What is UNIX?

- Operating system
- Powerful
- Multi-user
- Multitasking
Why is it important for bioinformatics?

- Efficiency and speed
  - Handling large datasets and running analyses efficiently
  - Using scripts to automate repetitive tasks

- Access to powerful tools and applications
- Facilitates sharing and reproducing analyses
The terminal

Make it comfortable to work in

- Resize the window
- Change the font size
- Open multiple terminal windows (or tabs)
- Make sure you have the right combination of colours that work for you.
File system organization
Paths - Absolute vs Relative

Absolute paths:
- `/home/Merce`
- `/home/Merce/Genomics`
- `/home/Merce/bin`

. refers to our current location

.. refers to the location above us
Paths - Absolute vs Relative

Absolute paths:
1. `/home/Merce`
2. `/home/Merce/Genomics`
3. `/home/Merce/bin`

Relative paths:
1. `./`
2. `./Genomics`

- `./` refers to our current location
- `../` refers to the location above us
Paths - Absolute vs Relative

Absolute paths:
- /home/Merce
- /home/Merce/Genomics
- /home/Merce/bin

Relative paths:
- . refers to our current location
- .. refers to the location above us
- ./bin
**Paths - Absolute vs Relative**

You are here!

Absolute:

- `/home/Merce/Genomics`
- `/home/Merce/bin`

Relative:

- `./`
- `../`

. refers to our current location

.. refers to the location above us
Paths - Absolute vs Relative

Absolute paths
- /home/Merce
- /home/Merce/Genomics
- /home/Merce/bin

Relative paths
- ..
- .
- ../bin

. refers to our current location

.. refers to the location above us
Absolute paths

- `/home/Merce`
- `/home/Merce/Genomics`
- `/home/Merce/bin`

Relative paths

- `.` refers to our current location
- `..` refers to the location above us

How do we refer to this directory?
Paths - Absolute vs Relative

Absolute paths
- /home/Merce
- /home/Merce/Genomics
- /home/Merce/bin

Relative paths
- . refers to our current location
- .. refers to the location above us
- ../../User2
File system navigation

`pwd` - where am I?

`cd` - change directory
File system navigation

**pwd** - where am I?

/home/Merce

**cd** - change directory

> cd /home/Merce/Genomics

> cd ./Genomics
File system navigation

**pwd** - where am I?

```
/home/Merce/Genomics
```

**cd** - change directory

```
> cd /home/Merce
> cd ../
```
**File system visualization**

**ls** - shows you the contents the directory you are in

```
> ls
> ls .
> ls ./
> ls ../
```
Key shortcuts

Ctrl + C          halts current command
Ctrl + Shift + C  copy (linux) -  Cmd + C (mac)
Ctrl + Shift + V  paste (linux) -  Cmd + V (mac)
Ctrl + W          erases one word in current line
Ctrl + U          erases whole line
Ctrl + A          go to beginning of line
Ctrl + E          go to end of line
Type exit         log out of current session
Create, copy, move, and remove files and folders

**mkdir** - create new directory

**cp** - copy file

**mv** - move file or directory

**rm** - remove file

"Unix was not designed to stop its users from doing stupid things, as that would also stop them from doing clever things." - Doug Gwyn
Symbolic links

`ln -s /path/to/file link` create a symlink of file

```bash
> ln -s /home/Merce/Genomics/Data/seq.fastq /home/Merce/Genomics/Analyses/Analysis1/
```

If we are already inside the folder Analysis1:

```bash
> ln -s /home/Merce/Genomics/Data/seq.fastq .
```
**Manual**

`man command` - manual of the command

```
> man ls
```

- `ls -l` formatted list
- `ls -h` “human” formatted list
- `ls -lh` combination of flags
Inputs and outputs

**stdin** It stands for standard input, and is used for taking text as an input.

**stdout** It stands for standard output, and is used to text output of any command you type in the terminal, and then that output is stored in the stdout stream.

**stderr** It stands for standard error. It is invoked whenever a command faces an error, then that error message gets stored in this data stream.
stdin, stdout, stderr

command `stdin`  
- if it works: prints in our terminal the `stdout`
- if it fails: prints in our terminal the `stderr`

command `stdin > stdout`  
- if it works: `stdout` is redirected to a file
- if it fails: prints in our terminal the `stderr`

command2 `stdin2 > stdout`  
- `stdout` is redirected to a file and rewrites its contents

command2 `stdin2 >> stdout`  
- `stdout` is redirected to a file and appended after its contents
stdin, stdout, stderr

command stdin
if it works: prints in our terminal the stdout
if it fails: prints in our terminal the stderr

command file1 > output.txt
  stdin  stdout
if it works: stdout is redirected to a file
if it fails: prints in our terminal the stderr

command2 file2 > output.txt
  stdin  stdout
stdout is redirected to a file and rewrites its contents

command2 file2 >> output.txt
  stdin  stdout
stdout is redirected to a file and appended after its contents
stdin, stdout, stderr

- `command file1 2> errors.txt` if it works: prints in our terminal the `stdout` if it fails: `stderr` is redirected to a file
- `command file1 &> output.txt` redirects both `stdout` and `stderr` to a file
- `command file1 > output.txt 2> errors.txt` redirects both `stdout` and `stderr` to a separate file each.
Explore file content

**wc** - word count (-l lines, -c characters, -w words)

**less** - visualize file contents in your terminal screen (press q to exit)

**cat** - prints contents of your file as *standard output* in your terminal

**head** - visualize the first 10 lines of a file

**tail** - visualize the last 10 lines of a file
The character | (pipe) is used to concatenate commands, so that we can run one command after the other, avoiding the creation of intermediate files.

```
command1 input | command2 > output
```

Instead of:

```
command1 input > output1
command2 output1 > output2
```

Using pipe, the output of running `command1` on a given input gets directly piped into `command2`, and we obtain an output of these two consecutive commands, generating only one output.
A bit more advanced file-handling commands

**cat** - prints contents of your file as *standard output* in your terminal

**redirect to a command**

- `cat fileA | command > output.txt`

**concatenate files**

- `cat fileA fileB >> fileC`

- `cat fileA > fileC`

- `cat fileB >> fileC`
A bit more advanced file-handling commands

**sort** - puts in certain order a series of lines in our file

- `sort -r fileA` sorts in reverse order
- `sort -n fileA` sorts lines in fileA numerically
- `sort -k 2 fileA` sort fileA by column 2
- `sort -k 2nr fileA` sort fileA by column 2, numerically and in reverse order
- `sort -V fileA` sort lines in fileA numerically natural.
- `sort -u fileA` sort lines and removes duplicates -> `sort fileA | uniq`
A bit more advanced file-handling commands

Are these two files different?

**diff** - can tell us if there are differences between two files

```
diff -q fileA fileB
```

“Files fileA and fileC differ”

```
diff fileA fileB
```

prints differences
A bit more advanced file-handling commands

Splitting a file

**split** - split a given file into multiple files (default 1000)

```
split -l 20 fileA
```

produce x number of files from fileA, each containing 20 lines.

**cut** - extract specific parts of a file

```
cut -c 2 fileA
```

extract specific columns from a file
Nano - The simpler option of text editor. All commands within the nano text editor are given by pressing the Control-key, usually represented as ^

^S  save current file
^O  save to (a different file)
^X  exit from nano
Text editors

Vim - a highly configurable text editor built to make creating and changing any kind of text very efficient

- i  start insert mode (you can start typing after where your cursor is)
- ESC exits insert mode (also Ctrl + C)
- :w  save file without exiting
- :q  exit file (if there are unsaved changes, it fails)
- :wq  save and exit
- :q!  exit without saving changes
**Text editors**

**emacs** - a text editor characterized by its extensibility and configurability. Some essential commands get activated by typing Control + X, then the command (while holding the control key), but there is a wide range of key combinations to be used to move and edit the text.

- Ctrl + x + s  save file
- Ctrl + x + c  exit editor (if not saved, it ask if you want to save, then type "yes")
VIM
usable in just about any environment.
does one thing, well.

EMACS
flexible, customizable, and packed with every feature known to man.

NANO
mostly used by people who do not know what they are doing; or psychopaths.
What my family and friends think I do

What my supervisor thinks I do

What I actually do
### Cheat-sheet

#### Workshop on Genomics 2024

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pwd</code></td>
<td>show current path / directory</td>
</tr>
<tr>
<td><code>ls</code></td>
<td>list directory</td>
</tr>
<tr>
<td><code>cd dir</code></td>
<td>change to directory <code>dir</code></td>
</tr>
<tr>
<td><code>cd</code></td>
<td>change to home</td>
</tr>
<tr>
<td><code>cd ~</code></td>
<td>change to home</td>
</tr>
<tr>
<td><code>cd -</code></td>
<td>change to previous working directory</td>
</tr>
<tr>
<td><code>.</code></td>
<td>current directory</td>
</tr>
<tr>
<td><code>..</code></td>
<td>parent directory</td>
</tr>
<tr>
<td><code>mkdir dir</code></td>
<td>create directory <code>dir</code></td>
</tr>
<tr>
<td><code>cp file1 file2</code></td>
<td>copy <code>file1</code> to <code>file2</code></td>
</tr>
<tr>
<td><code>mv file1 file2</code></td>
<td>move <code>file1</code> to <code>file2</code> or rename <code>file1</code> to <code>file2</code></td>
</tr>
<tr>
<td><code>rm file1</code></td>
<td>delete <code>file1</code></td>
</tr>
<tr>
<td><code>ln -s file link</code></td>
<td>create symbolic link</td>
</tr>
<tr>
<td><code>man command</code></td>
<td>manual for command</td>
</tr>
<tr>
<td><code>chmod +x file</code></td>
<td>makes <code>file</code> executable</td>
</tr>
<tr>
<td>`command1</td>
<td>command2`</td>
</tr>
<tr>
<td><code>wget web-address-to-file</code></td>
<td>download file into current dir</td>
</tr>
<tr>
<td><code>ssh user@server</code></td>
<td>connect to <code>user</code> on <code>server</code></td>
</tr>
<tr>
<td><code>tar -xf archive.tar.gz</code></td>
<td>uncompressed <code>tar.gz</code></td>
</tr>
<tr>
<td><code>tar -czf archive.tar.gz archive</code></td>
<td>compress <code>archive</code> to <code>archive.tar.gz</code></td>
</tr>
<tr>
<td><code>gzip compress.gz</code></td>
<td><code>gzip</code> <code>compress.gz</code></td>
</tr>
<tr>
<td><code>gunzip uncompressed.gz</code></td>
<td><code>gunzip</code> <code>uncompressed.gz</code></td>
</tr>
</tbody>
</table>

#### Unix Cheat-sheet

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl + C</td>
<td>halts current command</td>
</tr>
<tr>
<td>Tab</td>
<td>autocomplete current line</td>
</tr>
<tr>
<td>Arrow up</td>
<td>previous commands</td>
</tr>
<tr>
<td>Ctrl + Shift + C</td>
<td>copy (linux)</td>
</tr>
<tr>
<td></td>
<td>Cmd + C (mac)</td>
</tr>
<tr>
<td>Ctrl + Shift + V</td>
<td>paste (linux)</td>
</tr>
<tr>
<td></td>
<td>Cmd + V (mac)</td>
</tr>
<tr>
<td>Ctrl + W</td>
<td>erases one word in current line</td>
</tr>
<tr>
<td>Ctrl + U</td>
<td>erases whole line</td>
</tr>
<tr>
<td>Ctrl + A</td>
<td>go to beginning of line</td>
</tr>
<tr>
<td>Ctrl + E</td>
<td>go to end of line</td>
</tr>
<tr>
<td><code>exit</code></td>
<td>log out of current session</td>
</tr>
</tbody>
</table>
Rubber duck
Now it is your turn to practice!
And become each other’s ducks!