	/sharelab/bioinf
10.2.2. [redacted]:/mnt/labdata/[redacted]	9.6T	7.5T	1.7T	82%	/home/[redacted]
10.2.2. [redacted]:/mnt/labdata/[redacted]	9.6T	7.5T	1.7T	82%	/home/[redacted]

```
[ocs@rum ~]$
```

Forget about Windows, Linux from now on!

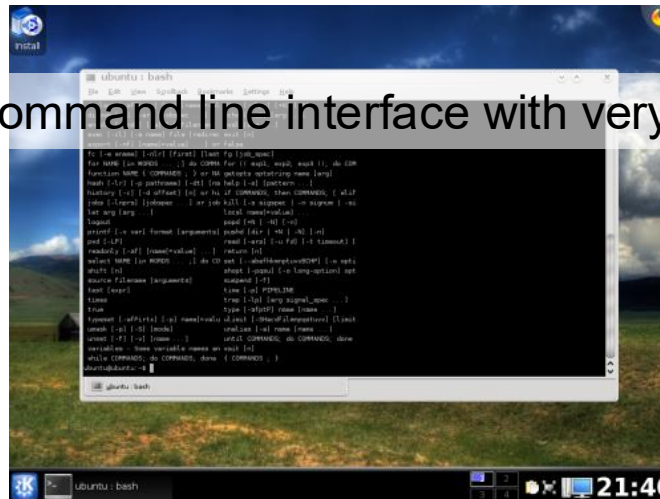
- Different Operative system, so requires different executables, but with the right application same data files can be read (mp3s, avi videos, MSOffice documents, images, etc).

- Different Graphic User interface.

- Multi user Operative system, so requires user identification



- Very powerful command line interface with very useful tools

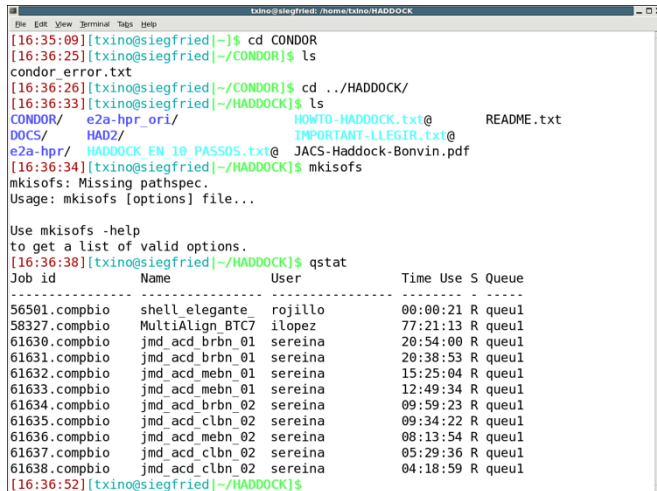


Operative System: why Linux?

- **Stability** (more than windows)
- **Multitask & Multiuser** (Real Multitask and real multiuser)
- **Application distribution** (Same source code can be compiled on all UNIX systems)
- **Libraries and compilers come installed whit the OS** (Compilation process is the same for all UNIX like platforms)
- **Command line Tools** (All UNIX have the same ones, but not necessarily working the same way)
- **Graphic Library (GL)** (Historical reasons, nowadays all OS have OpenGL, but at the beginning only SGI-IRIX had them. Do you remember Terminator 2?)

The Shell: The command line

Fundamental part in a operative system in charge of execute commands. It often has characteristics such as: process control, input / output redirection, and a **programming language to write scripts**.



```
txino@siegfried: /home/txino/HADDOCK
[16:35:09][txino@siegfried|~]$ cd CONDOR
[16:36:25][txino@siegfried|~/CONDOR]$ ls
condor_error.txt
[16:36:26][txino@siegfried|~/CONDOR]$ cd ../HADDOCK/
[16:36:33][txino@siegfried|~/HADDOCK]$ ls
CONDOR/  e2a-hpr_ori/  HOWTO-HADDOCK.txt@  README.txt
DOCS/    HAD2/        IMPORTANT-LLEGIR.txt@
e2a-hpr/ HADDOCK_EN_10_PASSOS.txt@ JACS-Haddock-Bonvin.pdf
[16:36:34][txino@siegfried|~/HADDOCK]$ mkisofs
mkisofs: Missing pathspec.
Usage: mkisofs [options] file...

Use mkisofs -help
to get a list of valid options.
[16:36:38][txino@siegfried|~/HADDOCK]$ qstat
Job id      Name                User           Time Use S Queue
-----
56501.compbio  shell_elegante      rojillo        00:00:21 R queu1
58327.compbio  MultiAlign BTC7     ilopez         77:21:13 R queu1
61630.compbio  jmd_acd_brbn_01     sereina        20:54:00 R queu1
61631.compbio  jmd_acd_brbn_01     sereina        20:38:53 R queu1
61632.compbio  jmd_acd_mebn_01     sereina        15:25:04 R queu1
61633.compbio  jmd_acd_mebn_01     sereina        12:49:34 R queu1
61634.compbio  jmd_acd_brbn_02     sereina        09:59:23 R queu1
61635.compbio  jmd_acd_clbn_02     sereina        09:34:22 R queu1
61636.compbio  jmd_acd_mebn_02     sereina        08:13:54 R queu1
61637.compbio  jmd_acd_clbn_02     sereina        05:29:36 R queu1
61638.compbio  jmd_acd_clbn_02     sereina        04:18:59 R queu1
[16:36:52][txino@siegfried|~/HADDOCK]$
```

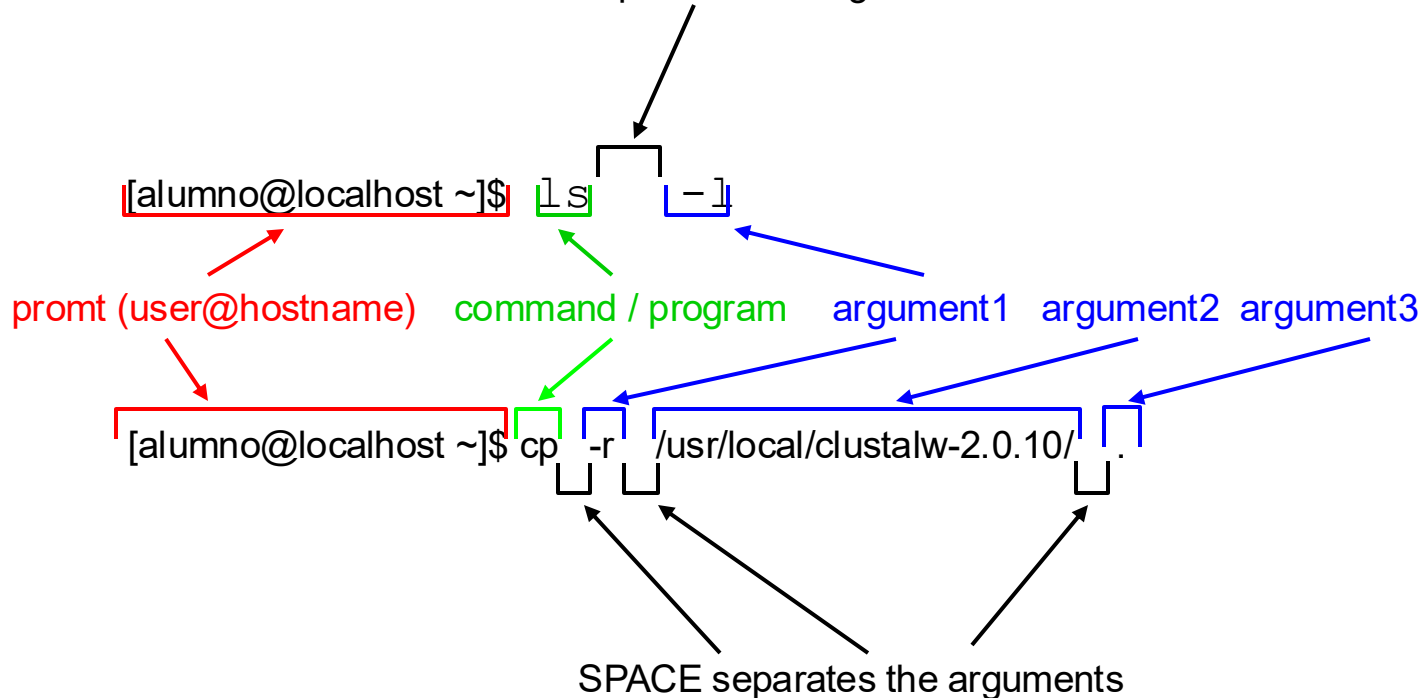
- Fast data access
- Fast command execution (when you know them)
- Multiple command combined execution
- Tasks automation (Scripts)
- Fast and easy remote work and administration with a little bandwidth consume

- bash: GNU Bourne Again Shell, default linux shell, based on the old shell known as "sh".
- tcsh: shell with similar syntaxes to C programming language, based on the old "csh"
- zsh: An extended Bourne shell with many improvements over bash, currently default in macOS.

]\$ command [argument1] [argument2] ... [argN]

Exercise 0: execute a command

SPACE separates the arguments



```
[alumno@localhost ~]$ rm -rf clusterw-2.0.10/
```

```
[alumno@localhost ~]$ ls -l -a
```

```
[alumno@localhost ~]$ ls -la
```

```
[alumno@localhost ~]$ less .bashrc
```

Directory tree and the filesystem I

ls -l /

```
drwxr-xr-x  2 root    root    4096 Dec 14 12:51 bin/
drwxr-xr-x  3 root    root    4096 Jan  7 13:30 boot/
drwxr-xr-x  1 root    root      0 Jan  1  1970 dev/
drwxr-xr-x 32 root    root    4096 Nov 15 16:30 home/
drwxr-xr-x  2 root    root    4096 Sep 29 18:54 initrd/
drwxr-xr-x 42 root    root    4096 Sep 30 13:04 etc/
drwxr-xr-x 10 root    root    4096 Jun 20  2004 lib/
drwx----- 2 root    root    4096 Dec  1 11:24 lost+found/
drwxr-xr-x 10 root    root    4096 Nov 11 15:34 mnt/
drwxr-xr-x  2 root    root    4096 Aug 23  1999 opt/
dr-xr-xr-x 189 root    root      0 Jan  7 14:30 proc/
drwx----- 49 root    root    4096 Feb 23 17:40 root/
drwxr-xr-x  2 root    root    4096 Sep  7 18:32/sbin/
drwxrwxrwt 24 root    root    4096 Feb 26 04:15 tmp/
drwxr-xr-x 15 root    root    4096 Sep  1 16:40 usr/
drwxr-xr-x 18 root    root    4096 Sep 29 20:05 var/
[22:30:08] [txino@siegfried] ~/CURSDOCT TX/kk $
```

pwd

```
(root) /
└─ /home
    └─ /home/$USER
```

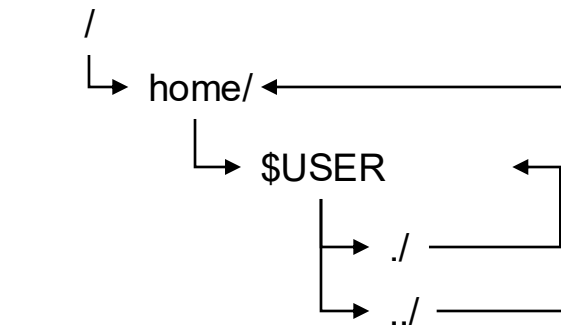
ls -la

ls -la /home/\$USER

```
[22:42:13] [txino@siegfried] ~/alumni $ ls -la
total 12
drwxr-xr-x  2 txino  users  4096 Feb 26 22:42 ./
drwxr-xr-x 109 txino  users 8192 Feb 26 22:42 ../
```

cd .
pwd

cd ..
pwd



cd \$USER
ls -la .

relative path: ls -l ../

absolute path: ls -l /home

Directory tree and the filesystem II

/bin	System executables
/boot	kernel and its modules, needed in order to boot up
/dev	Device acces
/home	User data
/etc	config files
/lib /lib64	System libraries
/mnt /media	Other disks (USBs, DVDs, network units ...)
/root	Administrator files (do not touch)
/sbin	admin executables
/tmp	temporary files
/usr	Not system programs, libraries, documentation, etc, installed by the OS package manager.
/opt y /usr/local	Not system programs, libraries, documentation, etc which are not provided with the OS.
/var	Varied stuff (received emails, web pages, “logs”, etc)

Directory tree and the filesystem III:

Mount points

less /etc/fstab

mount

df

```
/dev/hda5 / ext3 defaults 1 1
/dev/hda7 /Lhome ext3 defaults 1 2
none /dev/pts devpts mode=0620 0 0
dev/scd0 /mnt/cdrom auto user,ro,noauto,ioccharset=iso8859-1 0 0
#none /mnt/cdrom supermount dev=/dev/scd0,fs=udf:iso9660,ro,--,ioccharset=iso8859-1 0 0
none /mnt/cdrom2 supermount dev=/dev/hdd,fs=udf:iso9660,ro,--,ioccharset=iso8859-1 0 0
#none /mnt/floppy supermount dev=/dev/fd0,fs=ext2:vfat,--,ioccharset=iso8859-1,umask=0,sync,codepage=850 0 0
/dev/hda1 /mnt/windows vfat ioccharset=iso8859-1,umask=0,codepage=850 0 0
none /proc proc defaults 0 0
/dev/hda6 swap swap defaults 0 0

compbio:/home /home nfs defaults 0 0
compbio:/bin /bin nfs defaults 0 0
compbio:/sbin /sbin nfs defaults 0 0
compbio:/usr /usr nfs defaults 0 0
compbio:/lib /lib nfs defaults 0 0
compbio:/root /root nfs defaults 0 0
compbio:/var/lib /var/lib nfs defaults 0 0

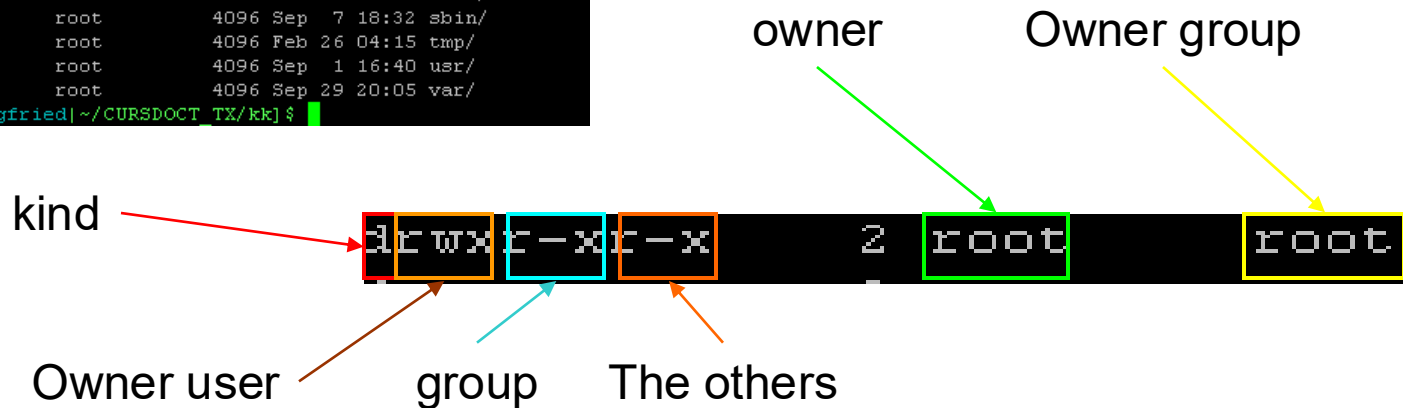
compbio:/home /mnt/netcompbio/home nfs defaults 0 0
binf:/home /mnt/netbioinf/home nfs defaults 0 0
binf:/raid /mnt/netbioinf/raid nfs defaults 0 0
#compbio:/mnt/extra /mnt/netcompbio/extra nfs defaults 0 0
compbio:/extra /extra nfs defaults 0 0
clus15:/Lhome /mnt/clus15 nfs defaults 0 0
clus16:/Lhome /mnt/clus16/Lhome nfs defaults 0 0
clus16:/Lhome /mnt/clus16/backup nfs defaults 0 0
~
~/etc/fstab lines 1-27/27 (END)
```

There is only one root, different devices are mounted in directories inside a unique tree

Directory tree and the filesystem IV : Permissions

ls -l /

```
drwxr-xr-x  2 root  root    4096 Dec 14 12:51 bin/
drwxr-xr-x  3 root  root    4096 Jan  7 13:30 boot/
drwxr-xr-x  1 root  root      0 Jan  1 1970 dev/
drwxr-xr-x 32 root  root    4096 Nov 15 16:30 home/
drwxr-xr-x  2 root  root    4096 Sep 29 18:54 initrd/
drwxr-xr-x 42 root  root    4096 Sep 30 13:04 etc/
drwxr-xr-x 10 root  root    4096 Jun 20 2004 lib/
drwx----- 2 root  root    4096 Dec  1 11:24 lost+found/
drwxr-xr-x 10 root  root    4096 Nov 11 15:34 mnt/
drwxr-xr-x  2 root  root    4096 Aug 23 1999 opt/
dr-xr-xr-x 189 root  root      0 Jan  7 14:30 proc/
drwx----- 49 root  root    4096 Feb 23 17:40 root/
drwxr-xr-x  2 root  root    4096 Sep  7 18:32 shin/
drwxrwxrwt 24 root  root    4096 Feb 26 04:15 tmp/
drwxr-xr-x 15 root  root    4096 Sep  1 16:40 usr/
drwxr-xr-x 18 root  root    4096 Sep 29 20:05 var/
[22:30:08] [txino@siegfried] ~/CURSDOCT TX/kk $
```



Only owner and administrator can change permissions from a file or directory

Environment Variables

Some hidden values that OS has, so it can access certain data in a fast way.

- Command “**set**” o “**printenv**”: (do not scare) they show us which environment variables are defined in a system.

echo \$PATH Where are the executable programs

echo \$HOME Our main working directory

echo \$LD_LIBRARY_PATH Where the user defined function libraries are

- .bashrc / .cshrc / .tcshrc: where users define their own environment variables and other stuff that shells need. **IMPORTANT**: this file is only loaded when you start a shell, so for any changes to take effect you need to start a new one (execute bash or zsh, exit and open a new terminal, etc)

less .bashrc

- .bash_history: where all commands executed by the users in a bash shell are stored

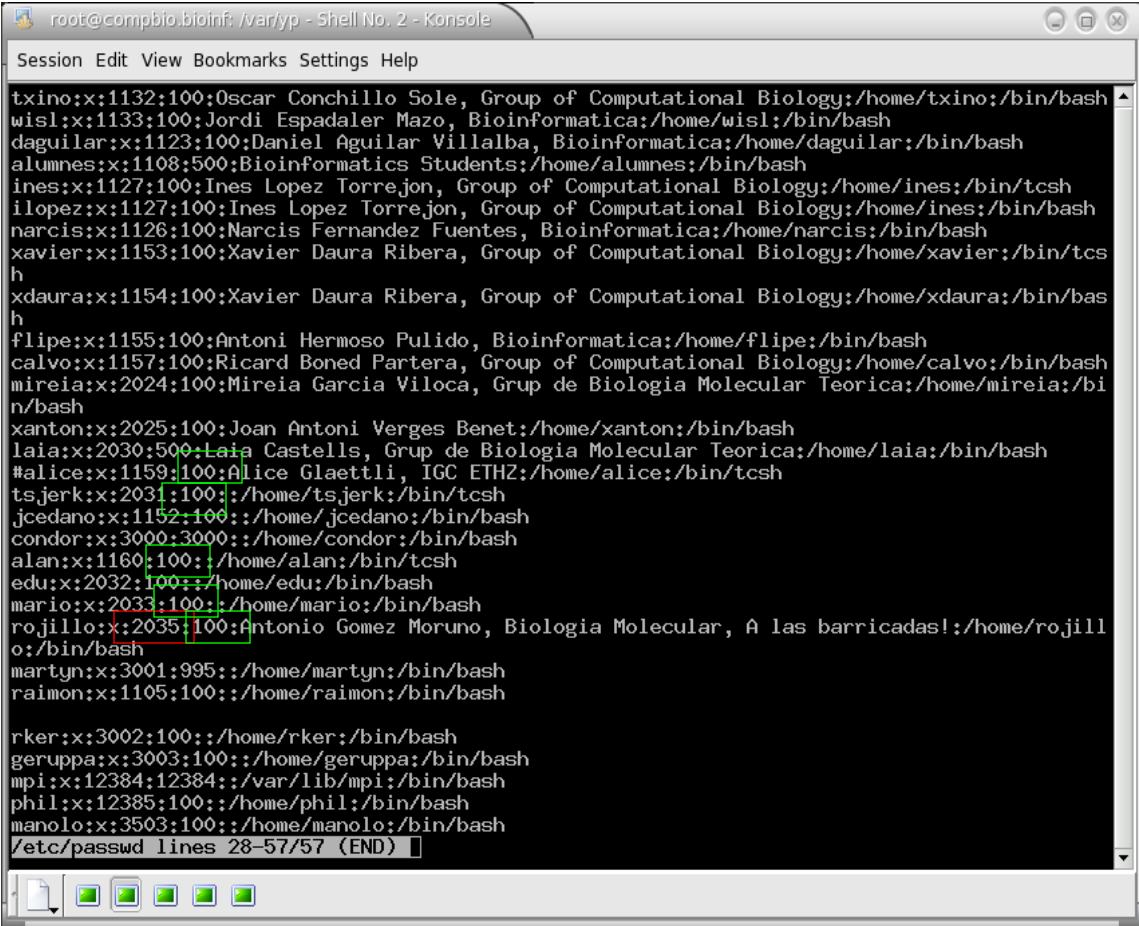
less .bash_history

- **source** : exports environment variables defined in a text file to the present shell.

Users

`less /etc/passwd`

- Every user has a unique identification number (UID)
- If two users have the same UID they are the same user
- Every user belongs to one or more groups
- Users in a same group have more privileges between them than with others



```
root@compbio.bioinf: /var/yp - Shell No. 2 - Konsole
Session Edit View Bookmarks Settings Help

txino:x:1132:100:Oscar Conchillo Sole, Group of Computational Biology:/home/txino:/bin/bash
wisl:x:1133:100:Jordi Espadaler Mazo, Bioinformatica:/home/wisl:/bin/bash
daguilar:x:1123:100:Daniel Aguilar Villalba, Bioinformatica:/home/daguilar:/bin/bash
alumnos:x:1108:500:Bioinformatics Students:/home/alumnos:/bin/bash
ines:x:1127:100:Ines Lopez Torrejon, Group of Computational Biology:/home/ines:/bin/tcsh
ilopez:x:1127:100:Ines Lopez Torrejon, Group of Computational Biology:/home/ines:/bin/bash
narcis:x:1126:100:Narcis Fernandez Fuentes, Bioinformatica:/home/narcis:/bin/bash
xavier:x:1153:100:Xavier Daura Ribera, Group of Computational Biology:/home/xavier:/bin/tcsh
h
xdaura:x:1154:100:Xavier Daura Ribera, Group of Computational Biology:/home/xdaura:/bin/bas
h
flipe:x:1155:100:Antoni Hermoso Pulido, Bioinformatica:/home/flipe:/bin/bash
calvo:x:1157:100:Ricard Boned Partera, Group of Computational Biology:/home/calvo:/bin/bash
mireia:x:2024:100:Mireia Garcia Viloca, Grup de Biologia Molecular Teorica:/home/mireia:/bi
n/bash
xanton:x:2025:100:Joan Antoni Verges Benet:/home/xanton:/bin/bash
laia:x:2030:500:Laia Castells, Grup de Biologia Molecular Teorica:/home/laia:/bin/bash
#alice:x:1159:100:Alice Glaettli, IGC ETHZ:/home/alice:/bin/tcsh
tsjerk:x:2031:100::/home/tsjerk:/bin/tcsh
jcedano:x:1152:100::/home/jcedano:/bin/bash
condor:x:3000:3000::/home/condor:/bin/bash
alan:x:1160:100::/home/alan:/bin/tcsh
edu:x:2032:100::/home/edu:/bin/bash
mario:x:2033:100::/home/mario:/bin/bash
rojillo:x:2035:100:Antonio Gomez Moruno, Biologia Molecular, A las barricadas!:/home/rojill
o:/bin/bash
martyn:x:3001:995::/home/martyn:/bin/bash
raimon:x:1105:100::/home/raimon:/bin/bash

rker:x:3002:100::/home/rker:/bin/bash
geruppa:x:3003:100::/home/geruppa:/bin/bash
mpi:x:12384:12384::/var/lib/mpi:/bin/bash
phil:x:12385:100::/home/phil:/bin/bash
manolo:x:3503:100::/home/manolo:/bin/bash
/etc/passwd lines 28-57/57 (END)
```

NEVER WORK WITH THE USER

root!!!

Basic Commands

- **man /info**: information about commands
- **cd**: change directory
- **ls**: lists files or directories
- **ls [directory/file name]** metacharacters (*,?) can be used
 - l long
 - a all
 - R recursive
 - S sort by size
 - t sort by date
 - r revert order
 - h human readable sizes
- **cp**: copy files or directories
cp [origin] [destination]
 - i interactive
 - R/r recursive
 - d keeps lincs
 - p keeps permissions
- **rm**: erase files
 - i interactive
 - r recursive (and directories)
 - f force
- **mkdir/ rmdir**: create/erase directories
- **mv**: move files and directories
- **mv [origin] [destination]**

Basic Commands II

- **chmod**: change permissions

chmod [options] [new permission] [file/directory]

-R recursive

chmod |u|+|r| [file/directory]
|g|-|w|
|o| |x|
|a|

- **pwd**: tells the current working directory
- **ln -s**: create a symbolic link

ln -s [origin] [destination] (take care with the absolute / relative path)

ln -s /usr/local/clustalw2 (if destination is "." it can be ignored for a single file)
less clustalw_help

- **export** / **setenv**: creates or changes environment variables and turns it exportable to new shells generated by the current one.

bash: **export CST="/usr/local/clustalw2"**

csh: **setenv CST "/usr/local/clustalw2"**

- **echo**: writes a text or variable value

echo "/usr/local/clustalw2" ; echo \$CST

Exercise 1: working with directories

Execute this commands in a terminal

```
#cd without arguments is equivalent to cd $HOME
cd
#wget is a command to download files from web or ftp servers
wget http://bioinf.uab.es/ocs/INTRO_LINUX/linux_CLI_exercises.tgz
#tar zxvf decompresses a gzipped tar file (usually .tgz or .tar.gz)
tar zxvf linux_CLI_exercises.tgz
#less is a text file reader
less ~/LINUX_CLI_EXERCISES/exercise_1.txt
```

Now execute the commands that you find in the opened file.
I strongly suggest that you read the whole file (comments included) before starting to execute anything

- You should be able to move through the directory tree
- And understand the difference of a relative path and an absolute one

process commands I

We call **process** to any program currently on execution

- **top**: shows in which tasks the processor is working on in real time
 - p [process ID] only this process

```
top - 16:41:20 up 51 days, 3:12, 2 users, load average: 0.03, 0.06, 0.07
Tasks: 173 total, 1 running, 172 sleeping, 0 stopped, 0 zombie
Cpu(s):  0.0% user,  4.6% system,  0.0% nice,  95.4% idle
Mem:   1032408k total, 1018764k used, 13644k free, 18980k buffers
Swap:  4313412k total,  231484k used, 4081928k free, 328576k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
5184	txino	16	0	1132	1132	868	R	2.6	0.1	0:00.62	top
1455	condor	9	0	1412	1376	1132	S	0.3	0.1	9:05.55	condor_master
32323	txino	10	0	6740	4392	3160	S	0.3	0.4	16:42.15	kdeinit
8081	txino	10	0	226m	226m	2188	S	0.3	22.4	611:15.82	mlnet
1	root	9	0	112	76	64	S	0.0	0.0	0:36.84	init

options: **M** sorts by memory usage
N by name
P by CPU usage
1 shows different processors usage

- **ps**: shows all process IN THE CURRENT SHELL

-e all process
-f more information about those process
-u [user] process belonging to a user

```
txino 5117 5116 4 16:33 pts/40 00:00:00 -bash
txino 5165 5117 0 16:33 pts/40 00:00:00 ps -ef
[16:33:27] [txino@siegfried|~]$
```

owner

process ID

process parent ID

process commands II

- **kill**: sends a signal to a process (by default TERM)
- **kill [options] [PID]**
 - 9 KILL signal
- **&** Modifier: executes a process in “background”
[executable] [options and arguments] &
- **Ctrl c**: kills a program
- **Ctrl z**: pauses (stop) a program
 - bg**: sends a paused program to “background”
 - fg**: returns a program from “background” to “foreground”
- **nohup**: immunizes a program against close signals (eg: exit)
- **xkill**: kills a graphical program

execution related commands

- **bash / csh / sh**: opens a new shell inside the current one, exportable environment variables are propagated to the son

bash		%text file with commands%
csh		
sh		

This commands are executed by a new shell which closes itself when the last command ends

bash -c "%command1%;%command2%"

creates a new **bash** shell, execute commands and exits

bash -x "%text file with commands%"

creates a new **bash** shell, execute the commands in the script showing each command (sort of a debugger)

- **alias**: defines a new alias or shows the meaning of one of them or shows all available aliases

alias [existing alias]

alias [new alias]=[meaning] (in bash)

alias [new alias] [meaning] (in csh or tcsh)

- **which**: where is an executable that is in PATH or defined by an alias
which [program]

Exercise 2: execute a process, identify it, monitor it and kill it

Follow the instructions in the file "`~/LINUX_CLI_EXERCISES/exercise_2.txt`"

- Watch how `clustalw2_sleep` has not run because the directory was not in the PATH
- Watch how the `bash` process is the father of the `clustalw2_sleep` process
- Check how a running process prevents `shell` functionality

Command Lines: User commands

- **passwd**: changes your password
- **who** / **w** : who is connected to this computer?
- **whoami**: who am I? (better use **id**)
- **su**: changes the user but keeping the current environment
 - l does not keep the current environment, loads the new user one and starts from the new user home directory
- **id**: who are you and to which group do you belong

File commands I

- **file [filename]** : tells what kind of a file it is

- **more / less**: shows the content of a text file

less:

-N shows line numbers

+ [num] starts with this line number

options:	g	beginning of the file
	G	end of the file
	SPACE	go forward one screen
	b	go backward one screen
	/	search forward
	?	Search backward
	n	repeats search
	N	repeats search in reverse sense

- **cat [file1] ... [file n]** : sends the file(s) contents to the standard output (the current terminal by default)

File commands II

- **grep**: shows lines in files which content a “pattern”
- **grep [options] [pattern] [file(s) name(s)]**
 - l shows only file name for those that match
 - v shows lines that do not match
- **rgrep**: recursive grep
 - r really recursive!
- **find**: search for files not only by name
find [directory] -name [name] -print
find [dir] -name [name] -exec [command, use {} as a substitute for file name] \;
- **wc**: counts how many bytes, lines and words are in a file
 - l only counts lines

File commands III

Vi (thanks to José A. Parra Merino)

vi / vim: the TEXT EDITOR, it has lots of options and allows us to automate lots of tasks (if you know how to use it, which is not easy).

When we are executing **vi** we can be working in 3 different modes:

command mode : default one, here keys are not used to write anything , they have different functions.

writing mode: here they write

ex mode: to save, exit, configure, etc.

File commands III

vi: command mode

Ways to move in the document

w	Moves forward a word.
W	Moves forward a word, but only separated by spaces.
b	Moves backwards a word.
B	Moves backwards a word, but only separated by spaces.
\$	Goes to the end of current line.
^	Goes to the beginning of current line.
nG	Goes to the n line.
G	Goes to the last line.

Editing commands

d	Deletes: dd deletes a whole line, ndd deletes n lines, d\$ and D deletes until the end of the line, 3dw deletes 3 words. When a text is deleted it is also copied to the buffer so later can be pasted with p .
x	Deletes just a character.
y	Copies selected text to the buffer (so it can be pasted).
p	Pastes the text from the buffer.
ZZ	Saves the document and exits the editor.

File commands III

vi: command mode

To go to the typing mode there are different commands, the ESC key exits this mode returning to the command one. These are some writing and inserting commands:

- i** Goes to typing mode in the same position where the cursor is.
- I** Goes to typing mode at the beginning of the current line.
- a** Goes to typing mode in the next character.
- A** Goes to typing mode at the end of current line.
- o** Creates a new line below the current one and inserts text on it.
- O** Creates a new line over the current one and inserts text on .

File commands III

vi: Ex mode

To access this mode we press `:` followed by any **Ex** command. **Ex** commands process the text line by line doing their corresponding changes. There are some **Ex** commands that do not affect the text, they are used to alter vi behavior and configuration or to perform specific tasks:

`:%s/pattern1/pattern2/g`:substitutes pattern1 by pattern2 in the whole document.

`:r file`

Inserts the file contents in the document.

`:r !command`

Inserts the stdout of this command in the document.

`:syntax on / off`

Activates or deactivates syntax highlighting.

`:w`

Saves the document.

`:wq`

Saves the document and exits the editor.

`:q!`

Exits without saving.

`:set nu / nonu`

Activates or deactivates line numbers.

File commands IV

user friendly editors

emacs: another editor, but more “user friendly”

Graphical mode Editors: **nedit**, **gedit**, **xemacs**, **kwrite**, **kate**, **gvim**, ...

File commands V

Compress/Uncompress

- **bzip2 / bunzip2**: compress / uncompress a “bzip2” compressed file (.bz2)
- **gunzip / gzip**: (un)compress a “.gz” or “.Z” file

- **tar**: join some files in only one

V verbose

C create a new file

X extract

t lists the contents

Z working with gz compressed tar files

j working with bz2 compressed tar files

f %filepath" use this file to extract or compress

tar zcvf %directory.tgz% %directory% adds all files from “directory” to a “gz” compressed tar file called directory.tgz

tar zxvf %filepath.tgz% extracts all files from **filepath.tgz** to the current directory

- **zip /unzip** : compress / uncompress a zip file

Exercise 3: vi or nano or whatever you like

Modify the `$HOME/.bashrc` file, add the WORK directory to the PATH and create some alias for `mv`, `cp`, `rm` and `ls -ltr` (take care with COPY PASTE!)

```
vim ~/.bashrc
#add next lines to the recently open file

export PATH=$HOME/PRACT_SO/WORK/:$PATH

alias mv='mv -i'
alias cp='cp -i'
alias rm='rm -i'
alias t='ls -ltr'
```

This Environment variables are still unknown for the shell, we need to load them with the command `source`.

```
source ~/.bashrc
```

BUT I strongly advise against it, it is better that you start from scratch.

execute:

```
exit
```

or

```
bash
```

Check that now `clustalw2_sleep` can be executed from any directory.

Check that now `t` is a command that produces the same result as `ls -ltr`

Standards (input/output)

- **stdin**: standard input, keyboard usually.
- **stdout** standard output, the screen usually.
- **stderr** standard error, the screen usually.

Standards (input/output)

Redirection: pipes I

| : redirects **stdout** from a program to the **stdin** from another

> : redirects **stdout** from a program to a new file, if that one exists it is destroyed.

>> : redirects **stdout** from a program to a file, adding that to the end of it. If that one does not exist it is created.

< : redirects the contents of a file to the **stdin** of another.

2>&1 : redirects **stderr** to the **stdout**.

xargs: redirects **stdin** to a command arguments
 \ls |xargs -ti du -s {}

Exercise 4: Pipes & Standards

Follow the instructions in the file "~/ LINUX_CLI_EXERCISES /exercise_4.txt"

- If we want to execute a program that require interactive input in a non interactive way (for example from another program) this would be the way.
- How many steps do you think we need to compare 1 by 1 all sequences against the Ecoli one? (here I have 5 files, but they could be thousands!)

```
\ls *fst|grep -v Eco|xargs -ti bash -c "cat Eco.fst {}  
>clusin.inp;~/LINUX_CLI_EXERCISES/clustalw2 -ALIGN -OUTFILE={}+ecoli.aln -INFILE=clusin.inp"
```

Network commands

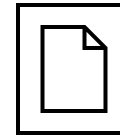
- **ping** : check if a computer is connected (if its firewall allows it)
ping [ip / computer's name]
- **/sbin/ifconfig**: shows all network devices and their IP address.
- **ip a s** (the same, more modern approach, the standard today).
- **nmcli** (the same, more user friendly).
- **hostname**: shows our computer's name.
 - f full name
 - A all names
- **ssh**: opens a shell from a remote computer in our computer. The connection with it is encrypted.

ssh [user]@[ip/computer's name]
- **wget**: ftp and http client file downloader.
wget "http://ftp.ebi.ac.uk/pub/software/clustalw2/2.0.10/clustalw-2.0.10-src.tar.gz"
 - i [file] download all address in this file
- **lynx**: Text web browser (very used at the www beginnings)

Exercise 5: the Script

One of the main advantages of the command line are the scripts, we can type different commands in a text file, give it execution permissions and execute it, the command interpreter will execute all of them one by one:

```
cd
ls PRACT_SO/
tar zcvf PRACT_SO.tgz PRACT_SO/
rm -rf PRACT_SO
less LINUX_CLI_EXERCISES/script_all.sh
# LINUX_CLI_EXERCISES/script_all.sh
```



- That script allows us to execute everything that we've done in class with only one command.

After executing it, navigate through the directories and observe that all the files have been created as they were.

- Now concatenate the **exercise_?.txt** files and modify the resulting file to create your own **script_all.sh**.

INFO: you can view the execution line by line using:

bash -x %your_script%

WARNING: text files in windows are not equal in UNIX. Investigate about the following command: **dos2unix**

Compiling I

Computer's language is very complicated for a human being (01000001 = A) so in order to create a new program we use programming languages (Fortran, C, java, python, etc).

To “translate” this programs to a language that a computer understands we have two methods:

Interpret: “translation” is done line by line. The computer must have the interpreter installed to execute your program. In this case, the program that it is executed is the interpreter, your program (in this case AKA: script) is a data file which is read by the interpreter which acts according to it. In a similar way as Microsoft Word reads a "*.docx" file and acts according to it (eg: python).

Compile: we translate the whole program at the same time, once compiled it is not necessary that the executing computer has the interpreter, but if the program uses shared libraries, they are going to be needed (eg: C or Fortran).

Almost all languages have a compiler (or interpreter) for each O.S., so the same source code can be interpreted or compiled in different O.S. and/or architectures without any changes (if we wrote it right).

Compiling II

When we download a source code (what a programmer has written), generally, the compilation and installation process is done in three steps:

(But first we will **ALWAYS** read the README or INSTALL or any other documentation file)

cmake %several options here%

or

./configure

Both ways check that our computer has the compiler and all necessary libraries. Some user depending options will be specified at this moment. This process generates the “**Makefile**” file that contains all the compiler options and the location of all needed libraries.

- help shows all possible options

- prefix=[install directory], **/usr/local** by default.

make: the true compilation process, all compiler options are read from the “**Makefile**”.

- j %number of simultaneous jobs to run%

make install: compiled program is installed in the directory specified when “**configure**” was executed.

Exercise 6: compile and install an application

- Download clustalw2 source code (maybe it has already been done)

```
cd
cd PRACT_SO/
mkdir PROGRAMS
cd PROGRAMS/
wget "http://ftp.ebi.ac.uk/pub/software/clustalw2/2.0.10/clustalw-2.0.10-src.tar.gz"
tar zxvf clustalw-2.0.10-src.tar.gz
ls clustalw-2.0.10
```

- Take care of permissions when you install something, maybe you don't have writing permissions in the default installation directories.
- **YOU ARE NOT ALLOWED** to use sudo or su to acquire root privileges. You must complete this exercise as a simple user. (You can't, READ again the previous slide!)

!!!GOOD LUCK!!!

tmux is a terminal multiplexer. (read more here https://linuxcommand.org/lc3_adv_termmux.php) **tmux**

It keeps our session alive as we have it even if we close the terminal, the connection to a remote computer is closed and other cases.

it also allows us to generate "tabs" inside the session.

It is very useful for remote sessions. We only need to make one connection and then we can create new "tabs" if we need other terminals. Apart from that, as we mentioned, if the connection is closed we do not lose this session.

It is also very useful when running long jobs, we leave them running inside a tmux and we can close the terminal, even logout (of course, we cannot turn off the computer where the job is running, but if it is running in a remote server, we can turn off ours, reconnect later and recover the session)

We are going to repeat the compilation process but inside a tmux session

```
tmux
cd /home/ocs/PRACT_SO/PROGRAMS/clustalw-2.0.10
./configure --prefix=$HOME/PRACT_SO/PROGRAMS/clustalw2
#while it is running create a new tab with pressing "ctrl b" and later "c" (without ctrl)
#we can go back to the previous tab with "ctrl b" and "p"
make
#while it compiles go to the other tab with "ctrl b" "p" (previous) or "ctrl b" "n" (next)
# you only have two tabs so "next" and "previous" is the same. You can see all tabs and select which one to use
with "ctrl b" "w"
#in the other tab execute:
top
#now close the terminal window
open another terminal and execute
tmux ls # lists the running sessions
tmux att #short for attach-session
# play with "ctrl b" "n" or "ctrl b" "p" to see that everything is as it was.
# press "ctrl b" "d" to detach the session
tmux att to reattach it again
# you can play a bit with tmux. When you finish close each opened tab with :
exit
# make sure no session remains
tmux ls
# if anyone remains reattach it and exit from all of them
```

<https://linuxcommand.org>

We strongly advise that you take your time to go this site:

<https://linuxcommand.org>

and follow the the following tutorials:

https://linuxcommand.org/lc3_learning_the_shell.php

https://linuxcommand.org/lc3_writing_shell_scripts.php

if you are still in the mood go to [Adventures](#):

https://linuxcommand.org/lc3_adv_termmux.php

https://linuxcommand.org/lc3_adv_mc.php