Quiénes van a hacer el commit queda registrado quién va a hacerlo:

- **git config --global user.name**
- **git config --global user.email**

Ejecutamos el comando git config –global core.editor nano
En algunos momentos git va a necesitar un editor de texto.

CREAR UN REPOSITORIO:

git init –> inicializa el repositorio
Cuando estamos en una terminal, cuando ejecuto comandos se ejecuta SIEMPRE dentro de una carpeta.

pwd Me dice dónde estoy. Ejemplo: /home/kas

ls Me dice todas las carpetas que hay dentro de donde estoy
Ejemplo:

App Music
Desktop Downalds

cd Oye git quiero ir a la carpeta Desktop. Cd te lleva a esa carpeta. Si hacemos
cd Desktop y luego hacemos pwd nos aparecería: /home/kas/Desktop. Y si ahora hacemos ls: Nos dirá todos los archivos que hay en Desktop.

Creamos una carpeta para el proyecto.

"Oye Git todo lo que haya dentro de esa carpeta quiero que vayas controlando todos los cambios que haya en ella". Si a git no le digo nada, no va a controlar los cambios.
Por ello, hacemos **git innit.** Pero primero tenemos que crear la carpeta.

mkdir Make directory. mkdir LarryPlotter
Ya hemos creado nuestra carpeta.

(*) Cuando hacemos un proyecto: Trabajamos dentro de una carpeta.

cd LarryPlotter → Entrar dentro de la carpeta Larry Plotter
pwd → /home/kas/Desktop/LarryPlotter

3 zonas:

- **Working Copy :** Es la carpeta de muestro proyecto La carpeta LarryPlotter.
- **Staging Area:** donde pondremos los archivos que queremos que git vaya guardando una copia.
- **Repositorio:** Donde se almacena toda la info de los cambios.

Vamos a crear en el Working Copy(en la carpeta) un archivo: Para crear un archivo hacemos nano nombrearchivo.md :

Creamos nano index.md

(*) md → Mark Down es un formato que permite escribir texto que luego podemos convertir a html o pdf.

Creamos otro archivo: nano readme.md

Ahora en mi working copy tengo index.md y readme.md y ya le he dicho a git que controle los cambios en esa carpeta.

De Working Copy a Staging Area

Utilizamos **git add <filename> :** Añade un archivo a Staging Area
git add <folder> : Añade la carpeta (con todo lo que hay dentro)

Vamos a hacer por ejemplo git add readme.md:
Hacemos git status y vemos que está en verde readme e index está en rojo. Es decir git nos dice en verde lo que está en staging area.

git add.
Me sube todos los archivos. El punto significa la carpeta donde estoy. Entonces le estoy diciendo que me añada todos los archivos de la carpeta.

git add *.md
Ponme todos los archivos en staging area que termine en .md

git rm --cached readme.md
Quita el archive readme de Staging area

A la hora de la verdad:
Primero creo mi carpeta(Working Copy). Luego el archivo. Escribo el código. Con git add pongo en el escenario mis archivos que luego quiero hacer algo con ellos.

De Staging Area a Repository

Ahora los cambios que yo hago en el Staging area los mando / guardo en el repositorio.

¿Qué es un commit?

Cuando yo pongo el código en el Staging area y le hacemos un commit : Le estamos haciendo un pantallazo. Git ese pantallazo lo guarda en la zona repositorio y eso me va a permitir volver a ese momento.

El commit es un paquete: contiene un conjunto de información:

- Contiene 1 o + Hunks → Hunk son las líneas que han cambiado en un archivo.
- Contiene un mensaje : (Motivo x el cual hago ese cambio). Nos va a permitir saber xq cambié el código.
- Contiene un hash SHA: para identificar el comit.

Cuando yo hago un commit hago una foto de un archivo. Es decir git no guarda una foto de todo solo hace de lo que cambia (commit).

Cada commit tiene un enlace con su padre. Un commit conoce a su padre, pero no conoce a sus hijos.

git log nos permite ver el grafo que se va formando en la zona repositorio.

COMMIT C
↓
COMMIT B
↓
COMMIT A – Primer cambio que se ha hecho

Aquí vemos que se han hecho 3 cambios en el proyecos. COMIT A ES EL PADRE DE COMIT B

Si hago commit el Staging Area se queda vacio y vuelve al Working Copy y se crea el commit A. Los cambios quedan guardados en la zona repositorio y el Staging Area se queda vacio para dejar espacio al prox.

Si hacemos git status: git compara mi Working Copy frente a todo el trabajo que hay en la zona repositorio. Como he hecho commit de index y readme y desde entonces no los he modificado desde que hice el commit: me dice que todo está bien.

rm <filename> borra un archivo

git restore index.md → lo que hace es deshace cambios que acabo de hacer. Recupera la última copia guardada. Git recupera el archivo porque lo rescata de la copia en repositorio. Por lo tanto, solo se puede hacer git restore si he hecho git commit. CONTROL Z a lo bestia

cat readme.md → muestra el contenido del fichero readme.

Deshacer un commit:

PRIMER CASO
D—Master Head
C
B
A
Aquí si hago git log me saldrían los 4 commit.

Deshago commit: **git reset HEAD~1** Lo que ocurre a nivel grado es que el puntero pasa al commit anterior. En este caso al C. Pero el comit D se queda ahí es indestructible. Comit que hacemos comit que se queda ahí. Moraleja por muy malo que nos parezca el trabajo que hayamos hecho es mejor hacer un comit para que ya por lo menos se quede ahí luego si queremos ya lo borramos

SEGUNDO CASO DESHACIENDO COMIT
D
C— Master Head
B
A
Aquí si hago git log me saldrían solo los comit que apunta Head es decir → Master, comit c, comit B, comit A

Si ahora hacemos git status (compárame lo que yo tengo en mi Working Copy con lo que tu ves / los comits a los que tu tienes acceso) En el segundo caso el trabajo que yo he hecho en git con el comit D no lo tiene en cuenta

Si yo hago un comit de un archivo y a continuación hago un git status (compárame lo que yo tengo en mi Working Copy conlos comits a los que tu tienes acceso) → Me va a decir estamos al día tengo la última copia de ese código.

Si deshago el comit y hago git status → git me va a decir que hay cambios porque al deshacer ese comit ya no lo ve.