

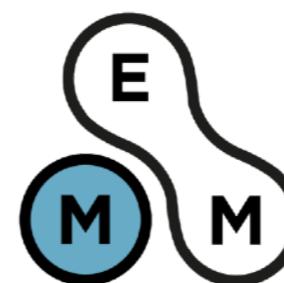
INTRODUCTION TO BIOINFORMATICS IN HIGH-PERFORMANCE COMPUTING ENVIRONMENTS

Pablo Sánchez

Who am I?



Institut
de Ciències
del Mar

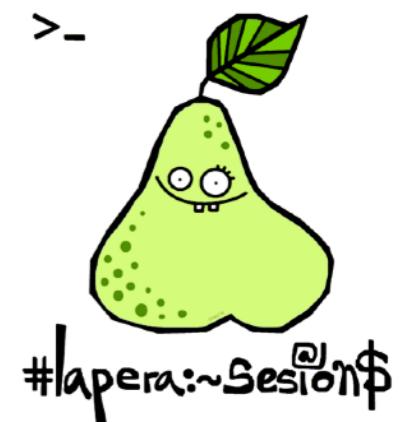


Ecology
of Marine
Microbes



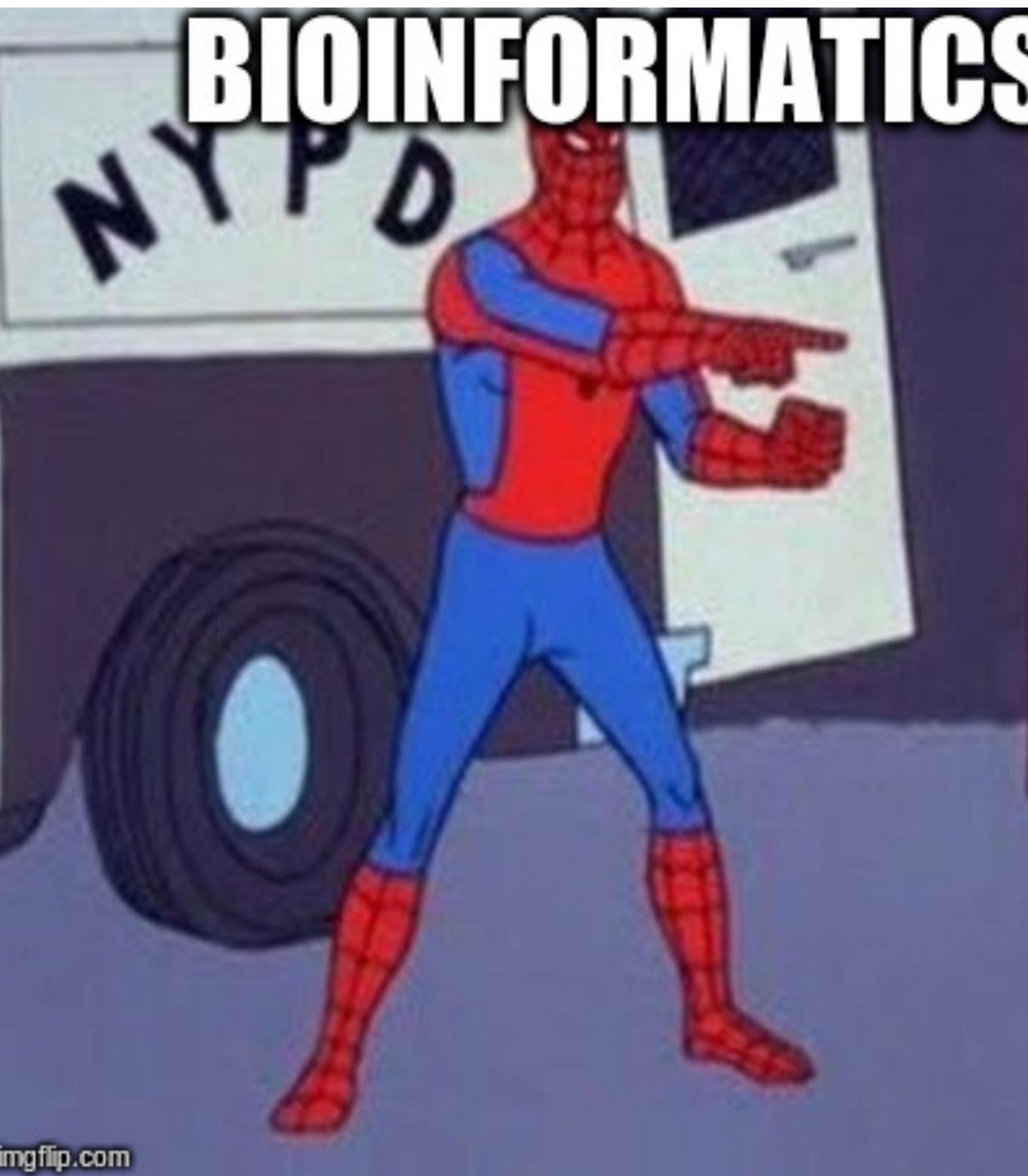
Pablo Sánchez

Ecology of Marine Microbes group, Bioinformatics Core Service
Accidental marbits system administrator

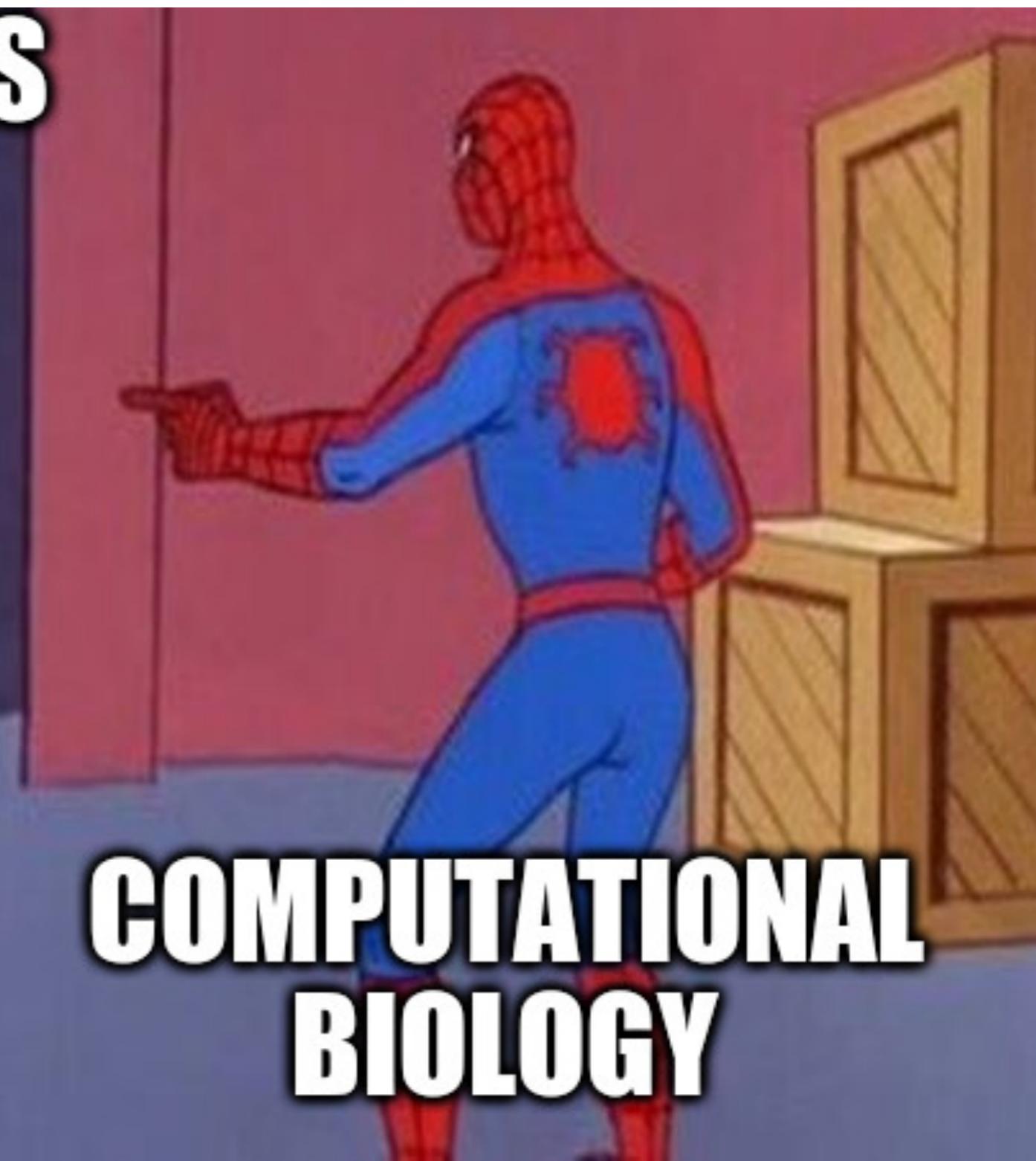


What is Bioinformatics?

BIOINFORMATICS



COMPUTATIONAL BIOLOGY



Why Bioinformatics?

Why Bioinformatics?

We have more biological data than we can process. Biology has become a (Big) Data Science

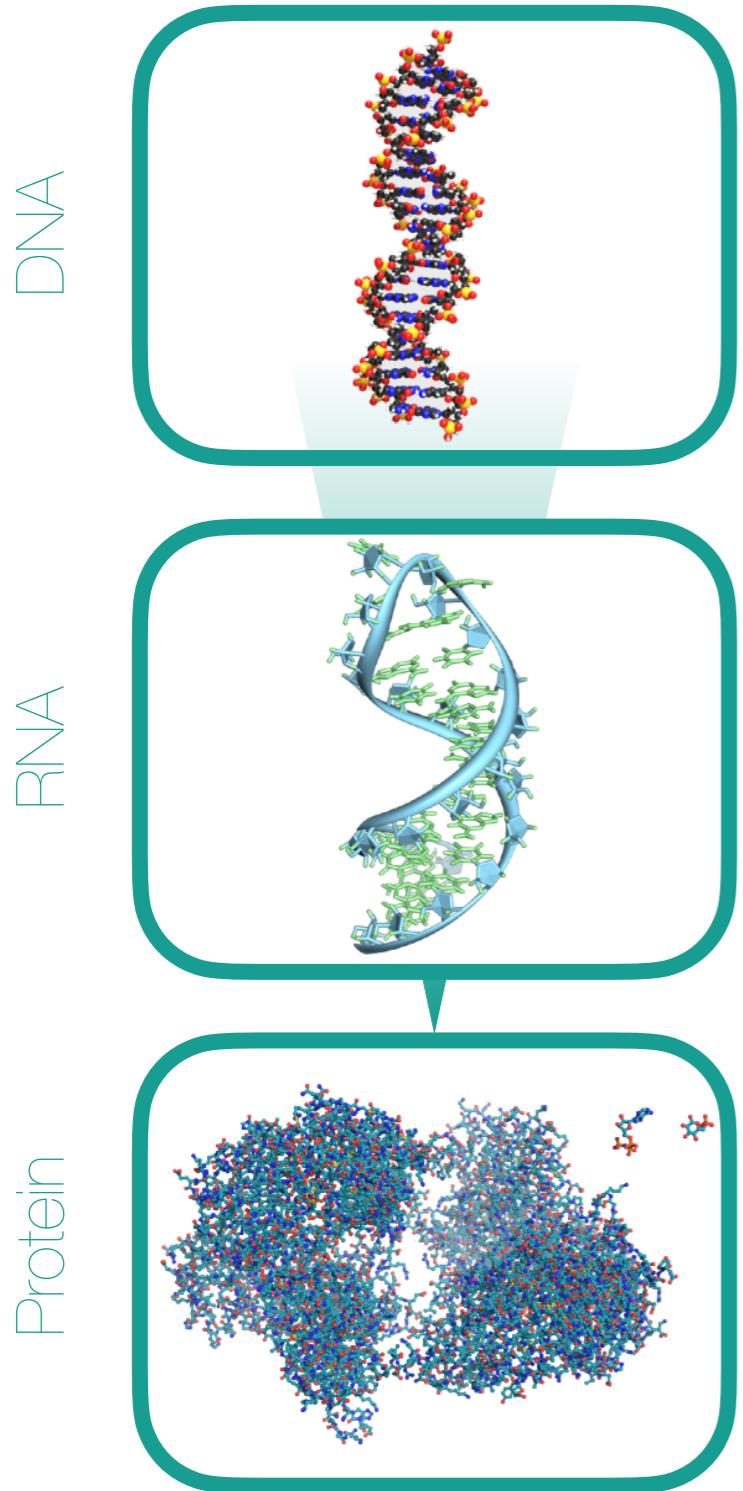
DNA sequences as the prototypical bioinformatics use case

We'll focus in DNA sequences as they are the great bulk of biological data.

But there are many other things: sequence and structure of proteins, studies on the simultaneous expression of many genes under different conditions, mass spectrometry experiments, microscope and 3D image processing, cytometry, network analysis...

Biological information

is coded in genes and expresses itself from/through genes.



A A G T C G G A T

Transcription

A U G U C A G A U

Traduction

M

Methionine

S

Serine

D

Aspartic
acid

Central dogma of biology

Phi X174 bacteriophage. First sequenced genome ever. 5.400 bp

```

CCGTCAGGATTGACACCCCTCCCAATTGTATGTTCATGCCCTCCAAATCTTGGAGGCTTT
3927 3937 3947 3957 3967 3977
                                         ↑
                                         mRNA start

A MET VAL ARG SER TYR TYR PRO SER GLU CYS HIS ALA ASP TYR PHE ASP PHE GLU ARG
TTT ATGGTTCTCGTTCTTATTACCCCTCTGAATGTCACGCTGATTATTTGACCTTGAGCGT
3987 3997 4007 4017 4027 4037
                                         ↓
                                         mRNA end

A ILE GLU ALA LEU LYS PRO ALA ILE GLU ALA CYS GLY ILE SER THR LEU SER GLN SER PRO
ATCGAGGCTCTTAACCTGCTATTGAGGCTTTGTCGATTTCTACTCTTCTCAATCCCCA
4047 4057 4067 4077 4087 4097
                                         ↓
                                         T1/6

A MET LEU GLY PHE HIS LYS GLN MET ASP ASN ARG ILE LYS LEU LEU GLU GLU ILE LEU SER
ATGCTTGGCTTCCATARGCAGATEGATAAACCGCAGTCAGTCTTGGAGAGATTCTGTC
4107 4117 4127 4137 4147 4157
                                         ↓
                                         A7b/7a   M5/6   F5c/3

A PHE ARG MET GLN GLY VAL GLU PHE ASP ASN GLY ASP MET TYR VAL ASP GLY HIS LYS ALA
TTTCGATAGCAGGCGTTGAGTTCTGATATAATGCTGATATGTTGACGGCATAAAGGCT
4167 4177 4187 4197 4207 4217
                                         ↓
                                         T6/2   Q2/3a   Q2/3a   R4/3   22/6b

A ALA SER ASP VAL ARG ASP GLU PHE VAL SER VAL THR GLU LYS LEU MET ASP GLU LEU ALA
GCTTCTGACGTTCTGATGAGTTGATCTGTTACTGAGAAATTGATGGATTTGGCA
4227 4237 4247 4257 4267 4277
                                         ↓

A GLN CYS TYR ASN VAL ILE PRO GLN ILE ASP ILE ASN ASN THR ILE ASP HIS ARG PRO GLU
CAATGCTACAATGTCGCTCCCCCAACTTCAATTAAACACTATAAGACCAACGGCGCGA
4287 4297 4307 4317 4327 4337
                                         ↓
                                         Origin of viral strand replication

A GLY ASP GLU LYS TRP PHE LEU GLU ASN GLU LYS THR VAL THR GLN PHE CYS ARG LYS LEU
GGGGACGAAATACTTCTTACGAAACGAGAAACGCGTTACGCAATTGCGCGCAAGCTG
4347 4357 4367 4377 4387 4397
                                         ↓
                                         M8/6   A7a/4

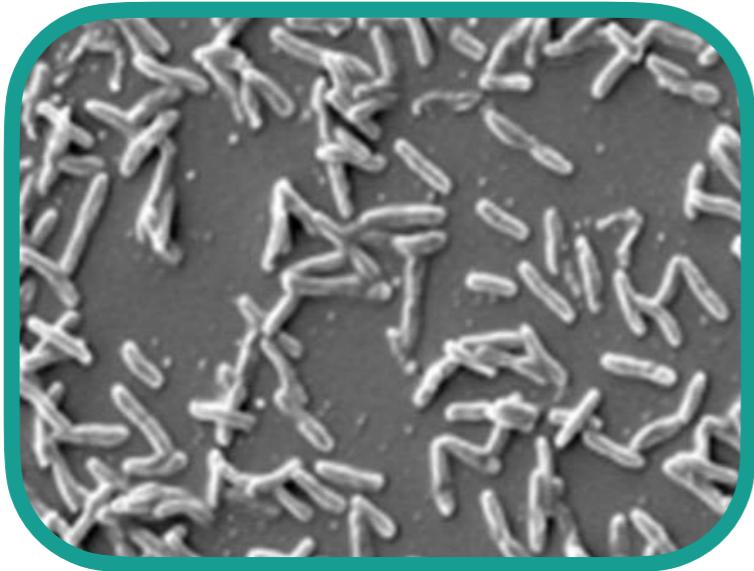
A ALA ALA GLU ARG PRO LEU LYS ASP ILE ARG ASP GLU TYR ASN TYR PRO LYS LYS GLY
GCTGCTGAAACGCCCTCTTAAGGATATTGCGATGAGTATAATTACCCCARAAAGAAAGT
4407 4417 4427 4437 4447 4457
                                         ↓

A ILE LYS ASP GLU CYS SER ARG LEU LEU GLU ALA SER THR MET LYS SER ARG ARG GLY PHE
ATTAAGGATGAGTGTTCAGATTTGCTGAGCGCTCAGACTATAAGTAAATCGCGTAGAGGCTT
4467 4477 4487 4497 4507 4517
                                         ↓
                                         Z6b/ba

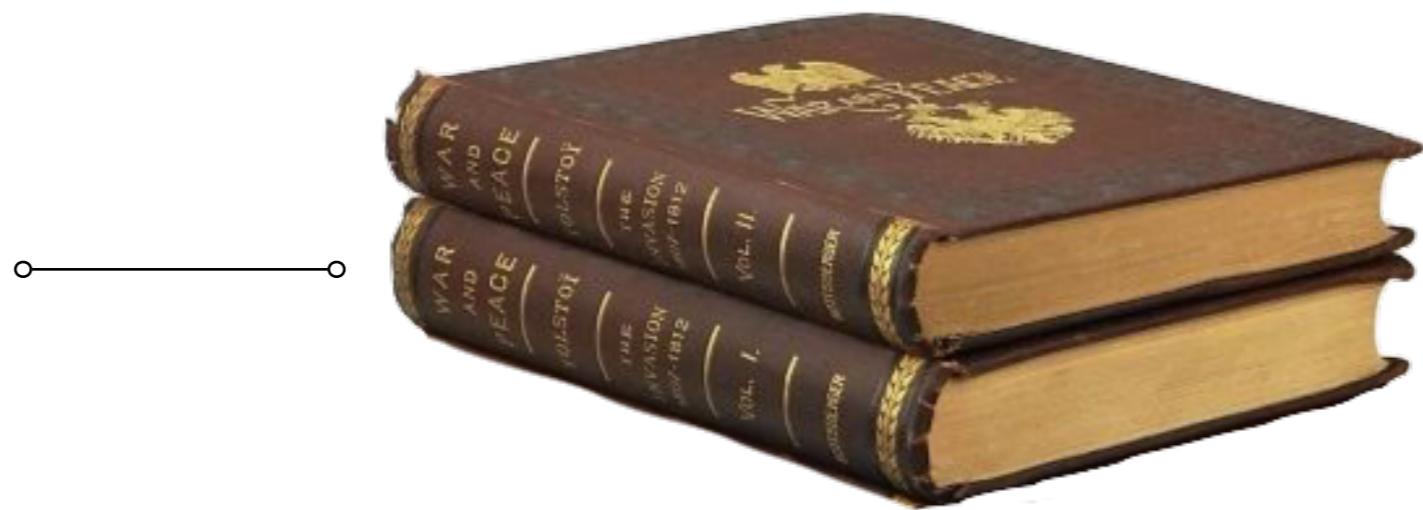
```

FIG. 4

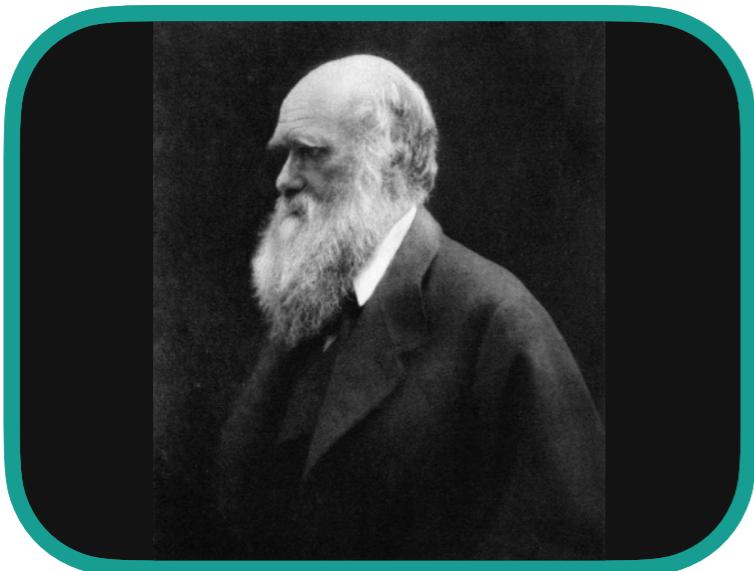
Sanger et al., 1978



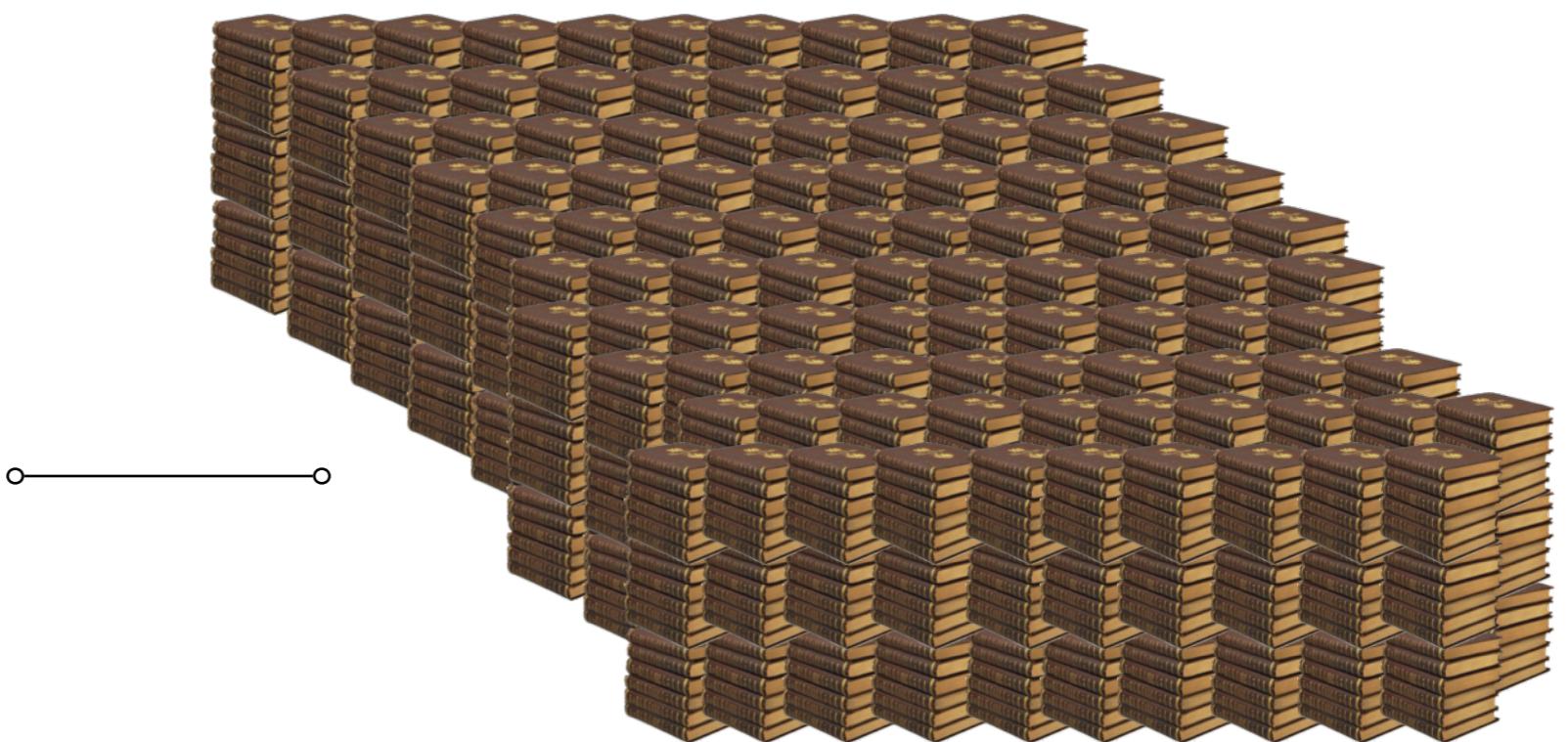
Legionella pneumophila



3.272.023 "characters"

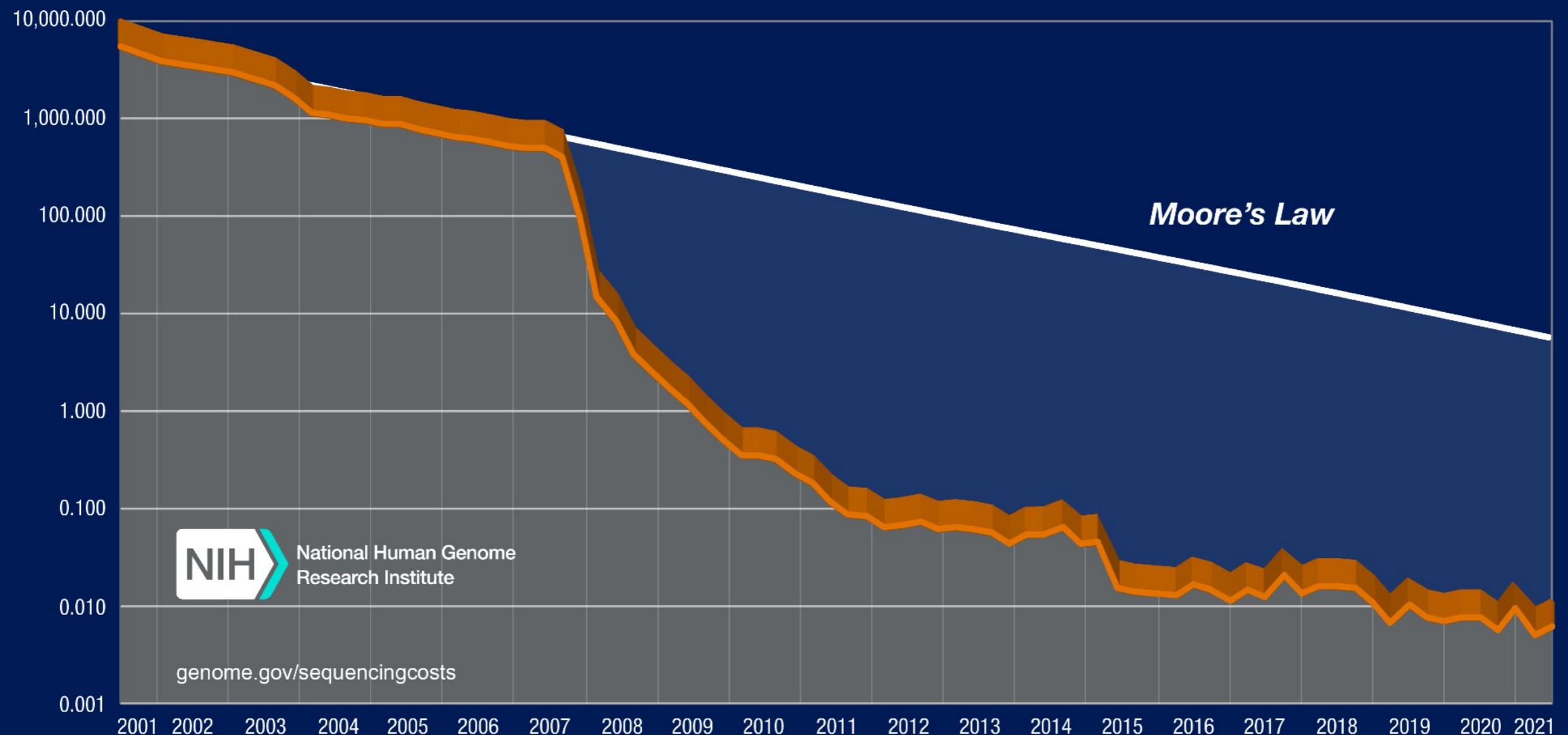


Homo sapiens



3.100.000.000 "characters"

Cost per Raw Megabase of DNA Sequence



Besides the large amount of sequencing data, it's also difficult to extract meaning from it (even with a large computer).

BIOLOGY IS LARGELY SOLVED.
DNA IS THE SOURCE CODE
FOR OUR BODIES. NOW THAT
GENE SEQUENCING IS EASY,
WE JUST HAVE TO READ IT.

IT'S NOT JUST "SOURCE
CODE". THERE'S A TON
OF FEEDBACK AND
EXTERNAL PROCESSING.



BUT EVEN IF IT WERE, DNA IS THE
RESULT OF THE MOST AGGRESSIVE
OPTIMIZATION PROCESS IN THE
UNIVERSE, RUNNING IN PARALLEL
AT EVERY LEVEL, IN EVERY LIVING
THING, FOR FOUR BILLION YEARS.

IT'S STILL JUST CODE.



OK, TRY OPENING GOOGLE.COM
AND CLICKING "VIEW SOURCE."

OK, I-... OH MY GOD.

THAT'S JUST A FEW YEARS OF
OPTIMIZATION BY GOOGLE DEV'S.
DNA IS THOUSANDS OF TIMES
LONGER AND WAY, WAY WORSE.

WOW, BIOLOGY
IS IMPOSSIBLE.



**Este verano...
¡Aplastamos los precios!**

999 €

Procesador Intel® Centrino™ Duo T 2050

Disco duro 100 Gb

Memoria 2048 Mb DDR2 RAM

Tarjeta gráfica Nvidia GeForce 7300 de 256 Mb Turbo Caché

DVI

ASUS

Z92JC

ORDENADOR PORTÁTIL

Procesador Intel® Centrino™ Duo T 2050, 2 x 1.60 GHz, memoria caché 2 Mb, FSB 533 MHz, memoria 2048 Mb DDR2 RAM, disco

duro 100 Gb, pantalla plana táctil 15.4" Ultrabrilante, grabadora de DVD-doble capa, tarjeta gráfica Nvidia GeForce 7300 de

256 Mb Turbo Caché, webcam, modem, red, WiFi, 4.0 USB 2.0, DVI, VGA, Windows XP Home, Antivirus.

2048 Mb DDR2 RAM



Ultrabrilante
15,4"

2048 Mb
DDR2 RAM

YO NO SOY
TONTO

Horario de Lunes a Sábado
10:00 - 22:00

MEDIA MARKT ALICANTE
Parque Comercial Vistahermosa
Avda. Antonio Ramos Carrascal s/n,
03015 Alicante

www.mediamarkt.es

Media Markt

LCD · PLASMA · TV · DVD · VIDEOJUEGOS · HIFI · ELECTRODOMÉSTICOS · INFORMÁTICA · FOTO · CD · MULTIMEDIA MÓVIL

Why HPC?

There's no way of analysing todays' sequencing projects in your laptop...

...and if there's one, you'll also need to learn some skills that will allow you to deal with data, files, documentation, automation of tasks, ensure reproducibility of analyses...

...many of them in a set up without a mouse

(sorry)

Microsoft Excel does not work for bioinformatics

DRG_IDNAME	GENE_SYMBOL	PLATE_CODE	ROW_CODE
YBR124W	YBR124W	YB-01	n
YBR			
YBR094W	YBR094W	YB-01	l
YBR091C	MRS5	YB-01	l
YBR078W	ECM33	YB-01	h
YBR075W	YBR075W	YB-01	h
YBR072W	HSP26	YB-01	h
YBR069C	VAP1	YB-01	h
YBR054W	YR02	YB-01	d
YBR051W	YBR051W	YB-01	d
YBR048W	RPS11B	YB-01	d

Scientists rename human genes to stop Microsoft Excel from misreading them as dates

Sometimes it's easier to rewrite genetics than update Excel

By James Vincent | Aug 6, 2020, 8:44am EDT



Genes named like MARCH1 and SEPT1 which Excel converts into dates when imported the wrong way, usually directly via a CSV text file. Office Watch first mentioned this [gene naming trouble in 2016](#).

Imported by Excel	Should be	
Mar 01	MARCH1	could be converted to '1 March' depending on Windows regional setting.
Sep 02	SEPT2	could be converted to '2 September' depending on Windows regional setting.
2.31E+19	2310009E13	Text converted to large number

Genes have short names because the full name is more than a mouthful. MARCH1 is short for “*Membrane-Associated Ring Finger (C3HC4) 1, E3 Ubiquitin Protein Ligase*”

(Windows neither). Real job offer:

REQUIREMENTS

The candidate should have a Computer Science/Informatics/Bioinformatics degree/equivalent with/or a demonstrated track record in:

- Configuration and administration of Linux computer clusters
- Knowledge of scripting languages (bash, python, perl, ruby)
- Experience in automating OS installation
- Experience with configuration management tools (puppet, ansible)
- Strong diagnostic and analytical skills
- Basic understanding of IT security principles and best practices
- Experience in virtualization technology
- Experience in monitoring and firewall systems
- Installation and maintenance of scientific software
- Working knowledge of Linux systems administration in a biomedical research setting and/or scientific computing and/or communications with LINUX and UNIX-based systems (including Mac OS X)
- Network administration skills will be considered a plus
- Fluency in English is essential.
- Windows administrators will NOT be considered.

So what does work for bioinformatics?

Software-wise

Operating Systems based on UNIX

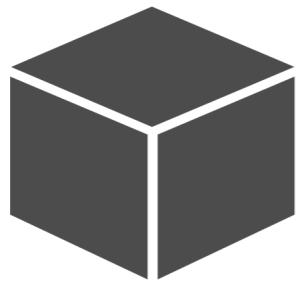


FreeBSD



ORACLE
SOLARIS

Black boxes: commercial packages



geneious

CLC bio

Open source: command line, programming languages



python



qiime
Quantitative Insights Into Microbial Ecology

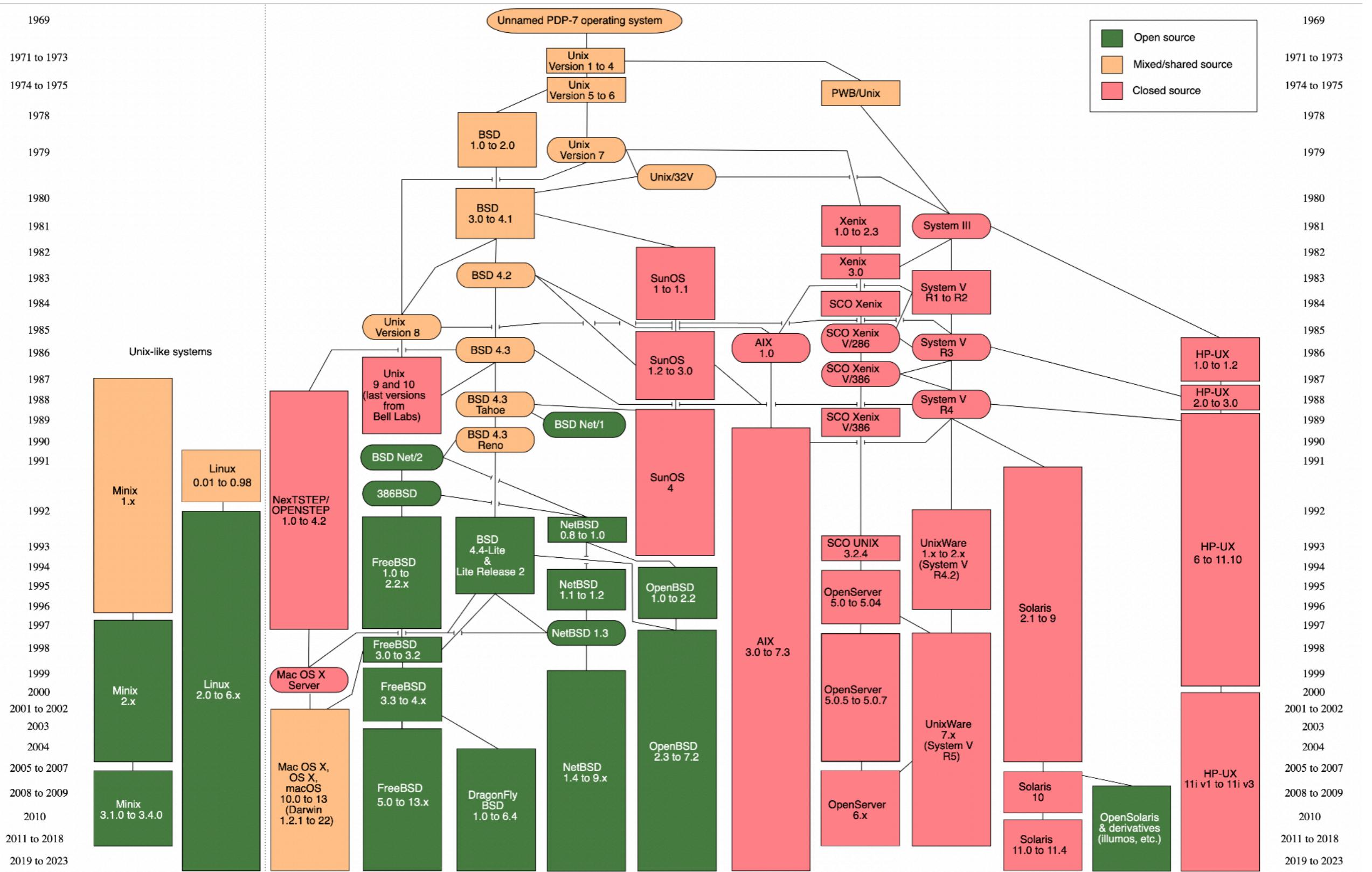
UNIX-based Operating Systems

- Initially developed in **1969** (Bell Labs / AT&T)
- Coded mostly in C (also developed by AT&T)
- **Muti-tasking** (resources are shared)
- **Multiuser** (can handle many users simultaneously)

Today used in:

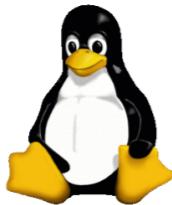
- **Large supercomputers**
- **Computer clusters**
- Servers
- PC
- Mobiles

The UNIX phylogenetic tree



UNIX-based Operating Systems

Pros



- Stability
- Security
- Good control of users and processes
- Can handle large computing load
- Several FREE (as in 'speech') options: Open Source.
- Several FREE (as in 'beer') options: (Ubuntu, CentOS, Debian, etc.)
- Large community of developers
- Large amount of free software (specially for bioinfo)
- Cute mascot

UNIX-based Operating Systems

Cons



- Some software may not be supported
- You may need more knowledge on computers to use it
- Less user friendly than other options (although this may be less true nowadays unless you have to use the **CLI***).

*SPOILER: you'll need to use the CLI (Command Line Interface)

So what does work for bioinformatics?

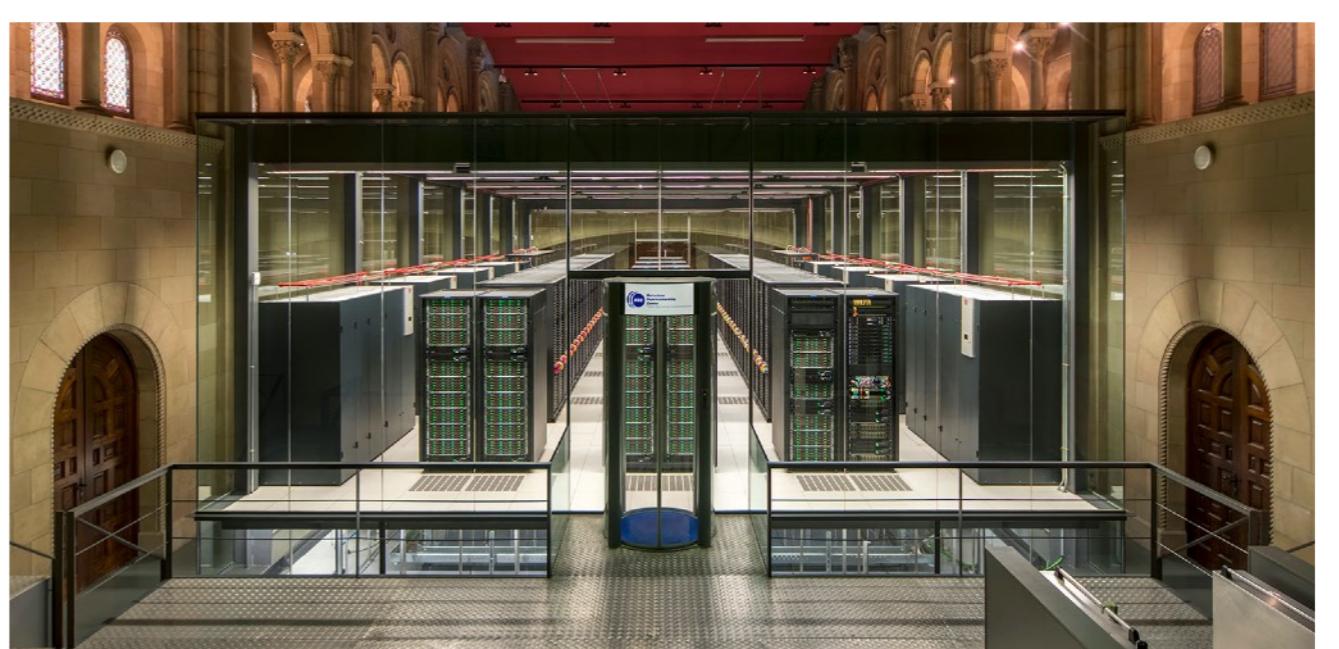
Hardware-wise

Cloud computing

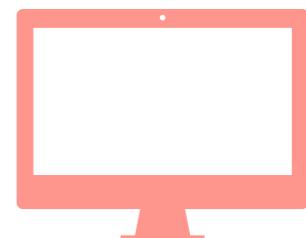


Google Cloud

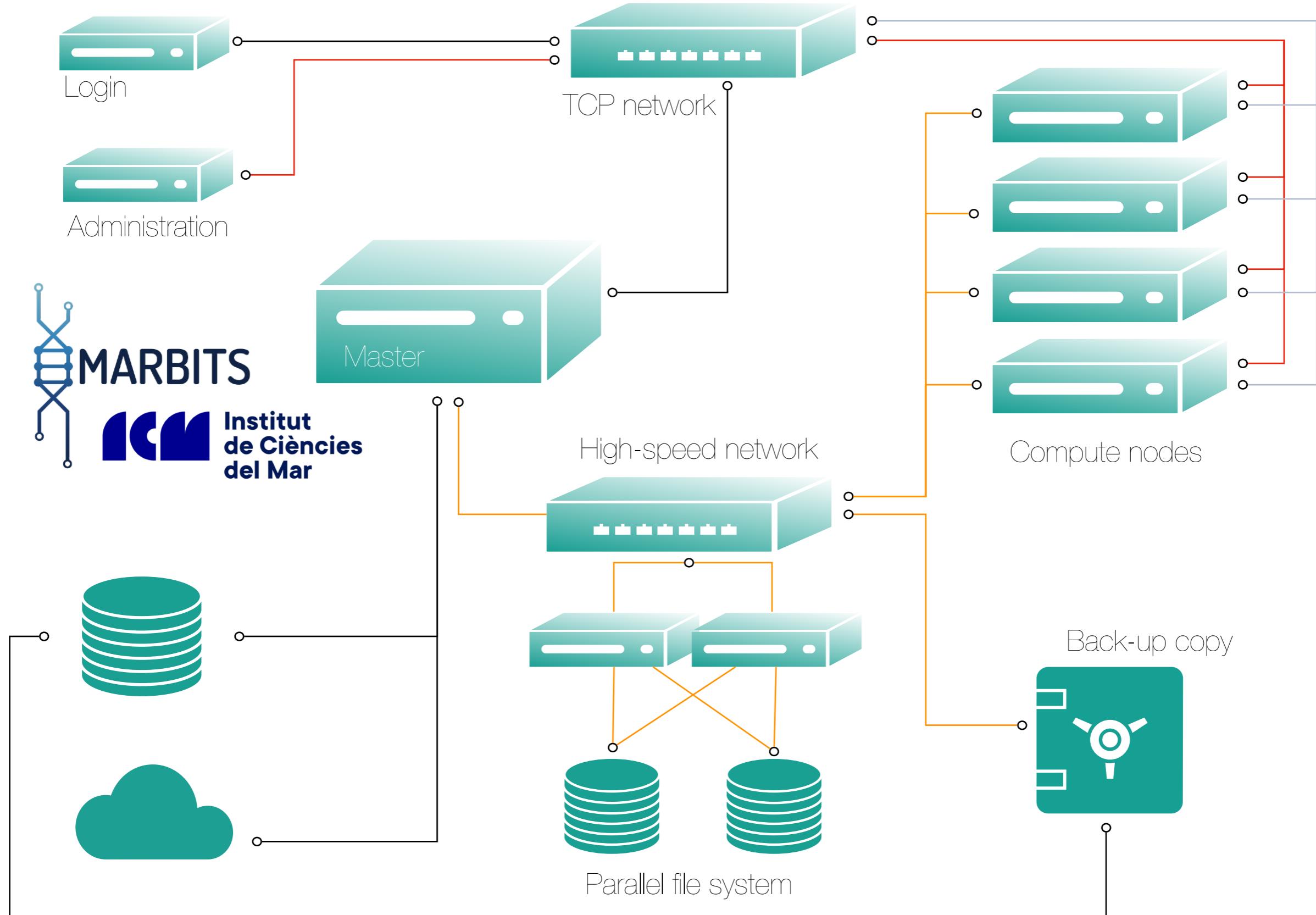
High Performance Computing Clusters



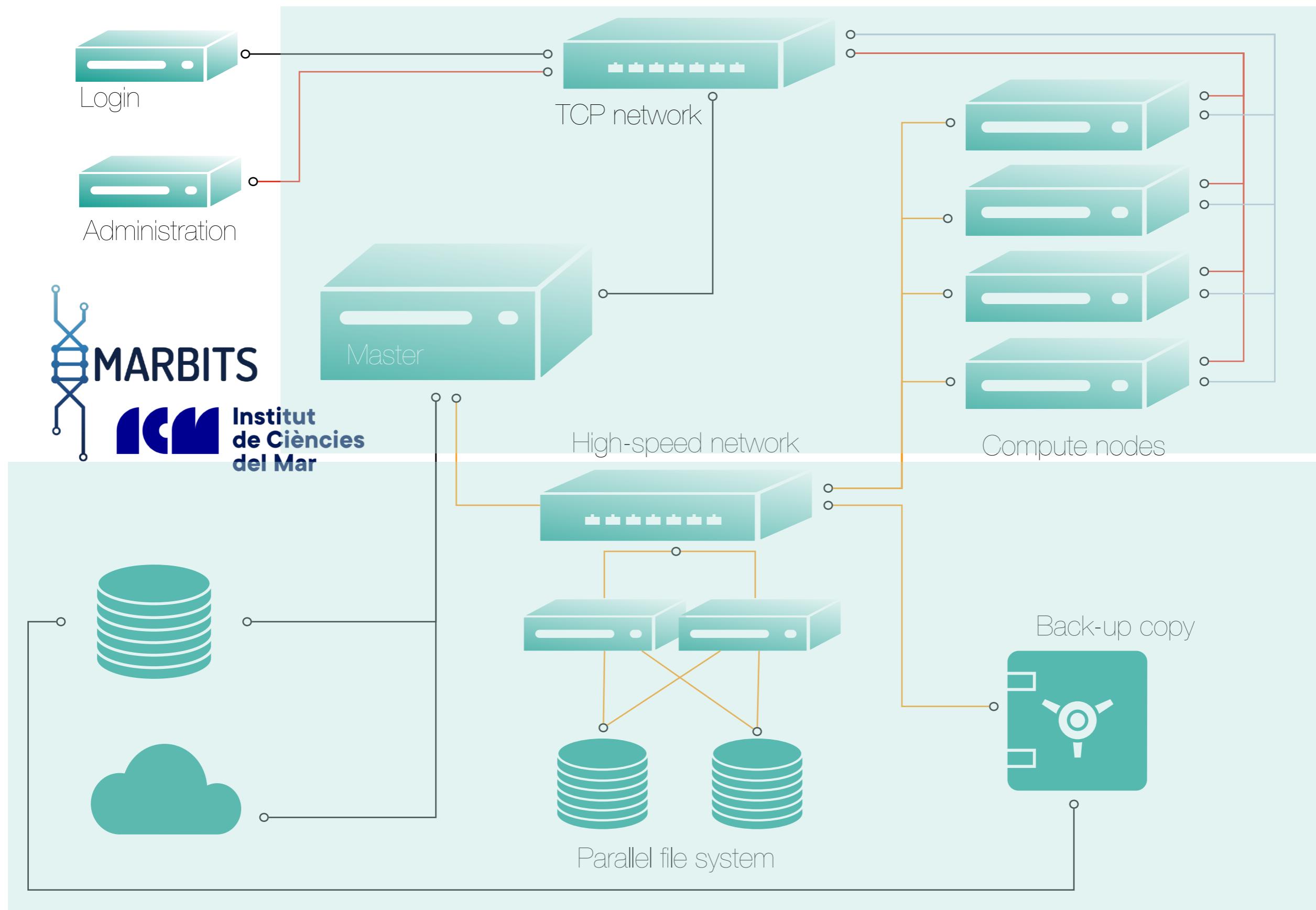
Linux workstations (well, and maybe your unix-based laptop too)



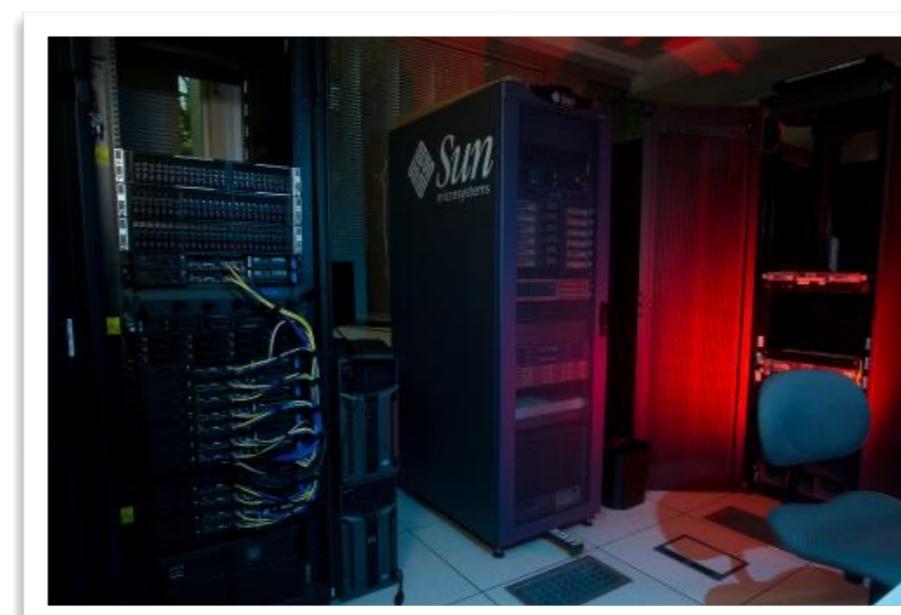
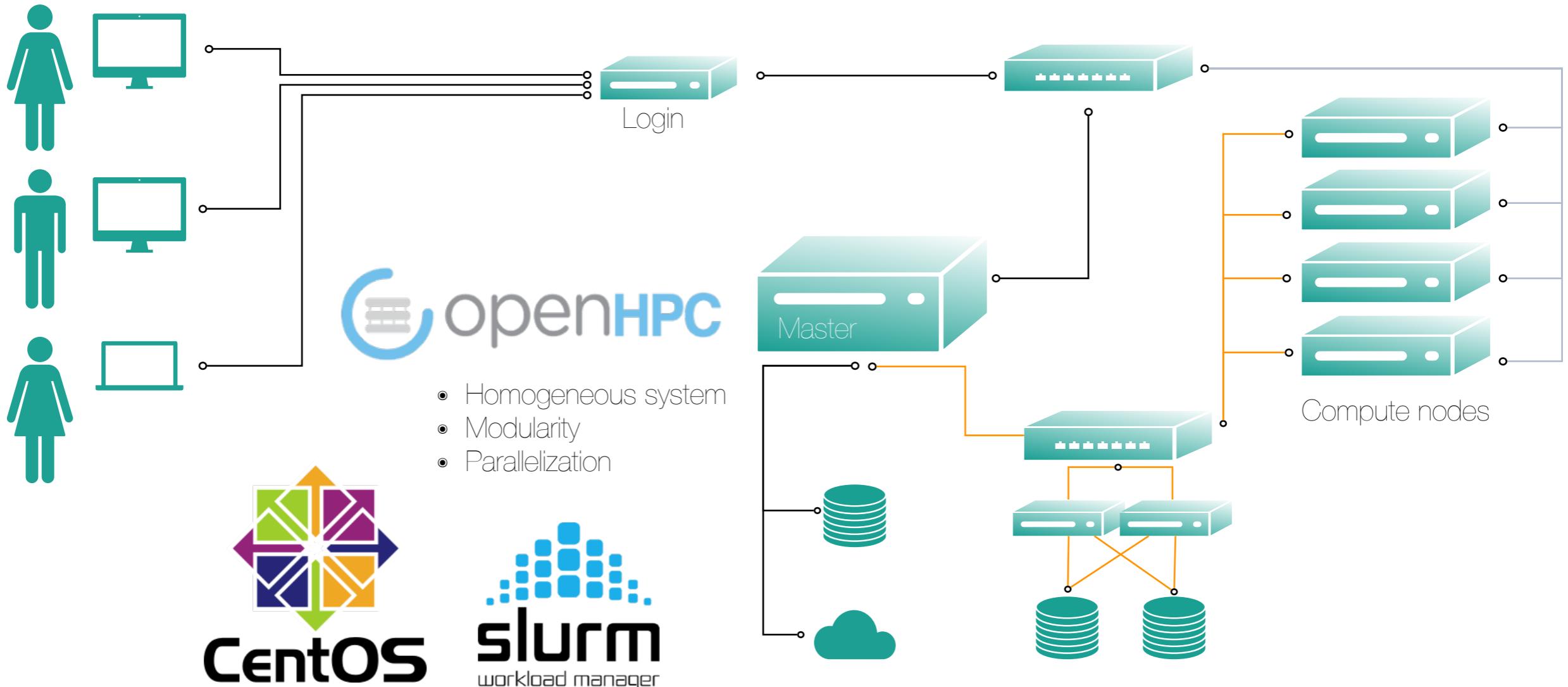
HIGH PERFORMANCE COMPUTING SYSTEMS (HPC)



HIGH PERFORMANCE COMPUTING SYSTEMS (HPC)



HIGH PERFORMANCE COMPUTING SYSTEMS (HPC)



ICM Institut
de Ciències
del Mar

MARBITS

WHERE IN THE WORLD IS MARBITS?

CPD Room. Floor 0, mountain corridor. Restricted access.



HPC history @ICM

~2012: Shared bioinformatics 'infrastructure' started with Ramiro Logares' Dell server *ironbabe*.
Scaled up with 2 more Dell servers (*kyuss* & *hotdog*)

CPU

2 x 4-core CPU = 16 cores

RAM

16 Gb

STORAGE

2 TB



HPC history @ICM

~2013: Upgrade to an IBM iDataPlex HPC cluster, *biocluster*.
This was a serious rig. @CPD CMiMA.



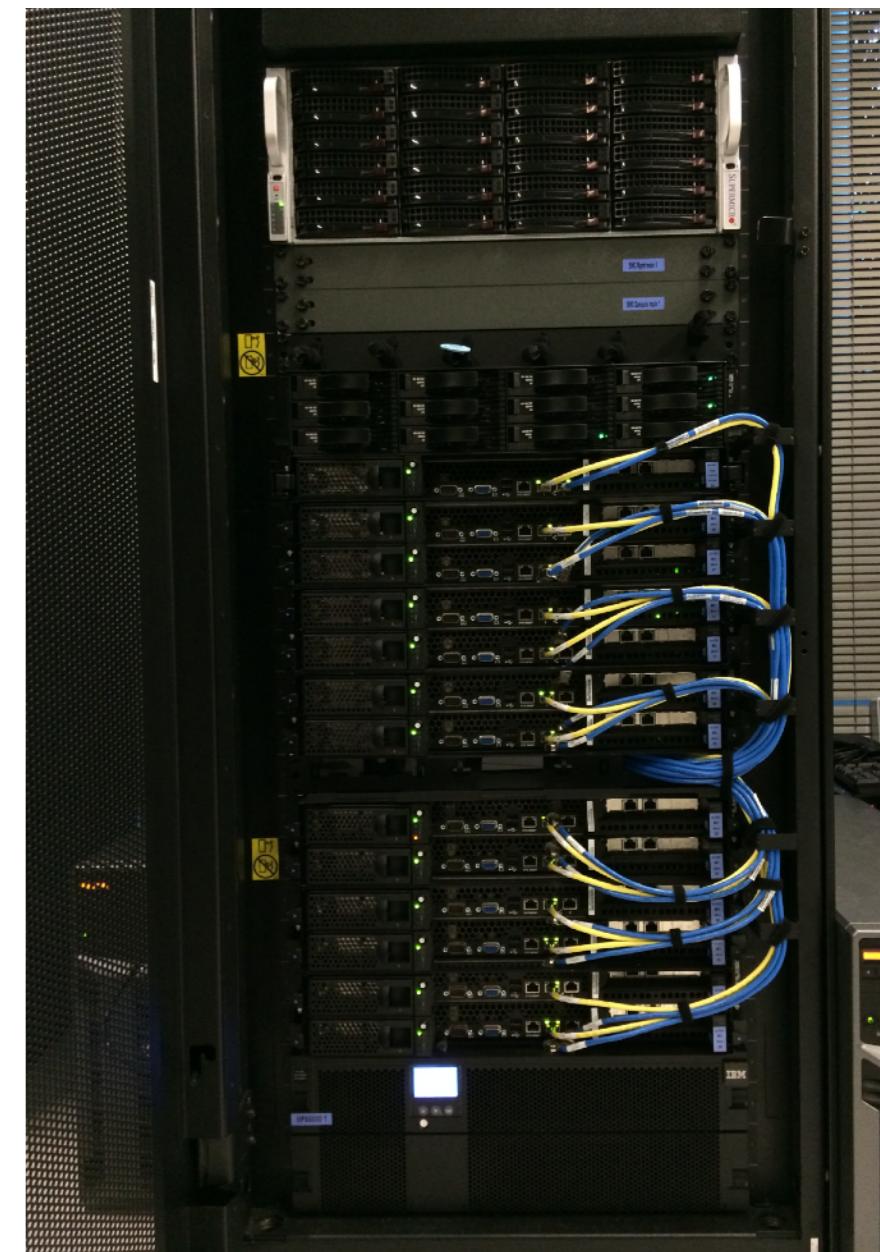
2 x 6-core CPU = 12 cores per node. Total = 144 cores



80 GB per node. Total ~1 TB



154 TB of raw storage. NFS.
Fragmented as hell.



HPC history @ICM

~2018: Marbits. New HPC cluster from scratch, although it integrates all the old biocluster parts. Granted to BMiO via an infrastructure project (MINECO/FEDER; CSIC15-EE-3579; 104,000€)

10 new compute nodes (supermicro)

CPU

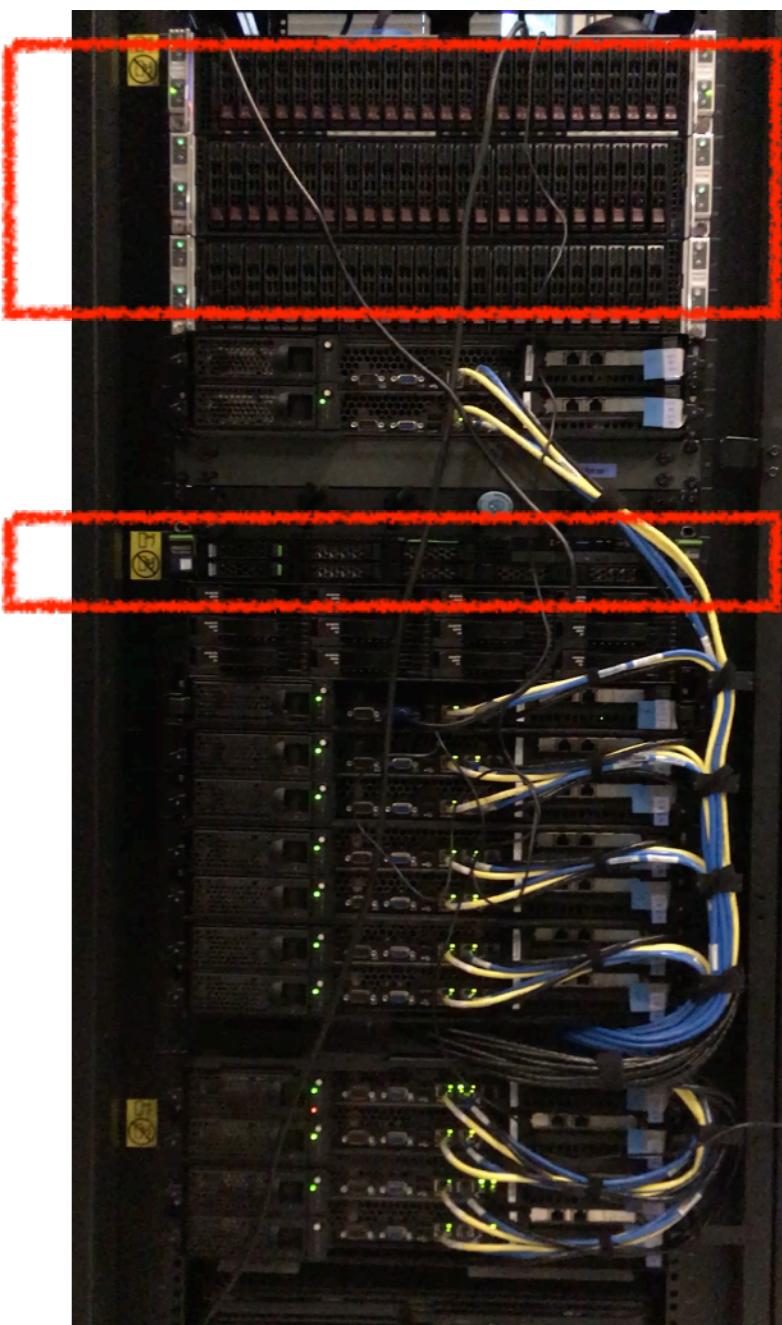
2 x 12-core CPU = 24 cores per node. Total = 240 cores

RAM

256 GB per node (8x), plus 772 GB per node (2x). Total ~3.7 TB

STORAGE

Integration in the Lustre filesystem at CPD. 450 TB.



Marine Bioinformatics Service

It is a service integrated within CSIC's services structure.

The Marine Bioinformatics Service (SBM) offers integrated solutions for the management, storage and analysis of massive DNA sequencing of marine microorganisms, although the infrastructure and functioning makes it open to work with other marine organisms (fish, cnidarians, etc). The SBM attributions go from complete sequencing data analysis to the formation of the users in computational biology techniques, going through the management and development of customized software.

This means that since 2019, research groups that use Marbits are billed according to the CPU usage.

Marine Bioinformatics Service



Pablo Sánchez



Ramiro Logares Haurie



Ramon Massana



Josep M. Gasol



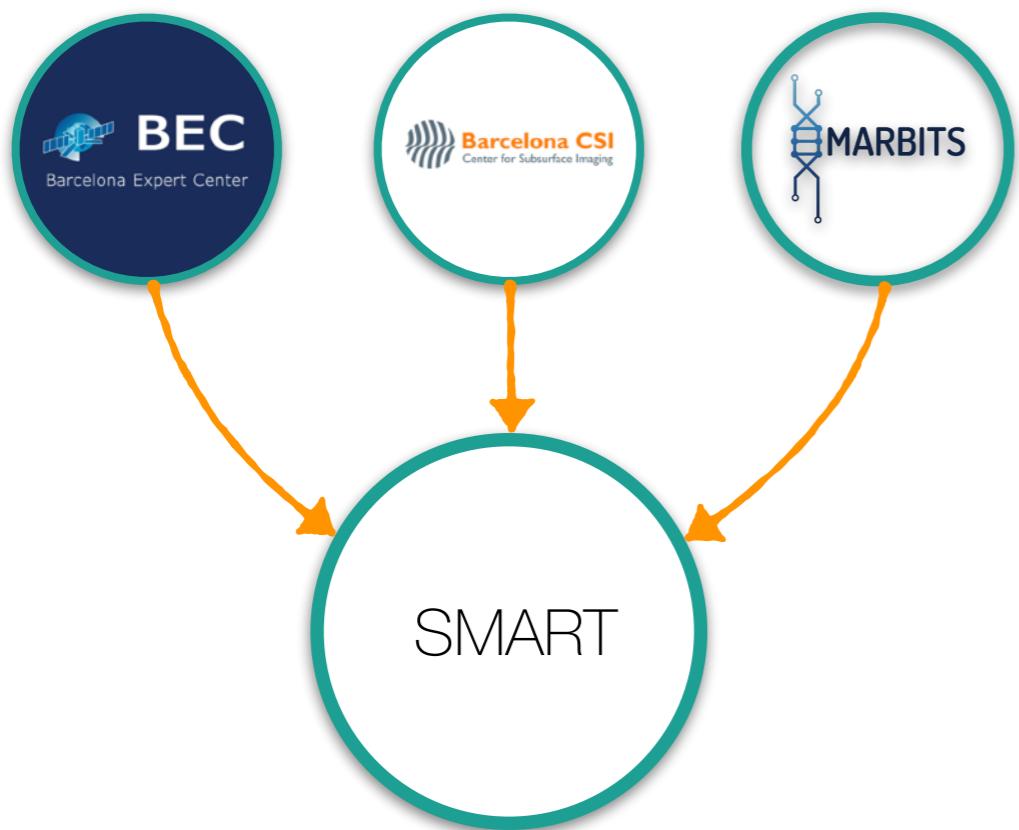
Silvia G. Acinas



Dolors Vaqué

FUTURE OF HPC @ICM

Infrastructure

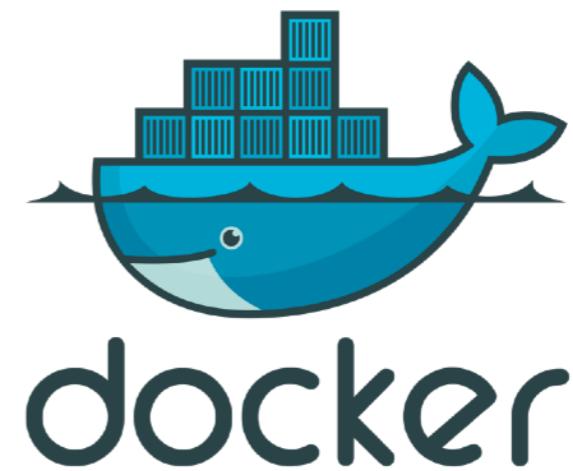


Cloud transition



Applications

Dockerization



Workflows:reproducibility



What can MARBITS* do for you?

- Run jobs in parallel within a node, up to 48 processes (with hyperthreading)
- Run jobs in parallel across nodes (openMPI)
- Run jobs that require large amounts of RAM (up to 772 Gb)
- Access to biological information databases stored locally
- Access to a large list of data science and bioinformatics software (modules)
- Data sharing between members of your team
- Data backup

*or any HPC cluster

What can't MARBITS do for me?

- It can't build your job scripts for you, nor design or decide the best tool for a job, nor do an informed interpretation of your results.
- It can't store your files forever

DISCLAIMER

You, the user, are the final and only responsible for the integrity of any original data that you may upload to the cluster, as well as of any analysis-derived result.

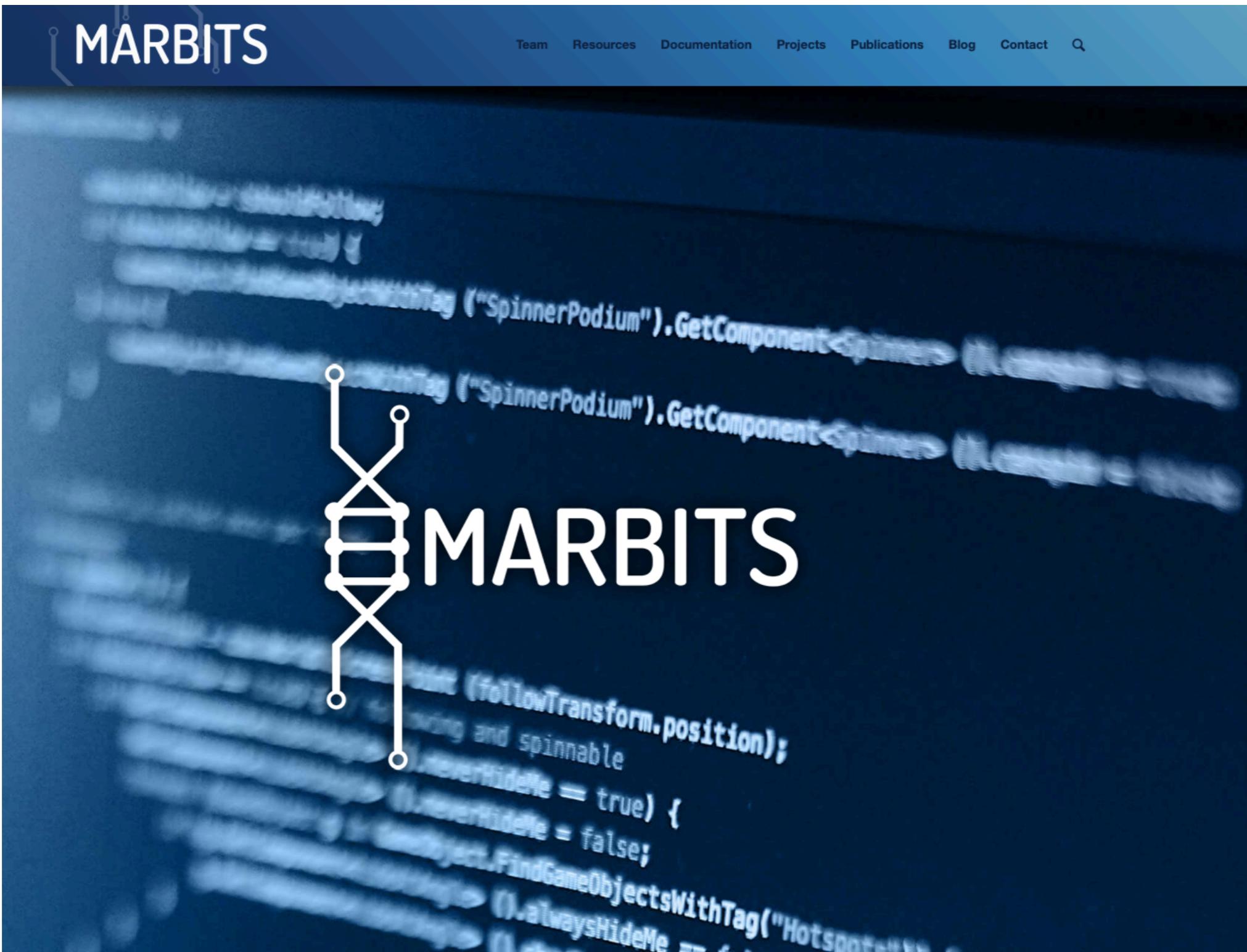
REMEMBER: MARBITS is not a long-term data storage facility.

Where can I get HELP?



marbits.slack.com

Where can I get HELP?



marbits.icm.csic.es

Where can I get HELP?



GitLab

gitlab.com/marbits/marbits.documentation/-/wikis

Where can I get HELP?



DuckDuckGo

Google

Your favourite evil search engine

How do I log into marbits?

You need to have a marbits account and use the SSH protocol within an xterm terminal in linux or macOS. Windows doesn't speak xterm natively, so you need an emulator like PuTTY.

```
ssh your_user_name@marbits.cmima.csic.es
```

- You have to be within the local ICM network. It means:
- Physically plugged to an ethernet cable*
- Remotely with the Eden VPN
- Remotely using a bridge server (salamandra). Access managed by IT

*with a valid IP address

Login

CLI

Command Line Interface

Read-Evaluate-Print loop

The command Shell

or just *the shell*

A shell is the outermost layer of an OS. It is a program that exposes the OS services to a human or other program.

Pros:

- high action-to-keystroke ratio
- support for **automating repetitive** tasks
- **reproducibility**
- it can be used to **access networked machines**.

Cons:

- user unfriendly (in the beginning)
- longer learning curve

The command Shell

or just *the shell*

There are several shells:

- sh
- bash
- csh
- zsh

The default in marbits is **bash**.

The command Shell. Let's do this!

```
[username@marbits ~]$
```

- \$ is a prompt. It is the shell waiting for your input
- marbits prompt shows the user's current identity, the host name and the current directory.

Your first command. Print your user name

```
$ whoami
```

Interacting with files and directories

- The file system is responsible for managing information on the disk.
- Information is stored in files, which are stored in directories (folders).
- Directories can also store other directories, which forms a directory tree.

```
directory-1
├── code
│   ├── 01_read-trimming-fastp.slurm
│   ├── 02_multqc.slurm
│   ├── 03_assembly.slurm
│   └── 04_summarize-results.r
├── data
│   ├── 01_raw-sequences.fastq.gz
│   └── 02_metadata.txt
└── output
    ├── analysis
    │   └── 01_summary-counts.tsv
    ├── figures
    │   ├── 01_figure-1.png
    │   └── 02_figure-2.png
    └── slides
        └── 01_report.key

7 directories, 10 files
```

Interacting with files and directories

- A **path** is a location (the route to a file or directory)
- A **relative path** specifies a location starting from the current location.
- An **absolute path** specifies a location from the root of the file system.
- The path to your **home directory** is usually `/home/your_name` or, simply, `~/`
- `/` on its own is the root directory of the whole file system.

```
directory-1
├── code
│   ├── 01_read-trimming-fastp.slurm
│   ├── 02_multqc.slurm
│   ├── 03_assembly.slurm
│   └── 04_summarize-results.r
└── data
    ├── 01_raw-sequences.fastq.gz
    └── 02_metadata.txt
└── output
    ├── analysis
    │   └── 01_summary-counts.tsv
    ├── figures
    │   ├── 01_figure-1.png
    │   └── 02_figure-2.png
    └── slides
        └── 01_report.key

7 directories, 10 files
```

Interacting with files and directories

- Directory names in a path are separated with `/` on Unix, but `\` on Windows.
- `..` means "the directory above the current one"; `.` on its own means "the current directory".
- Most file names are `something.extension`. The extension isn't required but is normally used to indicate the type of data in the file.

Path examples:

`/home/username/directory-1/data/01_raw.fastq.gz`

`output/figures/01_figure-1.png`

You are here: `pwd` and `ls`

Print your Working Directory

```
$ pwd
```

List the contents of your working directory

```
$ ls
```

Command options and arguments: `ls`

You can list the contents of any path in the filesystem, passing a path as an **argument** to the `ls` command

```
$ ls /mnt/lustre/scratch/laPera
```

You have listed contents of an absolute path. You may see that some names appear in different colours. This is a way to easily spot directories and other type of files (and even properties of the files)

You can modify the behaviour of the `ls` command with **options**

```
$ ls -l -h /mnt/lustre/scratch/laPera
```

Options can be condensed behind a single `-' . Try different options for ls and see what happens:

```
ls -1  
ls -lr  
ls -lrt  
ls -lh
```

Command options and arguments: ls

When using option **-l** or long listing, we see lots of information on the objects stored in a directory.

```
[psanchez@marbits:nextflow]$ ls -lh
total 253M
-rwxr-xr-x 1 psanchez bio 253M ene 12 10:43 fastqc:0.11.9--0
-rw-r--r-- 1 psanchez bio 152 ene 12 17:46 nextflow.config
drwxr-xr-x 3 psanchez bio 4,0K ene 11 17:59 results
drwxr-xr-x 40 psanchez bio 4,0K ene 16 11:39 work
```

permissions

owner group

size

timestamp

file/dir name

drwxr-xr-x
owner group other

Change permissions with **chmod**

```
$ chmod u+x filename
$ chmod g+w filename
$ chmod o-rx filename
```

What?? this is too much, I need help!

man to the rescue!

- a man page (manual page) is the software documentation usually found in *nix systems. All you need to know should be in a man page.
- You can invoke the man page for (almost) any command in your system. Just type

```
$ man ls  
$ man chmod
```

- Use the arrows to navigate the page, space bar to jump one screen forward, 'b' to jump one screen backwards and 'q' to quit the man page.

R T F N M



Navigating the filesystem

Change directory with **cd**. With no arguments, **cd** leads you to your **home** dir.

```
$ cd
```

cd can use absolute or relative paths as arguments. Change directory to the course directory:

```
$ cd /mnt/lustre/scratch/laPera
```

check it with **pwd** and **ls**

```
$ pwd  
$ ls
```

Notice that the prompt has changed to show the current directory.

Navigating the filesystem. Create directories

- Let's make some directories within our home, so as we have a nice framework to start working.

```
$ cd ~/  
$ mkdir hpc-course  
$ cd hpc-course  
$ mkdir my-project  
$ cd my-project  
$ mkdir data  
$ mkdir code  
$ mkdir output  
$ cd output  
$ mkdir logs  
$ mkdir figures
```

```
[psanchez@marbits:hpc-course]$ tree my-project/  
my-project/  
├── code  
├── data  
└── output  
    ├── figures  
    └── logs  
  
5 directories, 0 files
```

Navigating the filesystem

Going back one level

```
$ cd ..
```

Going back several levels

```
$ cd ../../..
```

You can get creative with your cd arguments

```
$ cd ~/hpc-course/my-project/output/figures  
$ cd ../../data
```

The second command moves you back two steps, to the branching point of the main directory of the project right into the data directory... all in just one command

Navigating the filesystem. Creating directories

You don't need to be in the parent directory to create a new one

```
$ cd ~/hpc-course/my-project  
$ mkdir docs  
$ mkdir docs/writing  
$ ls  
$ ls docs
```

As you see, we need to specify the full path (absolute or relative) to the location where we want the directory to be created (also to see it listed).

Use option '**-p**' to create elaborate directory structures

```
$ cd ~/hpc-course  
$ mkdir -p my-project2/{code,data,docs,output}  
$ mkdir -p my-project3/{code,data,docs/write,output/{logs,figures}}
```

It can become a curly-braces nightmare very quick, but do it once and copy-paste it as many times as you need. Forget about clicking and naming folders!

Be a keyboard ninja



- Working without a mouse, in a shell, writing commands... I don't see the supposed speed gain over the GUI
 - Behold the **keyboard shortcuts!**
-
- **Tab** autocompletes the name of a file, directory or program. Just write a few letters and hit Tab to try it.
 - **Up** and **Down** arrows cycle through your history of commands, so you don't need to re-type some things
 - **^A** and **^E** moves the cursor to the beginning and end of the line respectively
 - **^K** deletes all the line from the cursor to the right
 - **^U** deletes the whole line from the cursor to the left
 - **^W** deletes a whole word to the left of the cursor
 - **Alt + right or left arrow** moves the cursor one word to the right or left

Navigating the filesystem. Removing directories

We have many empty directories that we can remove. We can use the command **rmdir**. Directories must be empty, so we have to navigate to a terminal directory (minus one), i.e. in **my-project2** let's remove the **output** directory. You must be outside of a directory to remove it. Remember Tab autocomplete!

```
$ cd ~/hpc-course/my-project2/output  
$ rmdir figures  
$ rmdir logs  
$ cd ..  
$ rmdir output
```

→ Try to remove **my-project2**

```
$ cd ~/hpc-course/my-project2/output  
$ rmdir my-project2
```

If you try to remove a non-empty directory, the command will miserably fail.

Navigating the filesystem. Globbing

glob patterns specify sets of filenames with wildcard characters. Placeholder represented by a single character, such as an asterisk (*), which can be interpreted as a number of literal characters or an empty string. For instance, **catorcea.*** means all files named 'catorcea' independently of their extension (pdf, docx...)

```
* matches 0 or more characters  
? matches a single character (including end of line)  
[ ] matches a group of characters. i.e: [A-Za-z] or [1-3]
```

→ In /mnt/lustre/scratch/laPera/00_materials/dummyFilesRepo try to list all files

- with extension **png**
- that have a name like **nt.0X.nin**, where X is a number
- same as above but only for files with numbers 00 to 04 and with extension beginning by 'n' and ending on 'i'

Navigating the filesystem. Copying files

Now we want some files to work with. As our workspace is mostly empty, we could upload files, create them from scratch, or just copy them from a source location. The **cp** command does just that. You need to specify **2 arguments**: a source and a target location for your file. At your **~/hpc-course** directory:

```
$ cd ~/hpc-course
$ mkdir experiments
$ cp /mnt/lustre/scratch/laPera/00_materials/exp/01_experiment-A.csv experiments
$ cp /mnt/lustre/scratch/laPera/00_materials/exp/02_experiment-A.csv experiments
$ cp /mnt/lustre/scratch/laPera/00_materials/exp/03_experiment-A.csv experiments
$ cp /mnt/lustre/scratch/laPera/00_materials/exp/04_experiment-B.csv experiments
$ cp /mnt/lustre/scratch/laPera/00_materials/exp/05_experiment-B.csv experiments
$ cp /mnt/lustre/scratch/laPera/00_materials/exp/06_experiment-B.csv experiments
```

Or just remember globbing and copy many files at once (all of them, only experiment A or only experiment B):

```
$ cp /mnt/lustre/scratch/laPera/00_materials/exp/*.csv experiments
$ cp /mnt/lustre/scratch/laPera/00_materials/exp/*-A.csv experiments
$ cp /mnt/lustre/scratch/laPera/00_materials/exp/*-B.csv experiments
```

Navigating the filesystem. Moving files

Moving files is similar to copying them. Use command **mv**. And be very careful! Moving a file and renaming files use the same command and it can be very destructive. Use option **-i** if you want to be on the safe side.

```
$ cd ~/hpc-course  
$ mkdir tmp  
$ mv experiments/01_experiment-A.csv tmp  
$ mv experiments/02_experiment-A.csv tmp
```

If you don't specify a file name in cp or mv, the shell will use the original file names to complete the target path. Try to overwrite a file giving the full path as target (directory + file name)

```
$ mv -i experiments/03_experiment-A.csv tmp/02_experiment-A.csv
```

Renaming a file is like moving it to the same location but with a different name

```
$ mv -i tmp/01_experiment-A.csv tmp/01_failed.csv
```

Navigating the filesystem. Removing files

We saw how to remove empty directories. If the directory is not empty we first need to remove files with the command **rm**.

ATTENTION: rm is very dangerous. There's no recycle bin here. When something is gone, *is gone*. Please use option **-i** to make it a bit safer. Seriously. You can destroy everything in your home directory with **rm**.

```
$ cd ~/hpc-course/tmp  
$ rm -i 01_failed.csv  
$ rm -i 02_experiment-A.csv  
$ cd ..  
$ rmdir tmp
```

Remember you can glob file names. This is why this is so dangerous! you could easily do

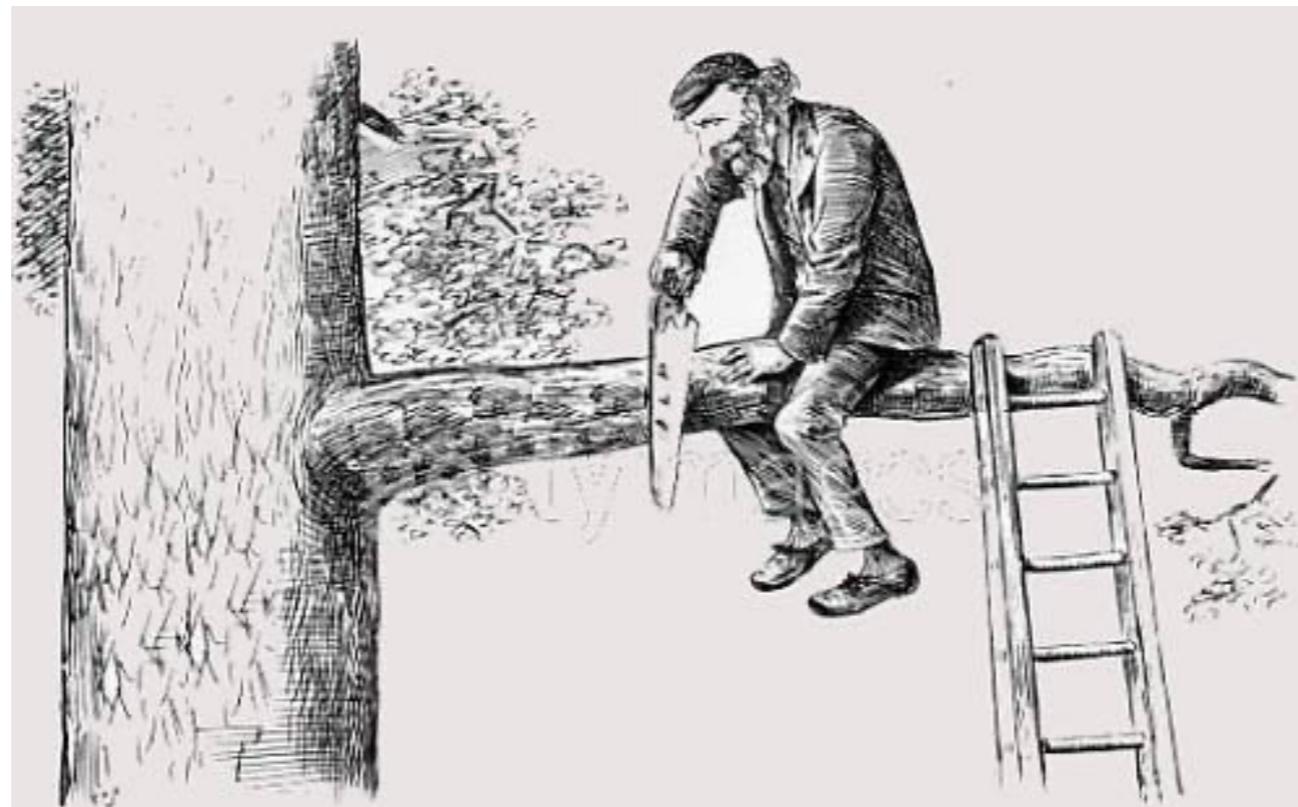
```
$ rm experiments/*.csv
```

And delete all .csv files. Use option **-i** until you are comfortable with **rm**

Navigating the filesystem. Removing files

You can remove a non-empty directory with all its contents with options
-r (recursive) and **-f** (force)

```
$ rm -rf directory name
```



Careless bioinformatician typing **rm -rf *** in his home directory

Navigating the filesystem. Copying files or directories with `rsync`

`rsync` is a fast and extraordinarily versatile file copying tool. It can copy locally, or to/from another host over any remote shell. The syntax is similar to `cp`, but has lots of options and reduces the amount of data sent over the network by sending only the differences between the source files and the existing files in the destination.

Typically:

```
$ rsync -avz <source> <destination>
$ rsync -avz --progress --partial <source> <destination>
```

→ Try it with option `--dry-run` and copy the directory `/mnt/lustre/scratch/laPera/` to your `hpc-course` directory. Notice that adding a trailing '`/`' won't create a new directory in the destination location!
If you are happy with the results, do it again without the `--dry-run` option

Navigating the filesystem. Copying remote files or directories with `rsync`

The syntax to copy files **from marbits** to your computer is, **from your laptop:**

```
$ rsync -avz --progress --partial username@marbits:~/hpc-course/experiments ./
```

The syntax to copy files **from your laptop** to marbits is, **from your laptop:**

```
$ rsync -avz --progress --partial my-local-file username@marbits:~/hpc-course
```

Exploring file contents

The raw text file is the typical file you are going to deal with in bioinformatics: sequence files, annotation files, alignment files, scripts... We have 3 commands to visualise the content of a text file. Try all 3 and discuss the differences.

```
$ cat /mnt/lustre/scratch/laPera/00_materials/turra.txt  
$ more /mnt/lustre/scratch/laPera/00_materials/turra.txt  
$ less /mnt/lustre/scratch/laPera/00_materials/turra.txt
```

Exploring file contents

Sometimes dumping all file contents to the screen is overkill. We can extract only a few lines from the beginning or the end of the file with **head** and **tail**.

```
$ head /mnt/lustre/scratch/laPera/00_materials/turra.txt  
$ tail /mnt/lustre/scratch/laPera/00_materials/turra.txt
```

head and **tail** accept a number of lines as the argument for option **-n**.

- Try to get the last 20 lines of file `turra.txt` from the beginning or the end of the file `turra.txt`

History

Your command history is stored in a file at `~/ .history`. The ‘.’ before the file name indicates that it is a hidden file. You can reveal hidden files with `ls -a`

```
$ cd ~/  
$ ls -a
```

- The command **history** shows the contents of `~/ .history`
- **!123** executes again the command number 123 in your history
- **!command** executes again the last instance of the specified command
- **!!** executes your last command one more time
- **command !\$** executes ‘**command**’ on the last argument of the previous command. i. e.:

```
$ ls -lh /mnt/lustre/scratch/laPera/00_materials/turra.txt  
$ ls -lh !$
```

Creating and editing text files

We have seen that the text file is the most common file in bioinformatics. How can we create or edit a text file within the shell?

We have several built-in text editors in marbits:

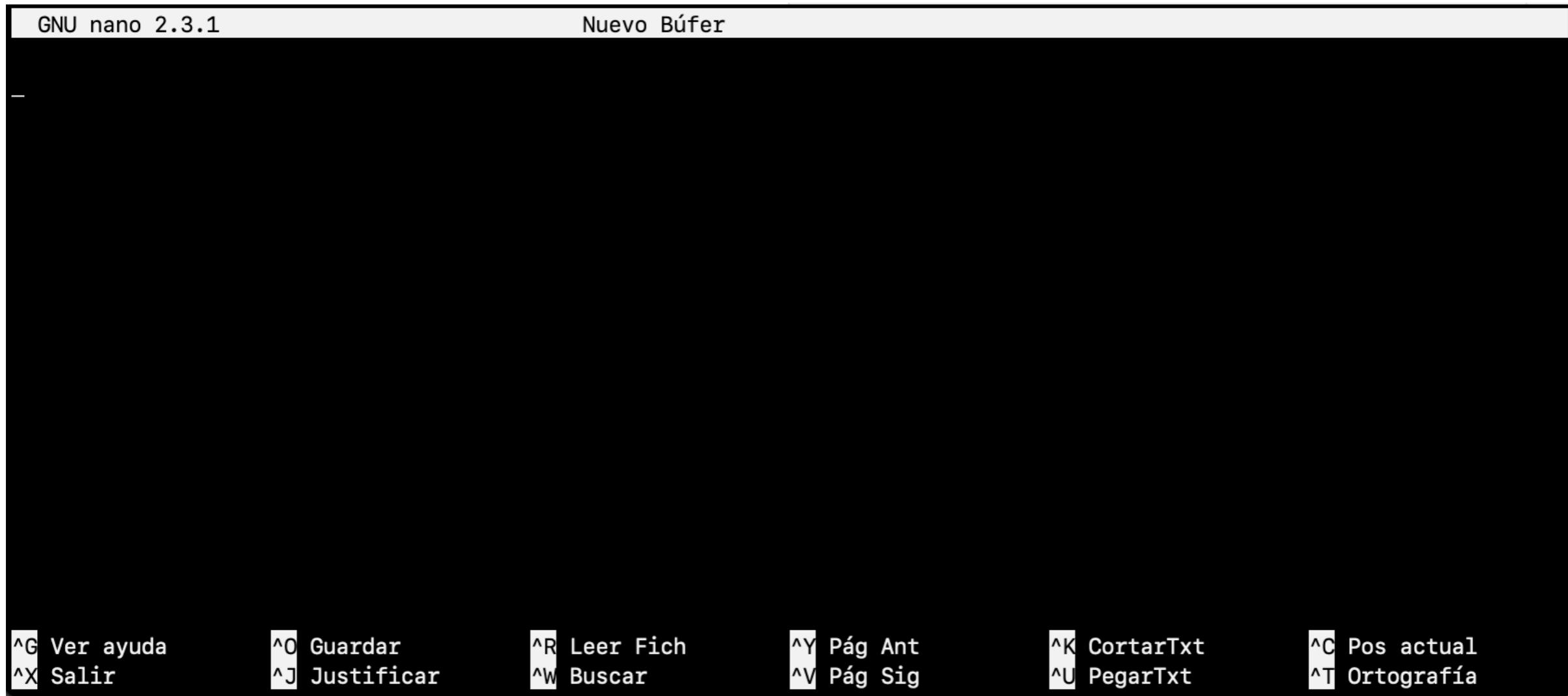
- vim
- emacs
- nano

Which one to use is a matter of personal preference. Let's start with nano (the easiest). nano can be called without arguments and start with a new text file, or give a file name as argument.

```
$ cd ~/hpc-course  
$ nano
```

Creating and editing text files. nano

You can start writing right away. It is quite straightforward and very similar to a typical and very basic text processor (no formats here). Try it.



→ After finishing your new file/edit you can use the control key (^) and one of the keys specified in the bottom of the window to do some actions like save, find, cut and paste, exit...

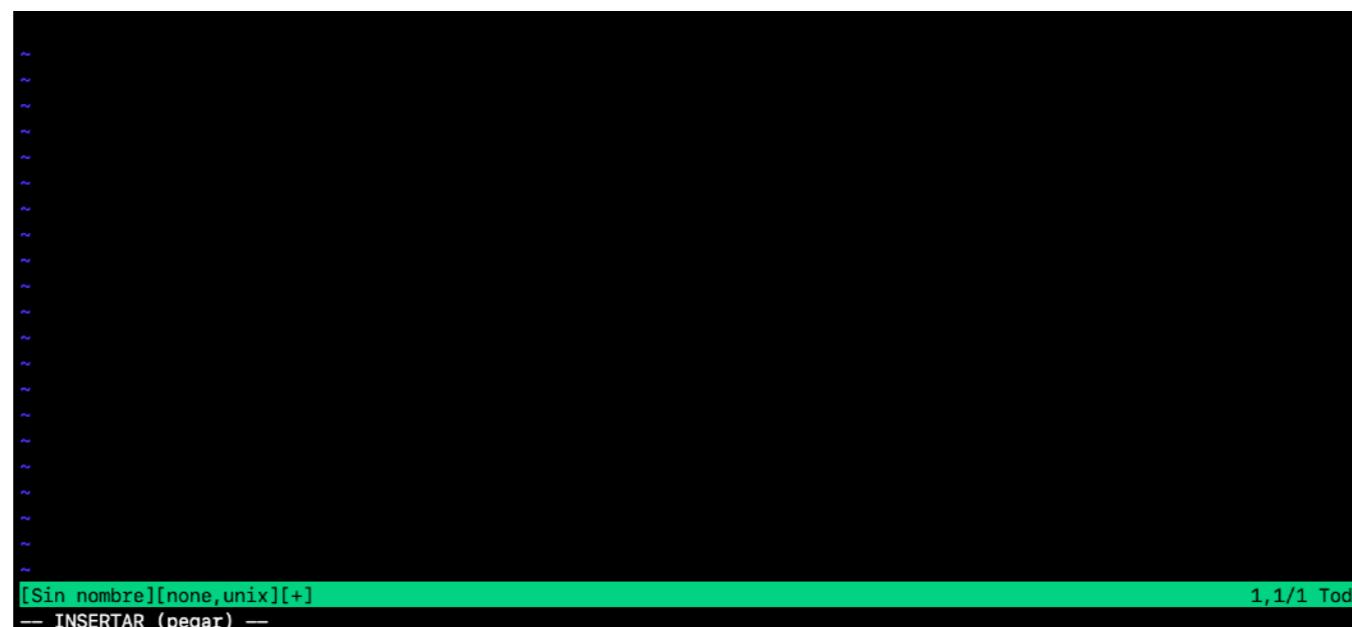
Creating and editing text files. vi/vim

vi (pronounced as distinct letters) is a screen-oriented text editor originally created for the Unix operating system in 1976. Vim isn't an editor designed to hold its users' hands. It is a tool, the use of which must be learned.

Vim "Vi IMproved" has many additional features compared to vi, including (scriptable) syntax highlighting, mouse support, graphical versions, visual mode, many new editing commands and a large amount of extension in the area of ex commands. Vim is included with almost every Linux distribution (and is also shipped with every copy of Apple macOS)

Vi is a modal editor. It operates in "insert" mode or in "command" mode.

For example, typing **i** while in command mode switches the editor to insert mode, but typing **i** again at this point places an "i" character in the document. From insert mode, pressing **ESC** switches the editor back to command mode.



Creating and editing text files. vi/vim

vi cheatsheet

→ Try to edit the file you created in nano with vim

The Unofficial Beginner's Vim Cheat Sheet

Exiting Vim

:w - Write (Save)
:wq - Write and quit
:q - Quit, fails if unsaved
:q! - Quit, even if unsaved

Movement

\$ - Jump to end of line
^ - Jump to start of line
h - Move left
j - Move down
k - Move up
l - Move right
H - Move to top of screen
M - Move to middle of screen
L - Move to bottom of screen
gg - Move to start of file
G - Move to end of file
420gg - Move to line 420
w - Jump to start of next word
b - Jump to start of prev word

Modes

ESC - Return to normal mode
i - Insert at cursor position
a - Insert after cursor position
o - Insert on line below cursor
v - Enter visual mode
ctrl+v - Enter visual mode (vertical)
V - Enter visual mode (full lines)

Toggling Case

u [in visual mode] - To lowercase u - Undo
U [in visual mode] - To uppercase ctrl+r - Redo

Search

/something - Search for string
n - Jump to next match
N - Jump to prev match
/something\c - Case insensitive search

Copy-Pasting

y [in visual mode] - Copy highlighted text
yy - Copy the current line
d [in visual mode] - Cut highlighted text
dd - Cut the current line
Ctrl+Shift+V - Paste from external clipboard

Find-and-Replace

Find and Replace All in Document
:%s/find/replace/g
Find and Replace All on Current Line
:s/find/replace/g [in visual mode]
Find and Replace All in Highlighted Section
:'<,'>s/find/replace/g [in visual mode]
Find and Replace All in Document
:%s/address/replace/g
Important Regular Expression (REGEX) Characters
. - Any single character
* - Up to unlimited characters



```
01001101 00000101
11110001 10101000
10100001 10110001
10000110 00011000
11111101 01111000
00110111 01100111
01101101 11010000
11011001 11110001
01101101 00101100
00111100 00000111
```

Vim Cheat Sheet

256kilobytes.com

INFOGRAPHIC BY

256 Kilobytes

Printing strings to the standard output

The command **echo** prints character strings to the standard output*.

```
$ echo 'Hello World!'
```

echo option -e enables the interpretation of backslash escape characters

\a	alert (BEL)
\b	backspace
\c	produce no further output
\e	escape
\f	form feed
\n	new line
\r	carriage return
\t	horizontal tab
\v	vertical tab

→ Try to echo 3 words separated by tabs and a second line that says "bye"

*The communication channels between a computer program and its environment are three: standard input (stdin), standard output (stdout) and standard error (stderr)

Redirecting standard output to a file

the **>** symbol redirects the output of a command to a file. If the file exists it will overwrite it!
If you want to append to the contents of a file use **>>**

```
$ echo 'Hello World!' > hello.txt
```

- Try to redirect the first 25 lines of file **turra.txt** to a file called **miniturra.txt** in your hpc-course directory. Remember the relative path to turra.txt should be **00_materials/turra.txt**

Inspect a file with od

The command **od** writes an unambiguous representation, octal bytes by default, of FILE to standard output. With no FILE, or when FILE is **-**, read standard input.

With option **-c** selects printable characters or backslash escapes

- Use **od -c** to explore the file **hello.txt** you just created, can you see the new line character? Create another file with echo with tabs and explore it with **od -c**

This command is very useful to realize that something is not working because it has the EVIL windows new line character (\r) that comes when saving an Excel file as txt.



Sort a file

The command **sort** writes write sorted concatenation of all FILE(s) to standard output.

Some useful options are:

- n compare according to string numerical value
- r reverse the result of comparisons
- k sort via a key (column)
- u output only the first of an equal run
- c check for sorted input; do not sort
- V natural sort of (version) numbers within text
- h compare human readable numbers

→ Check whether file **00_materials/MP28199.gff** is sorted by the first key. If it is not, sort it by the first key. Then combine the 1st key sort with a numeric sort in reverse of key 4: items in key one will be sorted and, within lines with the same value in column 1, numbers in column 4 must be decreasing.

Count lines, words and characters. wc

The **wc** command prints newline, word, and byte counts for each FILE, and a total line if more than one FILE is specified. With no FILE, or when FILE is -, read standard input. A word is a non-zero-length sequence of characters delimited by white space.

- Check how many lines does file **00_materials/MP28199.gff** have.

Find unique lines in a file. **uniq**

the **uniq** command filters adjacent matching lines from INPUT (or standard input), writing to OUTPUT (or standard output).

→ Check how many lines does file **00_materiales/pokemon.txt** have.

- Now with **uniq** see how many unique pokemons do we have.
- Use option **-c** to get a count of how many pokemons of each are there.
- Is everything ok?
- Check whether the pokemons in the file are sorted.
- Sort them and redirect the output to a file in your home called **sortedPokemons.txt**.
- Repeat the **uniq** command on the new file. Notice something?
- Redirect the unique pokemons to a file **uniquePokemons.txt**
- Use **wc -l** to see how many unique pokemons are there.
- Use option **-d** to get only pokemons that have replicates.

Find differences between two files. `diff`

`diff` compares FILES line by line. It needs 2 arguments (2 file paths/names)

```
$ diff 00_materials/pokemons.txt 00_materials/snometekop.txt
```

→ Use `diff` to compare files **pokemons.txt** and **snometekop.txt**

- Are they identical?
- Option **-y** gives a side-by-side comparison, and with **--suppress-common-lines** it only shows lines with differences.
- What pokemons are different in each file? Try both methods

Print parts of lines. `cut`

`cut` prints selected parts of lines from each FILE to standard output. The idea is that any text file can be a table. You just need to find the proper column separator (comma, tab, :, ;,...).

It allows us to extract any combination of columns of any delimited file (with tab by default) as if it was a table.

→ Use option `-f` and as many "column" indexes separated by ',' to print only columns 1 , 4 and 5 of file **MP28199.gff**.

Use option `--output-delimiter` to change the separation from tab to ';'.

Paste 2 files line by line. `paste`

paste writes lines consisting of the sequentially corresponding lines from each FILE, separated by TABs, to standard output. With no FILE, or when FILE is `-`, read standard input.

It needs 2 file path/names as arguments

- Use paste to reconstruct the full names of files `firstNames.txt` and `lastNames.txt` and redirect it to a file `people.txt`.

Am I using too much disk?. **du**

System admins are ALWAYS complaining (with a cause for reason) that users are hoarding files in the filesystem. This degrades the stability and performance of the filesystem. **du** summarizes disk usage of each FILE, recursively for directories.

► Try **du -sch** on your hpc-course directory

What are options -s, -c and -h doing?

Storing compressed directories. **tar**

GNU **tar** saves many files together into a single tape or disk archive, and can restore individual files from the archive. It is very useful to distribute a complex directory structure in a single file.

- ▶ File **00_materials/project1.tar.gz** is a "tarred" file (or "tarball"). Extension **.tar** indicates that. It is also compressed with the gzip utility (extension **.gz**).

If you want to replicate the tarball directory structure and its contents you need to "untar" it. To "untar" you have to provide some options and one argument (the tar file path/name). Try options **-x** (extract), **-v** (verbose) and **-f** (file) combined as **-xvf**.

Now try to create a tarball from project1 directory structure. You need to give it options **-z** (compress with gzip), **-c** (create tarball) **-v** and **-f**, and provide two arguments: the tarball filename (remember to use the double extension **.tar.gz**) and the directory path/name that you want to tar.

Find text patterns in files. grep

grep searches the named input FILEs (or standard input if no files are named, or if a single hyphen-minus (-) is given as file name) for lines containing a match to the given PATTERN. By default, grep prints the matching lines.

This command is so useful that it is used as a verb. You have to give grep a text string as a first argument and a file name as second argument. You can also add some options to enhance its behaviour.

```
$ grep [options] text_pattern <file_name>
```

→ Remember the file **00_materials/turra.tx**.

- How many lines does it have?
- which lines contain the string 'carrera' in **turra.txt**

Find text patterns in files. grep

- we want to bypass the default behaviour and get **the number** of lines that match the pattern use option **-c**.
- How many lines contain the pattern 'carrera'?

Now let's look for pattern 'funcion'.

- What happens here?
- Use option **-w** to match only lines with the full word 'funcionario'
- How many lines with 'funcionarios' there are?

Find text patterns in files. grep

→ The option **-n** numbers the matching lines. Use it to get the line number of lines with the word 'funcionario'

```
$ grep -n -w 'funcionario' 00_materials/turra.txt
```

We can also make our search case insensitive with option **-i**.

- How many lines does have the pattern 'Ciencia' with capital 'C'?
- How many lines have the pattern ciencia with or without capital 'C'?

Find text patterns in files. **grep**

What makes **grep** REALLY powerful is the ability to use regular expressions. We have to supply the option **-P** (perl) and make use of regular expressions instead of a literal text string.

Regular expressions (regex or regexp) are extremely useful in extracting information from any text by searching for one or more matches of a specific search pattern (i.e. a specific sequence of ASCII or unicode characters).

This is a world in itself. Besides the wildcards that you already know, let's see a few basic examples:

^The	matches a string that starts with 'The'
end\$	matches a string that ends with 'end'
abc+	matches a string that has ab followed by one or more 'c'
abc{2}	matches a string that has ab followed by 2 'c'
abc{2,5}	matches a string that has ab followed by 2 up to 5 'c'
a(b c)	matches a string that has a followed by b or c
\d	matches a single character that is a digit
.	matches any character
?	matches zero or one repetition
*	matches zero or more repetitions

Find text patterns in files. grep

For instance, the following grep plus regex finds lines that have an 'o' in the second position

```
grep -P '^ .o' 00_materials/turra.txt
```

- Are there any lines that end with the string 'Estado.' in the file turra.txt? (notice the capitalization and the end period)

Laws in Spain tend to be something like "Ley 6/2001" or "Ley 12/2007". Can you print the lines in turra.txt that refer to any Spanish law in this format without using the word "Ley"?



Manipulate text. awk

The awk command was named using the initials of the three people who wrote the original version in 1977: Alfred Aho, Peter Weinberger, and Brian Kernighan. It's a full scripting language, as well as a complete text manipulation toolkit for the command line.

The basic function of awk is to search files for lines (or other units of text) that contain certain patterns. When a line matches one of the patterns, awk performs specified actions on that line. awk continues to process input lines in this way until it reaches the end of the input files.

```
pattern {action}
```

Once you are familiar with awk, you will often type in simple one-shot throwaway programs the moment you want to use them. Then you can write the program as the first argument of the awk command, like this:

```
$ awk [options] 'program' <input-file-1>
```

Manipulate text. awk

A typical pattern for awk is looking for a text string or regular expression like this:

```
$ awk '/tRNA/' 00_materials/MP2819.gff
```

notice that the program (between ' ') only has a pattern, but no action. The default behaviour is to print the matching line (just like **grep**). The following awk program is equivalent:

```
$ awk '/tRNA/ {print}' 00_materials/MP2819.gff
```

Awk has special built-in variables that allows us to 'upgrade' its behaviour:

- **\$0** means the whole line.
- **\$1** means the first field (assuming one or more blank spaces as field separators).
- **\$n** means the nth field.

The following awk program will find the 'tRNA' pattern and print the 3rd field of the line.

```
$ awk '/tRNA/ {print $3}' 00_materials/MP2819.gff
```

Manipulate text. awk

You can skip the 'pattern' part of an **awk** program and use it as you used **cut**.



```
$ awk '{print $1}' 00_materials/MP2819.gff  
$ awk '{print $1"_"$2}' 00_materials/MP2819.gff
```



```
$ awk '{print "Gene Id \"$1\" is a \"$3\"'} 00_materials/MP2819.gff
```

Awk can take the following options (among others):

- **-F 'fs'** To specify a file separator.
- **-f file** To specify a file that contains awk script.
- **-v var=value** To declare a variable.

Compare the output of the following programs:



```
$ awk '{print $1"\t"$9}' 00_materials/MP2819.gff  
$ awk -F '\t' '{print $1"\t"$9}' 00_materials/MP2819.gff
```

Manipulate text. awk

Some other awk examples:

NR is another special variable in awk (Number of Record -or lines). Print the line number 100 and lines between 100 and 110.

```
$ awk 'NR==100' 00_materials/MP2819.gff  
$ awk 'NR==100, NR==110' 00_materials/MP2819.gff
```

Print lines using a logical condition. i. e. print lines with values in column 4 larger than 3000

```
$ awk -F '\t' '{if ($4 > 3000) print $0}' 00_materials/MP28199.gff
```

→ Look for lines with pattern 'tRNA' and print column 3 in file MP2819.gff

Find and replace. **sed**

sed is a stream editor. A stream editor is used to perform basic text transformations on an input stream (a file or input from a pipeline).

sed is useful to do "search and replace". The general form of searching and replacing text using **sed** takes the following form:

```
$ sed 's/SEARCH_REGEX/REPLACEMENT/g' INPUTFILE
```

- **s** - The substitute command, probably the most used command in sed.
- **/ / /** - Delimiter character. It can be any character but usually the slash (/) character is used.
- **SEARCH_REGEX** - string or regular expression to search for.
- **REPLACEMENT** - The replacement string.
- **g** - Global replacement flag. By default, sed reads the file line by line and changes only the first occurrence of the **SEARCH_REGEX** on a line. When the replacement flag is provided, all occurrences will be replaced.
- **INPUTFILE** - The name of the file on which you want to run the command.
- **-i** - By default sed writes its output to the standard output. This option tells sed to edit files in place. If an extension is supplied (ex -i.bak) a backup of the original file will be created.

Use **sed** to replace all instances of 'funcionario' to 'unicornio' in file **turra.txt** and redirect the output to file **turricornio.txt**. Use grep to print the lines with 'unicornio' in the new file

Pipes

This is the power of UNIX: small tools that do just one thing very well and can be put together to create complex tasks. Almost all of the standard Unix tools can work this way: unless told to do otherwise, they read from **standard input**, do something with what they've read, and write to **standard output**.

You can connect two commands together so that the output from one program becomes the input of the next program. Two or more commands connected in this way form a **pipe**.

To make a pipe, put a vertical bar (|) on the command line between two commands. Pipes can be read as "**then**".

This **one-liner** reads: with awk, using tabulator as a field separator, print the whole line if field number 4 is larger than 1000 and **THEN** do a word count to print the number of lines generated by the previous command.

```
$ awk -F '\t' '{if ($4 > 1000) print $0}' 00_materials/MP28199.gff | wc -l
```

What is this one-liner doing?

```
$ awk -F '\t' '/CDS/' 00_materials/MP28199.gff | head
```

Crazy one-liners

With all the commands we have seen and the pipes we can build complex streams of textual data processing. With some practice, the shell may replace several programs and speed up your workflow.

DNA sequence files can be reformatted with no more than unix tools. Transform fastq files to fasta files with a one-liner? no problem:

```
$ cat lib1Illumina_1.fastq | paste - - - - | cut -f1,2 | sed 's/^@/>/' | sed 's/\t/\n/' | head
```

→ Repeat the find and replace example with **sed** (replace 'funcionario' with 'unicornio' in **turra.txt**), but instead of dumping the result into a new file, use the pipe and **THEN** directly **grep** the lines that match the 'unicornio' pattern, and **THEN** use **wc** to count the number of words.

Variables in bash

Variables are named symbols that represent either a string or numeric value. When you use them in commands and expressions, they are treated as if you had typed the value they hold instead of the name of the variable.

To create a variable, you just provide a name and value for it. Your variable names should be descriptive and remind you of the value they hold. A variable name cannot start with a number, nor can it contain spaces. It can, however, start with an underscore. Apart from that, you can use any mix of upper- and lowercase alphanumeric characters.

- Let's create some variables. The format is to type the name, the equals sign =, and the value. Note there isn't a space before or after the equals sign. Giving a variable a value is often referred to as assigning a value to the variable.

```
$ x=1  
$ myName=your-name-here  
$ myAge=your-age-here
```

Variables in bash

To retrieve the value of a variable we need to use the command **echo** and precede the name of the variable with '**\$**' (notice that the **\$** is not present when we assign the variable).

→ Try to retrieve the values of the variables you have created with **echo**

It is a good practice to write the variable name between **{}**. i. e. **`${my_variable}`**

You can use a variable in more or less complex **echo** statements:

```
$ echo "My name is ${myName} and I am ${myAge} years old"
```

You can assign a variable with the result of evaluating an expression like this:

```
$ myVar=$(some expression)
```

→ Create a variable **pokemonNumber** that has the number of unique pokemon names in file **pokemon.txt**.

Scripting

Scripting allows for an automatic commands execution that would otherwise be executed interactively one-by-one.

The whole purpose of our first script is nothing else but print "Hello World" using echo command to the terminal output.

- Use your text editor of choice to create a new file named **hello-world.sh** containing the code below:

```
echo 'Hello world!'
```

and then run the script by doing:

```
$ bash hello-world.sh
```

Scripting

- You can specify the command language interpreter to use (in this case bash) and give execution permissions to make a "program-looking" program:

```
#!/bin/bash  
echo 'Hello world!'
```

```
$ chmod +x hello-world.sh
```

and then run the script by doing:

```
$ ./hello-world.sh
```

The **#!** (shebang) character sequence is both an interpreter directive (tells the shell which command interpreter should be used for the script), and a comment ignored by the command interpreter.

Scripting. Comments

You can write comments in your scripts. Short notes for the person who reads the script, explanations of what it does, etc. Is a good practice to do that so you don't need to understand a line of code or remember all the options of a program. Comments are preceded by a **#** symbol.

- Edit the hello-world.txt file with a comment explaining what it does and execute it again (first remove the output files generated previously).

Where are the programs?

Marbits uses a module system:

- Loads programs dynamically
- Coexistence of multiple versions of the same program

Use:

```
$ module avail
```

to get a list of all available modules

```
$ module help <module_name>
```

to get a brief help on a particular module

```
$ module spider <string>
```

to list all modules that match that string

Where are the programs?

```
$ module load <module_name>
```

to load particular module

module load generally adds the path to the executable programs to your **\$PATH** variable. This particular variable stores all the paths where the shell is going to look for executable files.

- Explore the contents of your \$PATH variable

```
$ module unload <module_name>
```

to remove a loaded module from your session

```
$ module purge
```

to remove all loaded module from your session

Where are the programs?

Some modules need the activation of a virtual environment or a conda environment. Check the module help to see particular indications for each module.

Conda is an open source package management system and environment management system that runs on Windows, macOS, and Linux. Conda quickly installs, runs and updates packages and their dependencies. Conda easily creates, saves, loads and switches between environments on your local computer. It was created for Python programs, but it can package and distribute software for any language.

Where are my things?

You have several paths in marbits to store your data and analyses, paths to share files and paths to do your daily work:

- **/home/your-user-name** This directory is your personal directory. It is very small and you should use it to store config files and small scripts or links to other locations.
- **/mnt/lustre/bio/users/your-user-name** is your personal directory in the lustre file system. Use it to store all the things that are important for you (raw data, analyses results...). Ask the admins to back up this directory.
- **/mnt/lustre/scratch** Here you can create a personal directory. This should be the place where you work on a daily basis. This directory doesn't have back-up. When you end an analysis, move the results to /mnt/lustre/bio/your-user-name
- **/mnt/lustre/repos/bio** Here you'll find data from several projects and public and custom databases. If your project has generated large quantity of data, ask for your data to be stored here. This directory has a back-up.
- **/mnt/lustre/bio/shared** Here you can create a directory with the adequate permissions to collaborate with other users.
- **/mnt/biostore** This is a cold storage unit, connected by NFS. It's slow, so it is used only to store low access data. In the future this will also have a back-up

Summarizing

We have seen:

- a brief introduction to bioinformatics in HPC environments @ICM
- How to log in to marbits
- How to get help
- How to navigate the file system
- How to copy, move, rename and delete files and directories
- How to examine and edit text files in a remote server
- How to sort and manipulate text files
- How to find text patterns and regular expressions in files
- How to manipulate text files with awk
- How to find and replace text patterns with sed
- How to pipe different commands in more or less complex workflows
- How to declare variables in bash
- How to write simple scripts
- How to load programs

But, have we done any bioinformatics yet? (well... implicitly, yes)

BUT...



DO NOT RUN JOBS IN THE MASTER NODE

Use the SLURM queue manager to allocate your
jobs in compute nodes

What is SLURM?



Marbits uses SLURM as a job scheduler. From its website: «As a cluster workload manager, Slurm has three key functions. First, it allocates exclusive and/or non-exclusive access to resources (compute nodes) to users for some duration of time so they can perform work. Second, it provides a framework for starting, executing, and monitoring work (normally a parallel job) on the set of allocated nodes. Finally, it arbitrates contention for resources by managing a queue of pending work.»

In order to submit a job you need to tell **slurm** what resources you will need so as it can allocate them and make sure that all users can fairly share them.

Submitting jobs using slurm

The easiest way of doing that is embedding your program calls within a script that can be understood by slurm and then submitting it with the command **sbatch**. This means foreseeing what modules you need to load, what variables you need to define, what programs, options and arguments you are going to call and write them into a script.

```
$ sbatch <your-script-name>
```

Alternatively, you can ask slurm to allocate an interactive session in a compute node with **srun**, that you can use to test all the things above. This interactive session will behave as any regular job and will be queued. This means that you may not get your interactive session right away, depending on the cluster occupation at the moment.

```
$ srun --pty /bin/bash
```

Option **--pty** asks for a pseudoterminal that uses **bash** as a command interpreter (we have to specify the path to bash as an argument to **srun**).

Your first bioinformatics job

We are going to do our first bioinformatics job. We are going to try it first in an interactive session.

```
$ srun --pty /bin/bash
```

This will give you one slot and as much memory as the system has bound to a core. You can tune your srun job to ask for more memory or more cores or nodes as if it was a regular job (we'll see it later)

Notice how your prompt has changed. Try **hostname** and **whoami** and check who you are and where are you...

Your first bioinformatics job

Our first bioinformatics problem is going to be a very simple one. We have a large-ish fasta file at **00_materials/fasta/mySequences.fasta**, and we want to split it in smaller files without losing any information.

A fasta file is a text file with DNA/RNA or amino-acid sequences. Each sequence has a header, preceded by a '>' symbol:

```
>NC_002695.2 Escherichia coli 0157:H7 str. Sakai DNA, complete genome
AGCTTTCAATTCTGACTGCAACGGCAATATGTCTCTGTGGATTAAAAAAAGAGTCTCTGACAGCAGC
TTCTGAACGGTTACCTGCCGTGAGTAAATTAAAATTATTGACTTAGGTCACTAAATACTTAACCAA
TATAGGCATAGCGCACAGACAGATAAAAATTACAGAGTACACAACATCCATGAAACGCATTAGCACCACC
>NC_002128.1 Escherichia coli 0157:H7 str. Sakai plasmid p0157, complete sequence
AGCCAGATTTCACCCGCCATCCTAAAGAAGGGGATAGTCAACCACATCTGACCAGCCTGCGGAAAAGTC
TGCTGCTTGTCCGTCCGGTGAAAGCTGATGATAAACACCTGTTAGGTGGAAGCCCGCATGATAATAA
TAAAATTCTCGGTACGTTAACCCATTATCCTCCTCATCACTACCGGATACAATCTACCATCTGGATGGT
```

- take a look at the contents of the file. How many lines does this fasta file have? How many sequences?

Your first bioinformatics job

Load the module **marbits-utils**. There we have a small program called **fasta-splitter.pl**

You can call the program by writing **fasta-splitter.pl**. If you don't specify any option or argument, it will supply a brief help information.

Read the help page and try to understand all the options of the program.

- 
- Now split the fasta file in 5 equal smaller files, wrapping sequences at 40 bp (line length) and adding a prefix '**-chunk-**' before the part number.

Your first bioinformatics job

→ Now that you have done it interactively, write a script that does the same, but with prefix '**parallel**' in the output file names. Remember to add the first line '#! /bin/bash'. It is not needed now, but will be in future steps. Execute the script with **bash** and see what happens.

You have successfully run a bioinformatics program from the command line, using only textual orders. This program could be used to divide a large job into many small jobs. With the adequate scripting, you could send as many simultaneous jobs to the SLURM queue, effectively reducing the total computing time in a **parallel job** (well, actually in what is called and embarrassingly parallel job https://en.wikipedia.org/wiki/Embarrassingly_parallel) which is what HPC clusters are good for.

Submitting a job to the SLURM queue

- Exit the interactive session with any of these

```
$ exit  
$ logout  
^d
```

And check again the prompt, **whoami** and **hostname**.

You are back in the master node. Now that you have a script with a valid bioinformatics program, you could run it non-interactively by sending it to the SLURM queue. Remember that the syntax was to use the command **sbatch** followed by slurm options and the script name as argument.

Submitting a job to the SLURM queue

You can call **sbatch** from the command line and specify the options inline. However, that is only recommended for experienced users. The best way of submitting a job is by writing a script that embeds all slurm options before the actual work that the script does. We call it a slurm header. You can then re-use them or use them to write comments about what that script is supposed to do, what programs and versions it uses, etc.

Slurm scripts needs a shebang indicating the command interpreter, generally

```
#! /bin/bash
```

All the script options are preceded by **#SBATCH**. Remember that a **#** works as a comment, so your shell is not going to interpret the option, although slurm will.

SLURM options

```
#! /bin/sh

#SBATCH --account=emm1
#SBATCH --job-name=myFirstJob
#SBATCH --time=01-12:00:00
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=12G
#SBATCH --output=path_to_my_output_directory/jobLog_%J.out
#SBATCH --error=path_to_my_output_directory/jobLog_%J.err
#SBATCH --mail-type=ALL
#SBATCH --mail-user=pepita@icm.csic.es
```

SLURM options

```
#!/bin/sh

#SBATCH --account=emm1
#SBATCH --job-name=myFirstJob
#SBATCH --time=01-12:00:00
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=12G
#SBATCH --output=path_to_my_output_directory/jobLog_%J.out
#SBATCH --error=path_to_my_output_directory/jobLog_%J.err
#SBATCH --mail-type=ALL
#SBATCH --mail-user=pepita@icm.csic.es
```

--account refers to the «bank» or project account you want your job to be accounted for. One user may have more than one of such accounts, so you may have to choose which one is going to be billed by your job. Also, the same bank/project account is going to be shared between the users that belong to the same group/project. Hence, don't mistake your «user» account (that'd be your user name that you use to log in to marbits) with the bank/project account, that will be used for billing purposes.

SLURM options

```
#! /bin/sh

#SBATCH --account=emm1
#SBATCH --job-name=myFirstJob
#SBATCH --time=01-12:00:00
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=12G
#SBATCH --output=path_to_my_output_directory/jobLog_%J.out
#SBATCH --error=path_to_my_output_directory/jobLog_%J.err
#SBATCH --mail-type=ALL
#SBATCH --mail-user=pepita@icm.csic.es
```

--job-name. The name you want your job to have. Defaults to the script name

SLURM options

```
#!/bin/sh

#SBATCH --account=emm1
#SBATCH --job-name=myFirstJob
#SBATCH --time=01-12:00:00
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=12G
#SBATCH --output=path_to_my_output_directory/jobLog_%J.out
#SBATCH --error=path_to_my_output_directory/jobLog_%J.err
#SBATCH --mail-type=ALL
#SBATCH --mail-user=pepita@icm.csic.es
```

--time. Time that your job will need to complete. If your job runs longer than the specified time it will be killed, but when you declare this option, the job scheduler is able to better optimize the available resources. Format is DD-HH:MM:SS. Currently marbits does not have a time limit and this option is not compulsory, BUT many HPC systems have time limitations and jobs won't run without this option. Nevertheless, it is a good practice to specify a time of execution to make the queue run smoother.

SLURM options

```
#! /bin/sh

#SBATCH --account=emm1
#SBATCH --job-name=myFirstJob
#SBATCH --time=01-12:00:00
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=12G
#SBATCH --output=path_to_my_output_directory/jobLog_%J.out
#SBATCH --error=path_to_my_output_directory/jobLog_%J.err
#SBATCH --mail-type=ALL
#SBATCH --mail-user=pepita@icm.csic.es
```

--ntasks. it's the number of tasks you are going to do. It is generally one, and defaults to one. You only need to change this if you are going to run MPI jobs (you probably won't in bioinformatics)

SLURM options

```
#! /bin/sh

#SBATCH --account=emm1
#SBATCH --job-name=myFirstJob
#SBATCH --time=01-12:00:00
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=12G
#SBATCH --output=path_to_my_output_directory/jobLog_%J.out
#SBATCH --error=path_to_my_output_directory/jobLog_%J.err
#SBATCH --mail-type=ALL
#SBATCH --mail-user=pepita@icm.csic.es
```

--cpus-per-task. Number of computing cores that your job is going to use. It defaults to 2. If your program or script can use more than one core, it is very important that the number of cores/threads that you are asking with --cpus-per-task matches the number of cores that you will ask your program to use. Failure to do that will make SLURM believe that it has a different number of cores available than what it really has, degrading its performance.

SLURM options

```
#! /bin/sh

#SBATCH --account=emm1
#SBATCH --job-name=myFirstJob
#SBATCH --time=01-12:00:00
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=12G
#SBATCH --output=path_to_my_output_directory/jobLog_%J.out
#SBATCH --error=path_to_my_output_directory/jobLog_%J.err
#SBATCH --mail-type=ALL
#SBATCH --mail-user=pepita@icm.csic.es
```

--mem. Amount of memory that your job will need per node. Units are MB, but can also be specified with suffixes as 12G instead 12000. You need to benchmark your jobs before using this parameter confidently (see how to benchmark your jobs).

SLURM options

```
#!/bin/sh

#SBATCH --account=emm1
#SBATCH --job-name=myFirstJob
#SBATCH --time=01-12:00:00
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=12G
#SBATCH --output=path_to_my_output_directory/jobLog_%J.out
#SBATCH --error=path_to_my_output_directory/jobLog_%J.err
#SBATCH --mail-type=ALL
#SBATCH --mail-user=pepita@icm.csic.es
```

Having a job running in a remote server in a non-interactive fashion makes you miss any output or error message that would be printed in the screen (standard output and standard error). The following options redirect those streams to files so you can review them later.

--output. Specifies a path for the job standard output file. The variable **%J** adds the job number to the output file name. Variable **%A** has the same effect when your job is an array of jobs. Variable **%a** adds the array index to the file name. i.e: **log%A%a.out**.

--error. Idem for standard error.

SLURM options

```
#!/bin/sh

#SBATCH --account=emm1
#SBATCH --job-name=myFirstJob
#SBATCH --time=01-12:00:00
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=12G
#SBATCH --output=path_to_my_output_directory/jobLog_%J.out
#SBATCH --error=path_to_my_output_directory/jobLog_%J.err
#SBATCH --mail-type=ALL
#SBATCH --mail-user=pepita@icm.csic.es
```

--mail-type. Send an email to the submitter in the following cases (pick one): BEGIN, END, FAIL, ALL (self explanatory, I think). There are more options, see **sbatch** man page. Currently, this option needs to be specified alongside **--mail-user**.

--mail-user. E-mail address of recipient of the **--mail-type** option. Only **@icm.csic.es** mail addresses are allowed. Anyway if you work a lot in marbits, the mail options get annoying very quickly and you'll end up removing them.

Single-core (batch/sequential) job

```
#!/bin/sh

# SLURM HEADER BEGINS HERE
#####
#SBATCH --account=
#SBATCH --job-name=
#SBATCH --time=
#SBATCH --ntasks=
#SBATCH --cpus-per-task=
#SBATCH --mem=
#SBATCH --output=
#SBATCH --error=
#SBATCH --mail-type=
#SBATCH --mail-user=
#####
# SLURM HEADER ENDS HERE

# write your job here
```

→ Here you have the bare skeleton of a slurm script. Complete it with the fasta-splitter script you did before and submit it to the queue with **sbatch**

Check the status of your jobs: **squeue**

- Edit your slurm script and add the following line AFTER the last command and resubmit with **sbatch**:

```
sleep 120
```

squeue shows the status of the queue, including all jobs from all users. It can be customized to show more or less information:

```
squeue -o "%10A %12j %5K %10u %8a %10P %3t %10M %6D %6C %10f %R" -u "$(whoami)"
```

A = job Id
j = job or step name
K = job array index
u = user name
a = account associated
P = Partition
t = job state in compact form
M = time used by the job
D = number of nodes
C = number of CPUs
f = features required by the job
R = Reason for waiting to execute

In marbits, the alias **sq** gives a comprehensive list of information on the jobs YOU are currently running. Alias **sqa** gives the same information of ALL the jobs currently running in the cluster

Cancel a submitted job: **scancel**

scancel allows you to kill one or more of your jobs. You can specify the job number or its name (with **-n**), or all your jobs (with **-u <your_user_name>**).

```
scancel <jobNumber>
```

You may use the output of **squeue/sq** and parse it to invent a rule that allows you to kill a large number of jobs (using common sense!).

- Re-submit the last version of your slurm job, check that it is queued or running with **squeue/sq**, note the job number and kill it with **scancel**.

Multi-core (parallel) job

```
#!/bin/sh

# SLURM HEADER BEGINS HERE
#####
#SBATCH --account=
#SBATCH --job-name=
#SBATCH --time=
#SBATCH --ntasks=8
#SBATCH --cpus-per-task=
#SBATCH --mem=
#SBATCH --output=
#SBATCH --error=
#SBATCH --mail-type=
#SBATCH --mail-user=
#####
# SLURM HEADER ENDS HERE

# write your job here
```

This is the skeleton of a slurm script for a multi-core or parallel job.

Multi-core (parallel) job

mafft is a program that aligns nucleotide or amino-acid sequences. These alignments can be then used to build a phylogenetic tree. Alignments can be costly computationally, and some aligners (i.e . **mafft**) can distribute the workload among more than one core to run faster.

mafft is a bit more complex than **fasta-splitter**. Load the **mafft** module and type

```
$ mafft -h
```

to take a look at the help page. As you can see, option **--thread** allows you to specify the number of threads/cores to use in the alignment. We are going to add **--thread 8** in the mafft command to match with the slurm header that we have written before.



```
# SLURM HEADER ENDS HERE
#####
module load mafft
mafft --maxiterate 1000 --globalpair --thread 8 00_materials/merA.faa >
00_materials/merA-aligned.faa
```

Multi-core (parallel) job

The **merA.faa** file is a multi-fasta file with sequences belonging to the *merA* gene, found in sediment metagenomes and metagenome-assembled genomes. The *merA* gene is involved in the methyl-mercury demethylation.

- Take a look at the file and check how many sequences are there.
- Submit the multi-core job with sbatch.
- Edit the script and submit it with a single core. Remember to edit both the header and the mafft command to have matching number of cores.

Accounting jobs

When jobs have finished you can use the command **sacct** to see the consumption of resources of your last jobs (CPU time, maximum memory usage...).

The **sacct** command displays job accounting data stored in the job accounting log file or slurm database in a variety of forms for your analysis. The **sacct** command displays information on jobs, job steps, status, and exit codes by default. You can tailor the output with the use of the **--format= option** to specify the fields to be shown.



```
$ sacct -j <jobNumber>  
  
$ sacct --format=jobid, jobname, submit, start, end, elapsed, ncpus, ntasks,  
MaxVMSize, AllocNodes, state
```

In marbits the alias **sc** gives you a comprehensive list of accounting info of your jobs



Check your last jobs with **sc**. Is there any difference in elapsed time from the single-core and multi-core **mafft** jobs?

Job arrays

Job arrays are an easy way of submitting many similar jobs when you have several input files that have to be processed with the same workflow and they are independent one from each other. If you can do your job using a loop, you definitely want to do a job array instead.

It takes advantage of variable **SLURM_ARRAY_TASK_ID**, that takes the values specified to the option **#SBATCH --array**.

The slurm option --array accepts a list of numbers separated with ',' or a numeric range separated with '-' or a mixture of both. All the following definitions are equivalent, taking all numbers from 1 to 10:

```
#SBATCH --array=1-10
#SBATCH --array=1,2,3,4,5,6,7,8,9,10
#SBATCH --array=1-5,6,7-10
```

Job arrays

The typical case would involve several input files with a numerical index somewhere in their file names:

```
infile1.fasta  
infile2.fasta  
infile3.fasta
```

we could submit a single slurm script that spawns 3 different jobs, one for each input file using variables that take advantage of the file numeration. i. e.

```
#!/bin/sh

#SBATCH --account=test
#SBATCH --job-name=simpleArray
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=1G
#SBATCH --output=path_to_my_output_directory/jobLog_%A_%a.out
#SBATCH --error=path_to_my_output_directory/jobLog_%A_%a.err
#SBATCH --array=1-3%1

# This simple job locates the `>` character at the beginning of a
# line in file infile.#.fasta, where # is the value of the ${SLURM_ARRAY_TASK_ID}
# variable, that is taken from the range given to `--array`,
# and counts how many of them there are, dumping the result to infile.#.count.txt

grep -c '^>' infile${SLURM_ARRAY_TASK_ID}.fasta > infile${SLURM_ARRAY_TASK_ID}.count.txt
```

Job arrays

You can run job arrays in which each task uses more than one cpu. You can run the example above with option **--cpus-per-task=4** and each task (each element of the array) will use 4 cpus. If you don't specify **%n** in **--array=1-3** and there are enough free resources, all three tasks will run at the same time, using a total of $3 \times 4 = 12$ cpus. Notice that the value of cpus is the one for each element of the array!

You are encouraged to be respectful with other users and specify a sensible number of simultaneous jobs in your job array with **%n**, where n is a number that won't clog the cluster. In the previous example, only one job will be run at the same time.

Job arrays

Notice that in the **--output** and **--error** options we have substituted **%J** to **%A** plus **%a**. The first option is equivalent to **%J**, as it takes the value of the job ID. The second takes the value of the array index, so you can identify each log file with first, a job submission, and second with the input file that has been processed.

If you are going to submit large arrays with lots of tasks, please use the option **--exclusive=user**. Otherwise, all tasks of your job will spread out throughout the cluster. Many users like to have empty nodes for their jobs and this would interfere with them.

Job arrays

You should have 5 fasta files from a previous job.

- Let's process each single file in a single-core job each. For this we'll use an array job that makes use of the variable **\$SLURM_ARRAY_TASK_ID** to process the files individually.

Try to extract ONLY the header of each sequence of each fasta file and write it in an output file FOR each one. You need to end up with 5 files in your work directory that have that information.

NOTES: make the job concurrency = 2, and add a 30 seconds sleep command to allow us to see how the job progresses.

Modify jobs when they are running

Sometimes you may need to change options of your jobs once they have started. You can achieve it using the command **scontrol update**. Many of the commands issued with **scontrol** are reserved to the administrator, but some others can be issued to jobs you are the owner of.

If you want to throttle up or down the number of simultaneous tasks that can be execute in a job array (remember, the number behind % in **--array=1-5%2** do

```
$ scontrol update JobId=<JobNumber> ArrayTaskThrottle=5  
$ scontrol update JobName=<JobName> ArrayTaskThrottle=5
```

and it will go from executing 2 simultaneous tasks to run 5 (all) of them.

Generalization of job arrays

Most of the times your input files will not be so neatly named with numeric suffixes.

You can create a text file with the paths or names of your input files. Every line of that file could be potentially called as line 1, line 2 and so on. We can take advantage of this using some tools you have already seen (namely **awk** and assigning a variable with the result of the evaluation of an expression):

```
myInputFile=$(awk "NR==$SLURM_ARRAY_TASK_ID" your-file-list.txt)
```

This **awk** command shows the contents of the **NR**th line of a file. When the variable holding the array element number is 1, **\${myInputFile}** takes the value of the contents of line 1 in **your-file-list.txt**. The second line for array element 2 and so on. This way your input files can have arbitrary names.

Generalization of job arrays

- Try to make a job array that reads the name of the split fasta files from a file list (instead of using the numerical suffix of the previous job).

Then try to count the number of DNA nucleotides with UNIX tools only! Dump the result in a single output file for each input file.

HINTS:

- The sequence headers have to be omitted
- The `\n` character COUNTS as a character and you don't need to count it. The command `tr` 'translates' or deletes characters like this respectively: `tr "\n" "something"` or `tr -d "\n"`