# Comandos básicos en la terminal:

* **pwd**: Nos muestra la ruta de carpetas en la que te encuentras ahora mismo.
* **mkdir**: Nos permite crear carpetas (por ejemplo, mkdir Carpeta-Importante).
* **touch**: Nos permite crear archivos (por ejemplo, touch archivo.txt).
* **rm**: Nos permite borrar un archivo o carpeta (por ejemplo, rm archivo.txt). Mucho cuidado con este comando, puedes borrar todo tu disco duro.
* **Rm -r:** Si quieres eliminar una carpeta (por ejemplo, rm -r mi carpeta).
* **cat**: Ver el contenido de un archivo (por ejemplo, cat nombre-archivo.txt).
* **ls**: Nos permite cambiar ver los archivos de la carpeta donde estamos ahora mismo. Podemos usar uno o más argumentos para ver más información sobre estos archivos (los argumentos pueden ser -- + el nombre del argumento o - + una sola letra o shortcut por cada argumento).
  + ls -a: Mostrar todos los archivos, incluso los ocultos.
  + ls -l: Ver todos los archivos como una lista.
* **cd**: Nos permite navegar entre carpetas.
  + cd /: Ir a la ruta principal:
  + cd o cd ~: Ir a la ruta de tu usuario
  + cd carpeta/subcarpeta: Navegar a una ruta dentro de la carpeta donde estamos ahora mismo.
  + cd .. (cd + dos puntos): Regresar una carpeta hacia atrás.
  + Si quieres referirte al directorio en el que te encuentras ahora mismo puedes usar cd . (cd + un punto).
* **history**: Ver los últimos comandos que ejecutamos y un número especial con el que podemos repetir su ejecución.
* **! + número**: Ejecutar algún comando con el número que nos muestra el comando history (por ejemplo, !72).
* **clear**: Para limpiar la terminal. También podemos usar los atajos de teclado Ctrl + L o Command + L.

# Comandos para mover archivos entre los estados de Git:

* **git status**: nos permite ver el estado de todos nuestros archivos y carpetas.
* **git add**: nos ayuda a mover archivos del Untracked o Unstaged al estado Staged. Podemos usar git nombre-del-archivo-o-carpeta para añadir archivos y carpetas individuales o git add -A para mover todos los archivos de nuestro proyecto (tanto Untrackeds como unstageds).
* **git reset HEAD**: nos ayuda a sacar archivos del estado Staged para devolverlos a su estado anterior. Si los archivos venían de Unstaged, vuelven allí. Y lo mismo se venían de Untracked.
* **git commit**: nos ayuda a mover archivos de Unstaged a Tracked. Esta es una ocasión especial, los archivos han sido guardado o actualizados en el repositorio. Git nos pedirá que dejemos un mensaje para recordar los cambios que hicimos y podemos usar el argumento -m para escribirlo (git commit -m "mensaje").
* **git rm**: este comando necesita alguno de los siguientes argumentos para poder ejecutarse correctamente:
  + git rm --cached: Mueve los archivos que le indiquemos al estado Untracked.
  + git rm --force: Elimina los archivos de Git y del disco duro. Git guarda el registro de la existencia de los archivos, por lo que podremos recuperarlos si es necesario (pero debemos usar comandos más avanzados).
* **git status**: Saber el estatus de los archivos en la carpeta en donde nos encontramos
* **git init**: Le indicaremos a Git que queremos crear un nuevo repositorio para utilizar su sistema de control de versiones. Solo debemos posicionarnos en la carpeta raíz de nuestro proyecto y ejecutar el comando **git init**.

# Configuraciones

Recuerda que Git está optimizado para trabajar en equipo, por lo tanto, debemos darle un poco de información sobre nosotros. No debemos hacerlo todas las veces que ejecutamos un comando, basta con ejecutar solo una sola vez los siguientes comandos con tu información:

git config --global user.email "tu@email.com"

git config --global user.name "Tu Nombre"

Existen muchas otras configuraciones de Git que puedes encontrar ejecutando el comando git config --list (o solo git config para ver una explicación más detallada).

# Comandos para moverse entre versiones

* **Git log**: ver el historial de cambios de responsable, commit y fecha
* **Git log –stat**: ver el historial de cambios de responsable, commit, fecha, archivos modificados
* **git show**: El comando **git show** nos muestra los cambios que han existido sobre un archivo y es muy útil para detectar cuándo se produjeron ciertos cambios, qué se rompió y cómo lo podemos solucionar. Pero podemos ser más detallados.
* **git checkout** + ID del commit nos permite viajar en el tiempo. Podemos volver a cualquier versión anterior de un archivo específico o incluso del proyecto entero. Esta también es la forma de crear ramas y movernos entre ellas.
* **git reset**: También hay una forma de hacerlo un poco más “ruda”: usando el comando **git reset.** En este caso, no solo “volvemos en el tiempo”, sino que borramos los cambios que hicimos después de este commit.
  + Hay dos formas de usar git reset: con el argumento **--hard**, borrando toda la información que tengamos en el área de staging (y perdiendo todo para siempre). O, un poco más seguro, con el argumento **--soft**, que mantiene allí los archivos del área de staging para que podamos aplicar nuestros últimos cambios, pero desde un commit anterior.

# Comandos para moverse entre ramas

* **git Branch**: Crea una rama con la estructura **git branch** *NombreDeLaRama*

La cabecera o HEAD representan la rama y el commit de esa rama donde estamos trabajando. Por defecto, esta cabecera aparecerá en el último commit de nuestra rama principal. Pero podemos cambiarlo al crear una rama **git branch** *NombreDeLaRama*

* + **git Branch:** Nos permite ver las ramas que se encuentran en el proyecto solo el **git Branch** sin el nombre de la rama
* **git checkout** *NombreDeLaRama*: Nos sirve para movernos en el tiempo a cualquier otro commit de cualquier otra rama con los comandos **git checkout** *NombreDeLaRama*
* (git reset id-commit, git checkout rama-o-id-commit).