

# GIT Y EL USO DE COMANDOS.

## Cómo navegar mis comandos

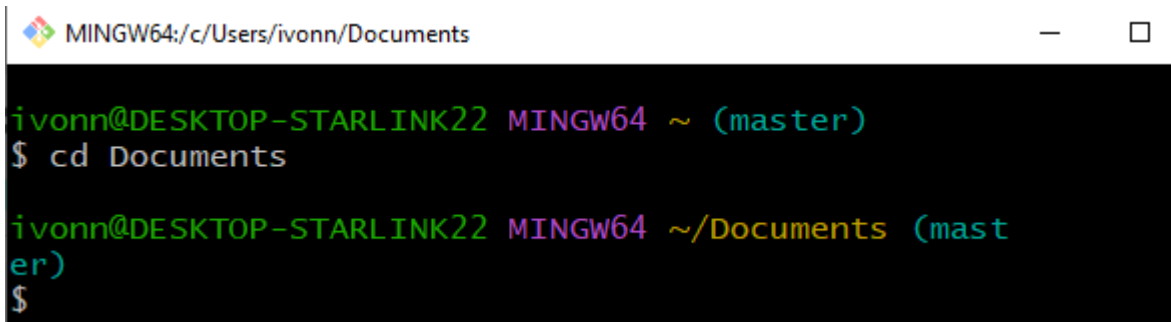
Directorio raíz es este símbolo ~

Cd comando para cambiar nuestro directorio actual

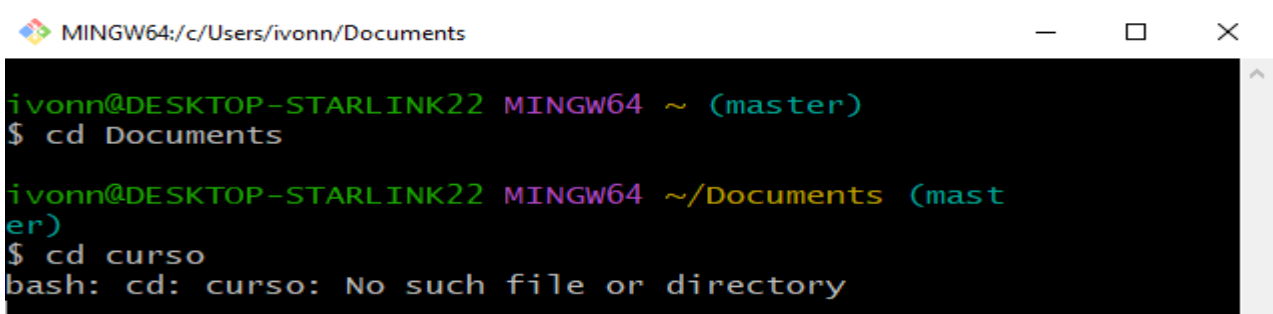


master es el directorio de git

Con este comando de cd y escribir enseguida Documents nos envia a esa carpeta



Aquí estamos buscando una carpeta llamada curso, pero esta no existe



Esto aparece cuando escribimos de esta forma y el poner espacios y ser mas largo el comando, provoca que nos aparezca la leyenda de too many arguments

## GIT Y EL USO DE COMANDOS.

```
MINGW64:/c/Users/ivonn/Documents
ivonn@DESKTOP-STARLINK22 MINGW64 ~ (master)
$ cd Documents

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents (master)
$ cd curso
bash: cd: curso: No such file or directory

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents (master)
$ cd Curso de Git y Github
bash: cd: too many arguments

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents (master)
$
```

Para resolverlo, se escribe entre comillas "Curso de Git y GitHub" y damos enter

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents (master)
$ cd "Curso de Git y Github"

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/Curso de Git y Github (master)
$ mkdir prueba

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/Curso de Git y Github (master)
$ ls
'USO DE COMANDOS EN GIT.txt'  prueba/

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/Curso de Git y Github (master)
$ ls
mi-archivo.txt  prueba/

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/Curso de Git y Github (master)
$
```

Se observa en la pantalla que se creo una carpeta y para ello se utilizo el comando llamado: mkdir al que se le dio nombre de la carpeta prueba.

Para observar la lista dentro del sistema en carpeta de Documentos con Curso de Git y GitHub, nos muestra que existe la carpeta llamada: **'USO DE COMANDOS EN GIT.txt'** que es un archivo y que también tenemos una carpeta llamada **prueba/** (Cuando observamos un /), significa que es una carpeta.

Posteriormente yo le cambie el nombre al archivo de bloc de notas ahora como: mi-archivo. Colocamos de nuevo el **comando ls**, para que nos enliste de nuevo que tenemos y se visualiza que existe ahora el archivo llamado: **mi-archivo** y la carpeta llamada **prueba/**

## GIT Y EL USO DE COMANDOS.

Ahora si queremos entrar a esa carpeta llamada prueba lo haremos usando el comando: `cd` y ya estamos dentro de la carpeta llamada prueba, como se ve en la siguiente imagen.

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/Curso de Git y G
itHub (master)
$ cd prueba

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/Curso de Git y G
itHub/prueba (master)
$ |
```

Ahora si estamos en la carpeta de prueba, pero queremos regresar al apartado de Documents que tiene a la carpeta llamada: Curso de Git y GitHub, basta con poner el comando: `cd..`. Este comando con los dos puntitos, nos va a regresar un nivel y podemos estar de nuevo en la carpeta llamada: Curso de Git y GitHub que esta alojada dentro de Documents, como se observa en la siguiente imagen:

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/Curso de Git y G
itHub (master)
$ cd prueba

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/Curso de Git y G
itHub/prueba (master)
$ cd ..

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/Curso de Git y G
itHub (master)
$
```

Ahora bien, si deseamos regresar un nivel más y estar por ejemplo en la carpeta de Documents, tenemos que poner el comando de nuevo de: `cd ..` y nos llevara a la carpeta llamada Documents, como se observa en la imagen:

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/Curso de Git y G
itHub (master)
$ cd ..

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents (master)
$ |
```

Si queremos regresar nuevamente a la carpeta de Curso de Git y GitHub y de la carpeta de prueba, entonces colocamos de nuevo los comandos siguientes:

## GIT Y EL USO DE COMANDOS.

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents (master)
$ cd "Curso de Git y GitHub"

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/Curso de Git y G
itHub (master)
$ cd prueba

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/Curso de Git y GitHub/prueba (master)
$ |
```

Si queremos regresar a la carpeta raíz ~ y comenzar de nuevo, entonces colocamos en gitbash el comando de cd (damos enter) y nos regresara a la carpeta raíz desde que empezamos este ejercicio al abrir GitBash.

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/Curso de Git y GitHub/prueba (master)
$ cd

ivonn@DESKTOP-STARLINK22 MINGW64 ~ (master)
$
```

Estos comandos son basicos para trabajar con git.

Ahora si deseamos regresar a la carpeta de mis documentos tecleamos comando: cd Documents y de ahí queremos ir a la carpeta de Curso de Git, tecleamos: "Curso de Git y GitHub" damos enter ya estamos ahí, pero quiero ver la lista que tengo ahí: tecleamos ls y observamos que tenemos el archivo llamado: mi-archivo.txt y mi carpeta llamada prueba/

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~ (master)
$ cd Documents

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents (master)
$ cd "Curso de Git y GitHub"

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/Curso de Git y G
itHub (master)
$ ls
mi-archivo.txt  prueba/
```

Para eliminar un directorio o conocido también como carpeta, en este caso eliminaremos prueba, tecleamos el comando rmdir (remove directory = eliminar o remover directorio), para eliminar prueba entonces ponemos el comando: rmdir prueba y damos enter, enseguida dentro e GitBash pondremos el comando ls, para que nos enliste de nuevo que tenemos dentro de la carpeta de Curso de Git y GitHub y se muestra que ya no esta la carpeta prueba/, pero si existe aún el archivo llamado mi-archivo.txt

## GIT Y EL USO DE COMANDOS.

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/Curso de Git y GitHub (master)
$ rmdir prueba

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/Curso de Git y G
itHub (master)
$ ls
mi-archivo.txt
```

Para limpiar la pantalla de Git Bash, teclear el comando clear y dar enter.

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/Curso de Git y GitHub (master)
$ clear

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/Curso de Git y G
itHub (master)
$ |
```

## CONFIGURAR USUARIO Y CORREO:

Se coloca el comando: git config (que significa que daremos la configuración a git, de manera global es decir, para todo lo que se realice, con el nombre de usuario, colocar su nombre: en el ejemplo puse: "Cía NERV" lo que me identificara cada vez que yo haga en un futuro un comentario (commit), para verificar que existe mi usuario, tecleamos en GitBash el comando: git config user.name (damos enter) y nos muestra que existimos como: Cía NERV.

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents (master)
$ git config --global user.name "Cía NERV"

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents (master)
$ git config user.name
Cía NERV

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents (master)
$ |
```

Ahora configuramos el correo:

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents (master)
$ git config --global user.email "ivhr1718@gmail.com"
```

Verificamos se haya creado y tecleamos: git config user.email y nos muestra ya fue creado para ser utilizado:

## GIT Y EL USO DE COMANDOS.

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents (master)
$ git config user.email
ivhr1718@gmail.com

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents (master)
$
```

### CREANDO NUESTRO PRIMER REPOSITORIO

Vamos a crear un repositorio de manera local (estará alojado en nuestra computadora), usando primero el comando: `mkdir curso-git` damos enter y ahora iremos a esa carpeta que acabamos de crear, tecleando: `cd curso-git`

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents (master)
$ mkdir curso-git

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents (master)
$ cd curso-git
```

Y ay entramos a la carpeta llamada: `curso-git`

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (master)
$ |
```

Ahora bien, vamos a crear un repositorio dentro de esta carpeta: tecleamos el comando: `git init` (donde se va a inicializar el repositorio en esta carpeta llamada `curso-git`, le damos enter y nos muestra que se ha inicializado la carpeta vacia del repositorio de Git ubicado en `C:/Users/ivonn/Documents/curso-git/.git/`

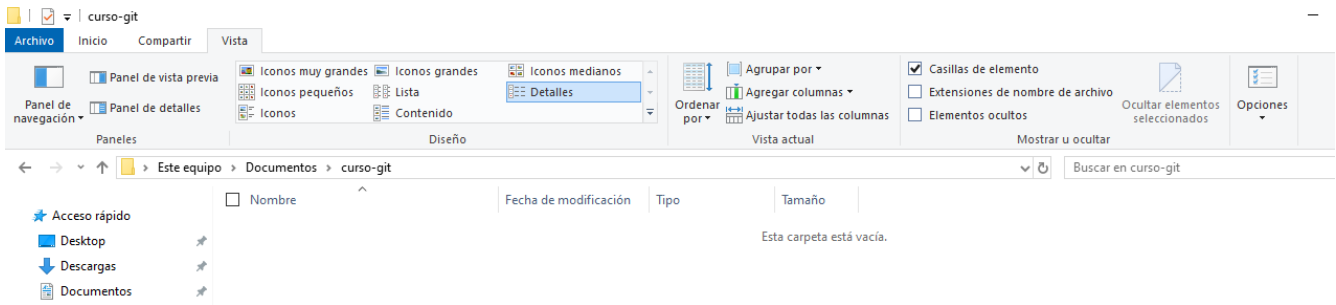
`.git` es una carpeta oculta que es donde se nos van a guardar todos los archivos del proyecto.

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (master)
$ git init
Initialized empty Git repository in C:/Users/ivonn/Documents/curso-git/.git/

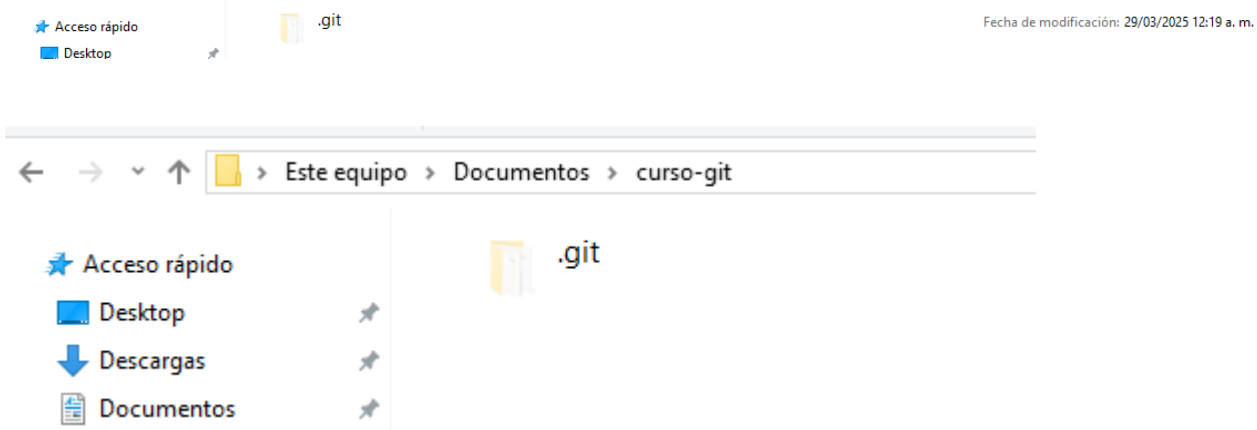
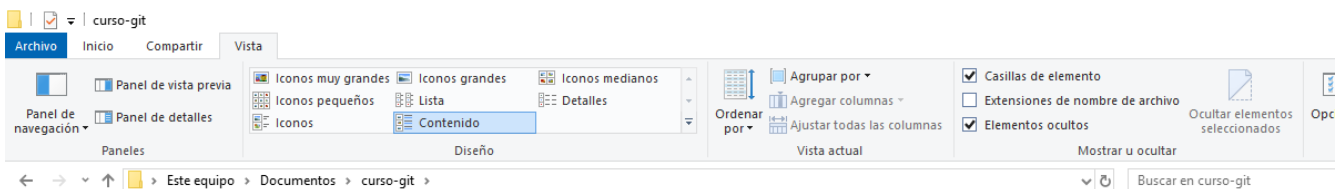
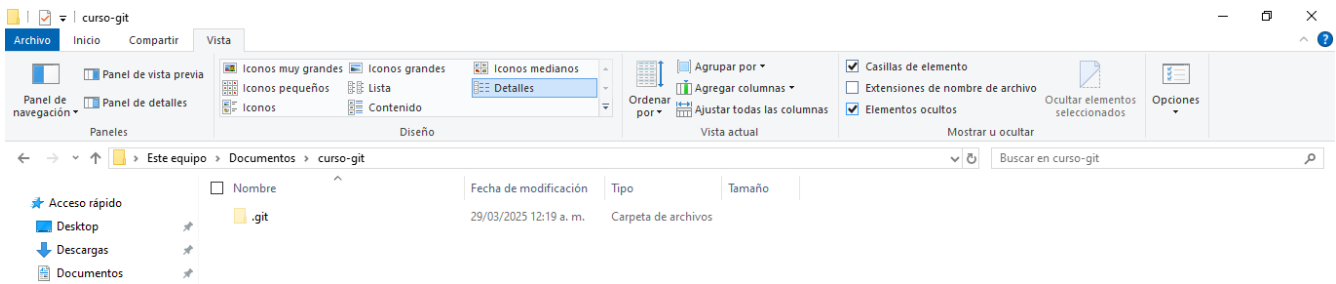
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (master)
$
```

Verificando en el sistema:

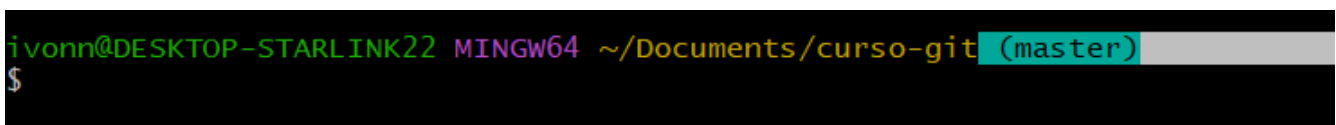
# GIT Y EL USO DE COMANDOS.



Existe la carpeta que se llama curso-git pero no vemos nada, eso es porque esta oculta, cuando se coloca dentro del comando .git (significa que estará oculta), para que que existe ahí, entonces iremos a la etiqueta Vista o llamada View, seleccionamos Elementos ocultos y ya se puede visualizar la carpeta:



Como verificamos que tenemos un repositorio, eso es porque tenemos en azul la palabra Master que es el nombre de la Rama que se creo por efecto o podría también ser llamado como: Main



## GIT Y EL USO DE COMANDOS.

Si queremos cambiar la rama y no sea Master pero aparezca como Main, entonces ejecutamos el siguiente comando: `git config --global init.defaultBranch main`

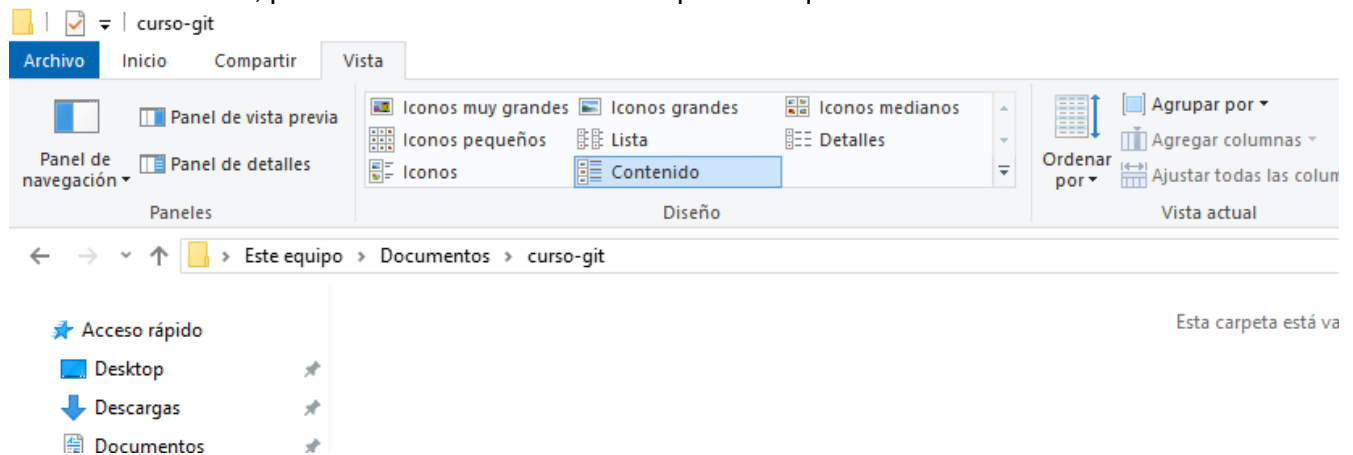
```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (master)
$ git init
Initialized empty Git repository in C:/Users/ivonn/Documents/curso-git/.git/

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (master)
$ git config --global init.defaultBranch main
```

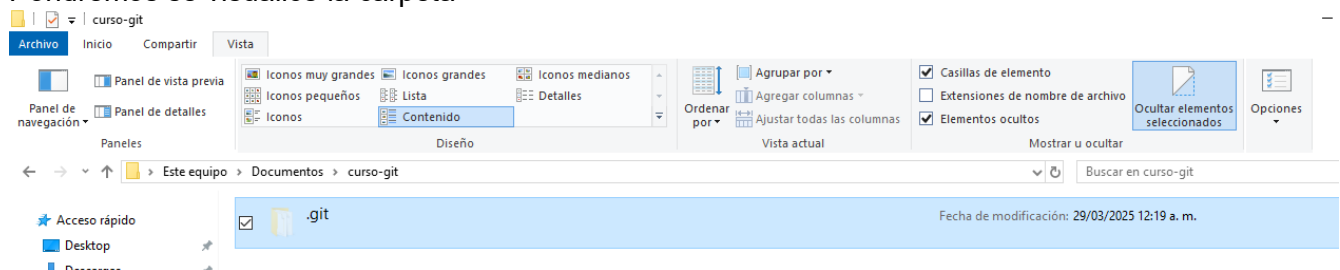
Al dar enter aparece así:

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (master)
$
```

Pero verificaremos, para ello vamos a eliminar el repositorio que teníamos



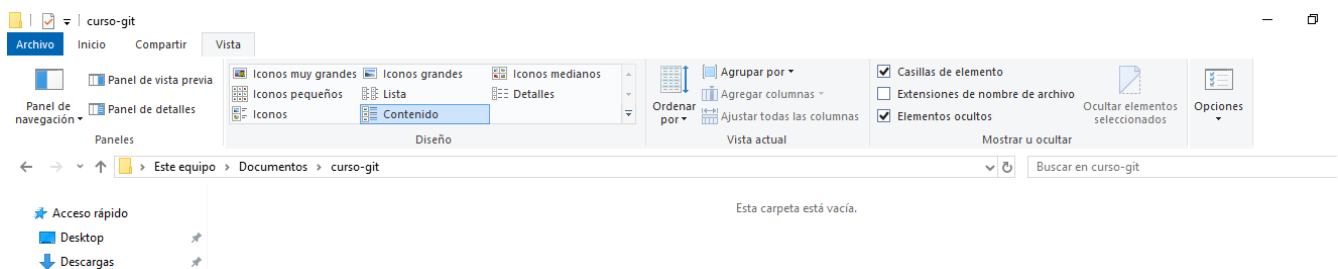
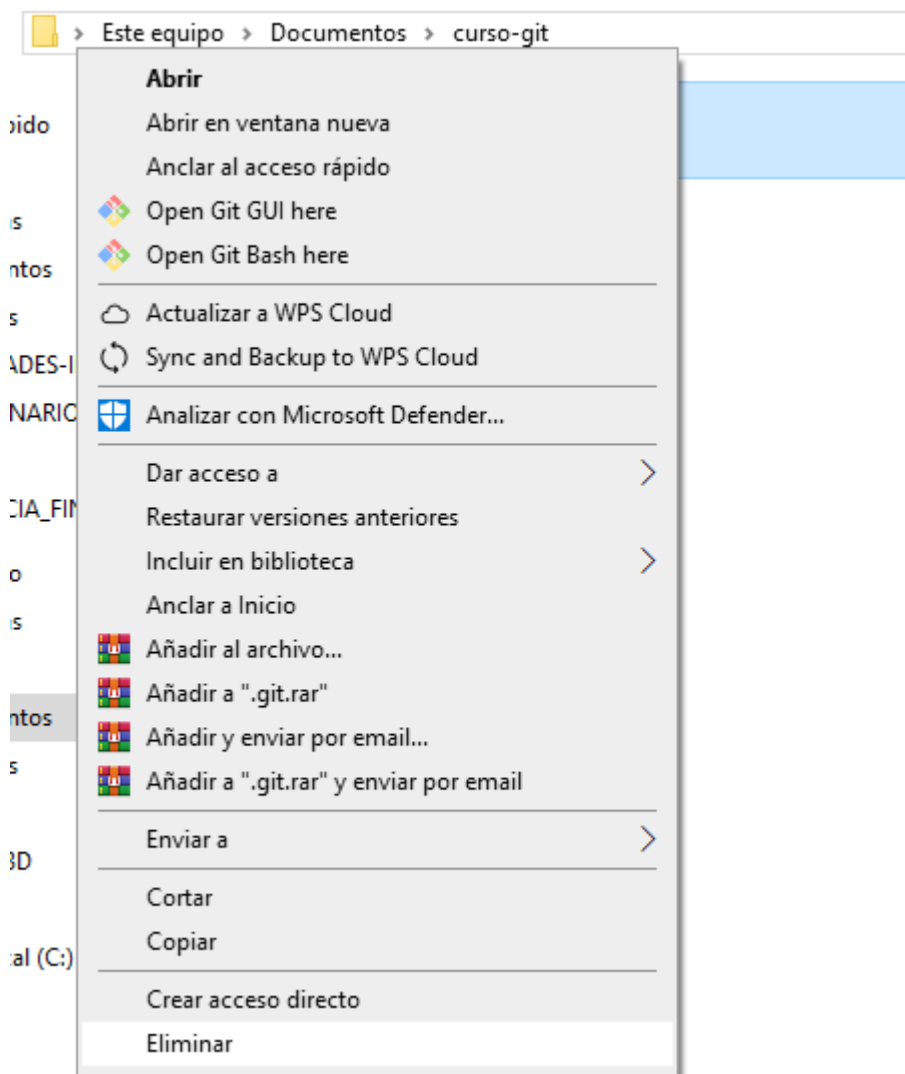
Pondremos se visualice la carpeta



Ahora la seleccionamos y la borramos:



## GIT Y EL USO DE COMANDOS.



Ahora creamos repositorio nuevo y vemos se llama en vez de Master, ahora como main y eso después de poner el comando git init damos enter y se observa:

## GIT Y EL USO DE COMANDOS.

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (master)
$

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (master)
$

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (master)
$ git init
Initialized empty Git repository in C:/Users/ivonn/Documents/curso-git/.git/

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$
```

Con estas practicas se aprendió a crear repositorio, a eliminar un repositorio, y a cambiar el repositorio con el nombre de Master a main

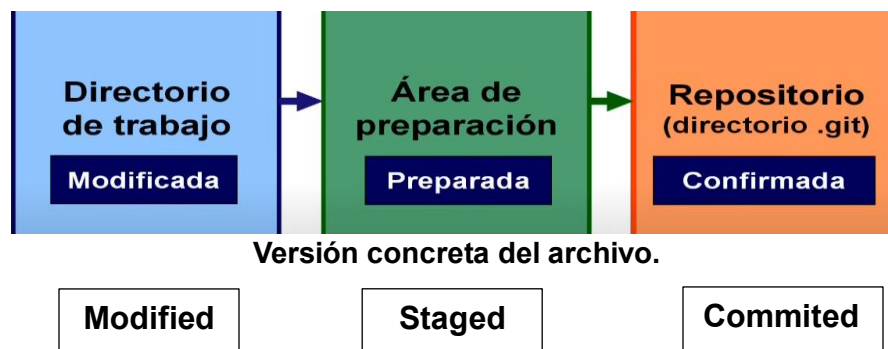
### LAS 3 ÁREAS DE GIT



**Directorio de trabajo (Working directory):** Es la carpeta del proyecto que contiene los archivos y el directorio .git del repositorio.

**Área de preparación (Staging area):** Es la intermedia que nos permite retractarnos y si queremos crear un commit, es decir, es el conjunto de archivos y cambios que serán incluidos en el próximo commit (es un registro en el tiempo de los cambios que hagamos).

**Repositorio (directorio.git):** Se crea el commit y los cambios ya serán en el repositorio git, es decir; que el Directorio que contiene los metadatos y las versiones de tu proyecto. Es la parte del repositorio que se copia cuando clonas un repositorio a tu computadora. Es la parte más importante de git



## GIT Y EL USO DE COMANDOS.

### Estados de los archivos:

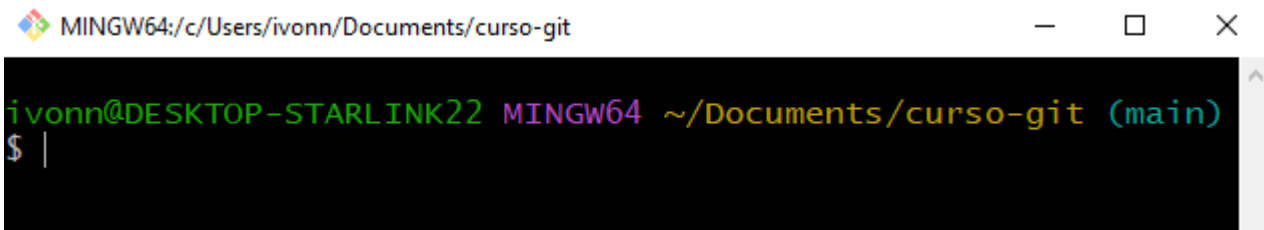
**Modificada (Modified):** Si la versión el archivo contiene cambios que **NO** son parte del repositorio y **NO** se ha añadido al área de preparación.

**Preparada (Staged):** Si la versión del archivo contiene cambios que **NO** son parte del repositorio pero fue añadida al área de preparación.

**Confirmada (Committed):** Si la versión del archivo ya se encuentra en el directorio de Git (en archivo oculto de git).

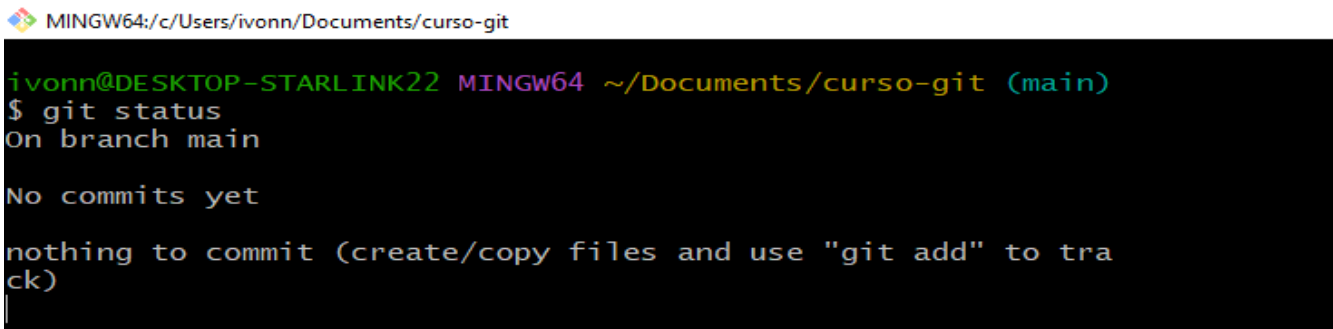
### Ejemplo: Verificando el status y crear el commit.

Vamos a verificar el estado de nuestro repositorio: Limpiamos la pantalla de Git Bash y aparecerá así, ya que estamos en Documentos y en la carpeta de curso-git:



```
MINGW64:/c/Users/ivonn/Documents/curso-git
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ |
```

Al verificar el status, escribimos el comando: git status y damos enter, aparecerá lo siguiente, que estamos en la rama principal llamada main y no tenemos aun commits y que tampoco hay nada pendiente para realizar un commit:



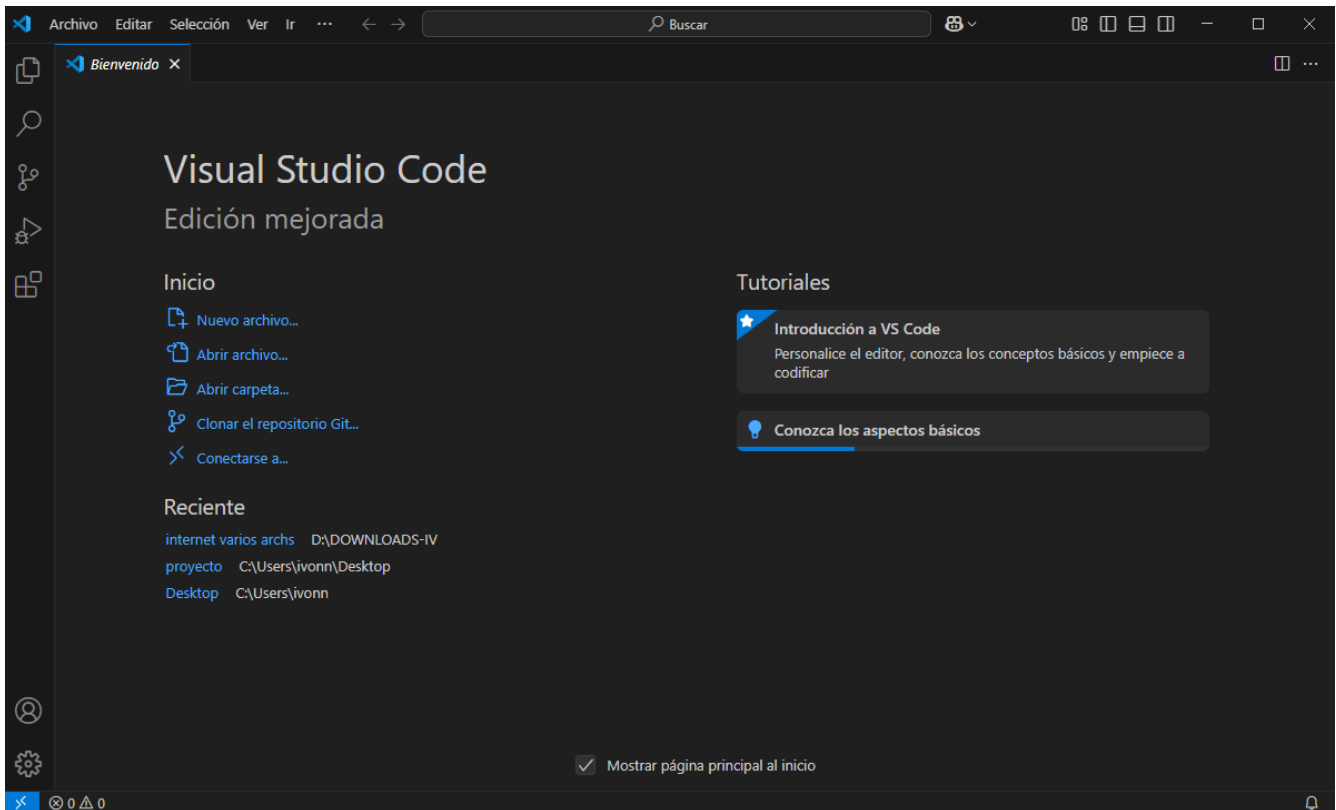
```
MINGW64:/c/Users/ivonn/Documents/curso-git
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ git status
On branch main

No commits yet

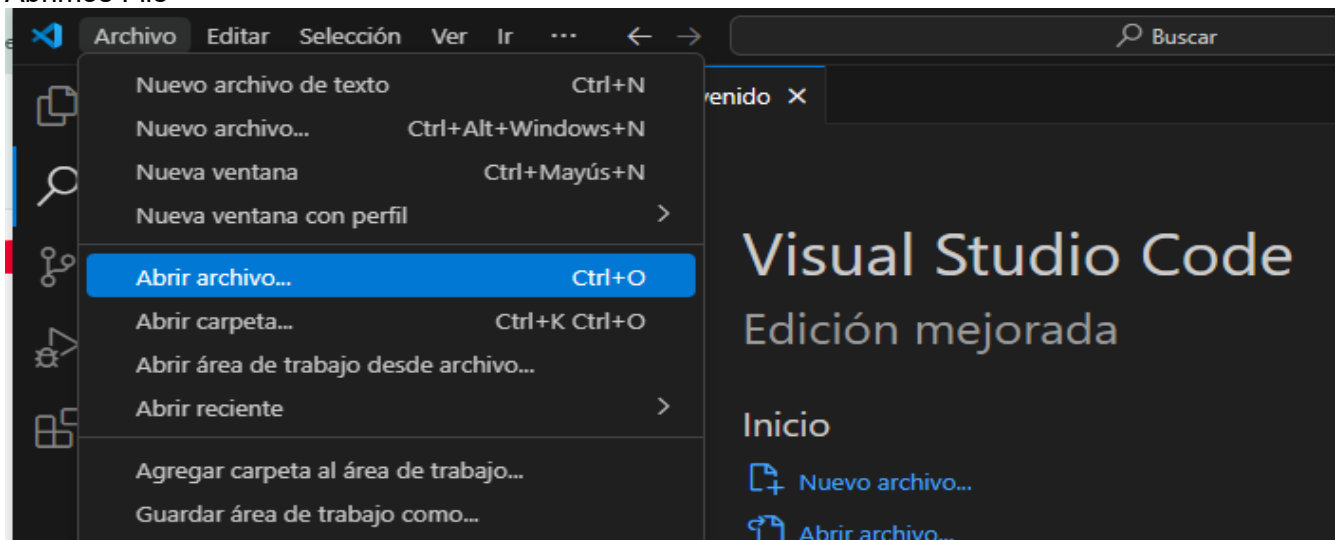
nothing to commit (create/copy files and use "git add" to track)
```

Y esto nos dice: que no estamos rastreando ningún archivo nothing to commit (create/copy files and use "git add" to track).

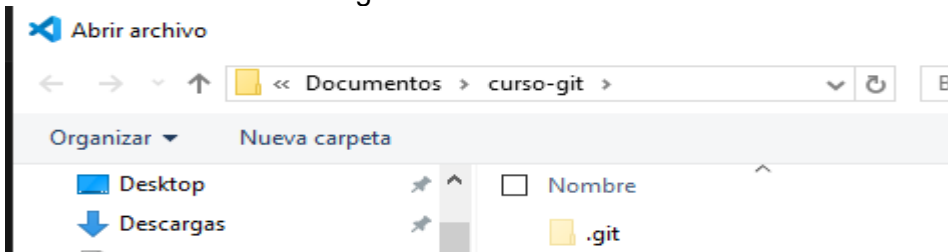
# GIT Y EL USO DE COMANDOS.



Abrimos File

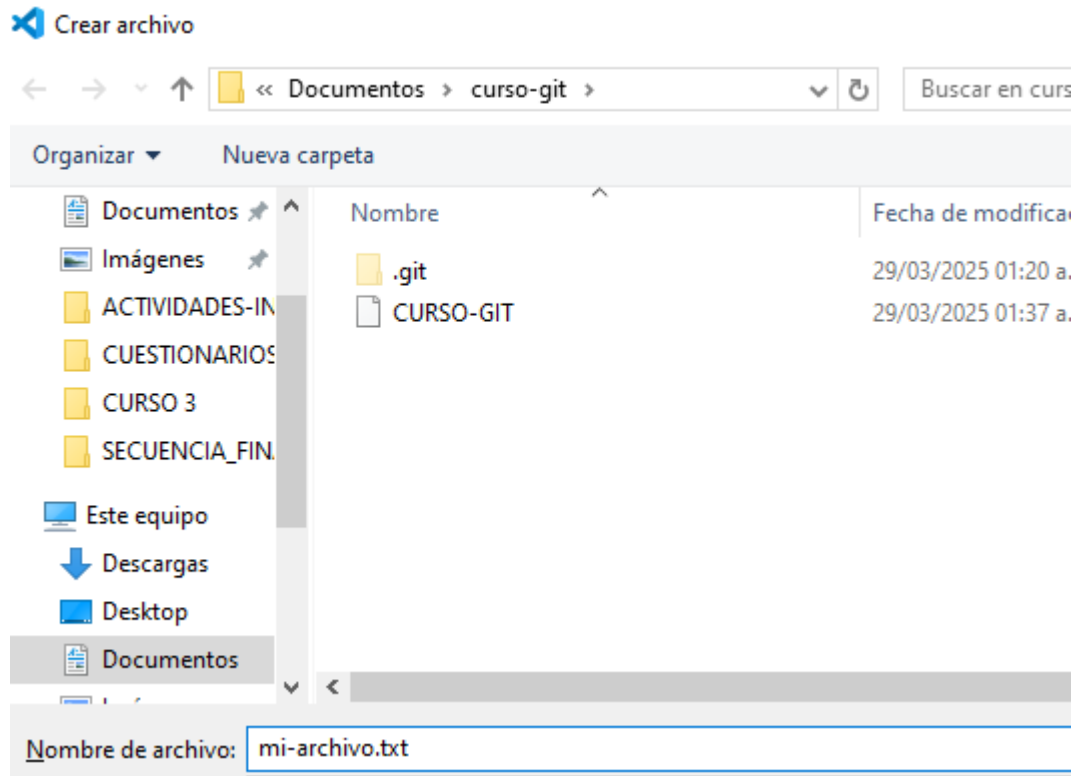
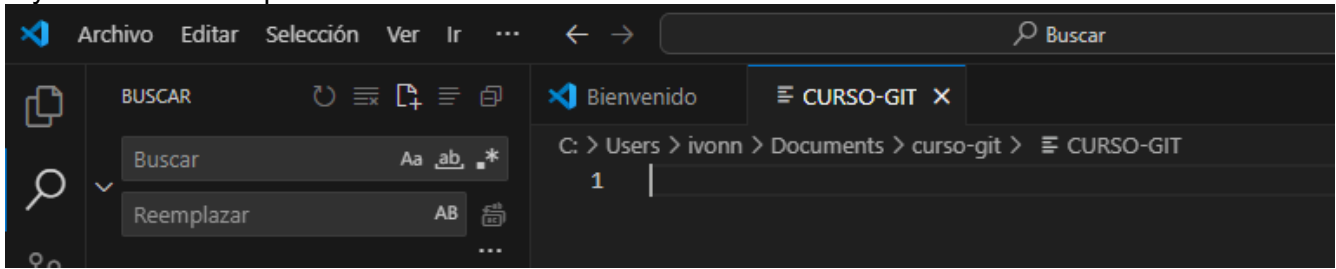


Abrimos el folder de curso-git

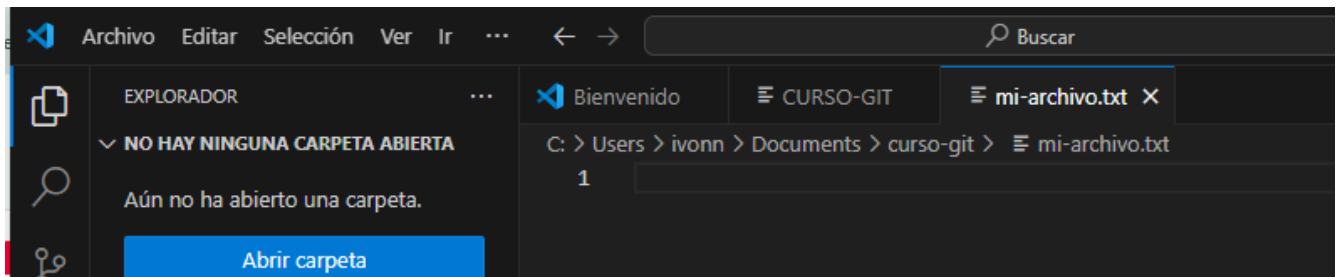


## GIT Y EL USO DE COMANDOS.

Y ya tenemos la carpeta abierta



Se crea en la carpeta de curso-git y ya aparece



Volvemos a GitBash y escribimos comando de nuevo: git status:

## GIT Y EL USO DE COMANDOS.

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        CURSO-GIT
        mi-archivo.txt

nothing added to commit but untracked files present (use "git add" to track)

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ |
```

Nos dice que no hemos agregado nada al commit, y para agregarlo deberemos usar add. Para el archivo de texto mi-archivo.txt, al darle enter no vamos a ver ningún cambio.

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ git add mi-archivo.txt

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$
```

Entonces si volvemos a poner git status observamos que el archivo que fue agregado de mi-archivo.txt que antes se veía en rojo, ahora se visualiza en color verde.

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        CURSO-GIT
        mi-archivo.txt

nothing added to commit but untracked files present (use "git add" to track)

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ git add mi-archivo.txt

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   mi-archivo.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        CURSO-GIT
```

## GIT Y EL USO DE COMANDOS.

Que seguimos en la rama main, no tenemos commits todavía y que si tenemos cambios que van a ser agregados al proximo commit

En este caso tenemos un nuevo archivo que agregamos:

new file: mi-archivo.txt

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git
$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   mi-archivo.txt
```

Ahora si no queremos agregar ese archivo en la preparación, el mismo git nos dice utiliza el comando git rm --cached y ponemos el nombre del archivo que fue entonces: git rm --cached mi-archivo.txt

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ git rm --cached mi-archivo.txt
rm 'mi-archivo.txt'

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ |
```

Y nos aparece al dar enter, la línea de: rm 'mi-archivo.txt'

Ahora si volvemos a verificar el estado del archivo, volvemos a verlo en color rojo:

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ git rm --cached mi-archivo.txt
rm 'mi-archivo.txt'

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    CURSO-GIT
    mi-archivo.txt

nothing added to commit but untracked files present (use "git add" to track)

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ |
```

## GIT Y EL USO DE COMANDOS.

Y significa que ese archivo en particular ya no esta siendo parte de esos cambios para ser un commit y por eso de nuevo se pone en rojo.

Si quiero volver a ponerlo en la preparación, entonces de nuevo usamos el comando git add mi-archivo.txt (damos enter) y para verificar el estatus volvemos a poner comando git status y damos enter y se observa que de nuevo el archivo se agrego y cambio a color verde.

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ git add mi-archivo.txt

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ git status
On branch main

No commits yet

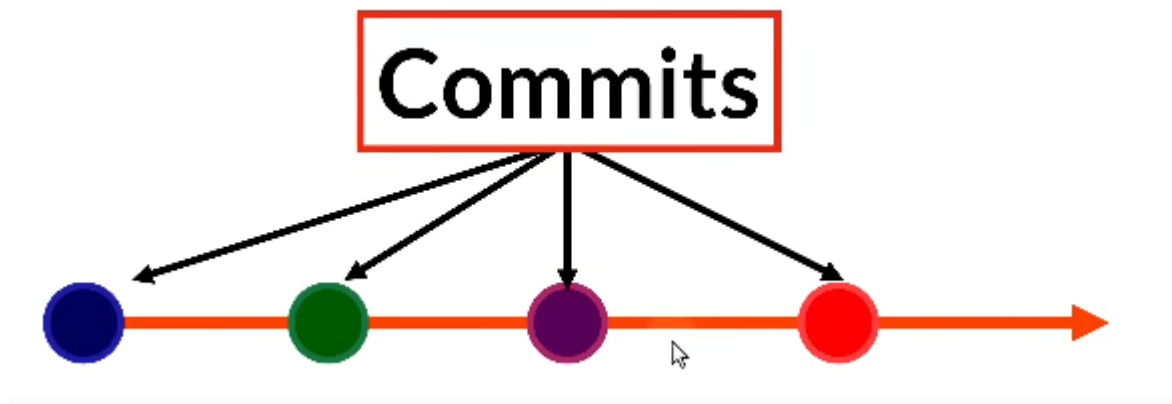
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   mi-archivo.txt
```

Ahora del área de preparación lo vamos a mandar al repositorio como tal

## COMMITTS

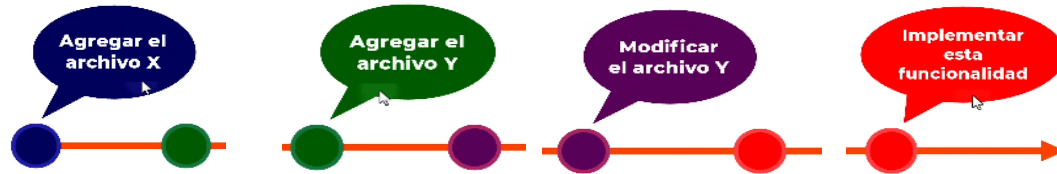
**Commit (Comprometerse):** Es un componente básico de la línea del tiempo de un proyecto de Git, es un registro o “foto” del estado de un proyecto en un momento específico. El commit va a verificar que se agrego al archivo, que fue modificado en cada línea, si se elimino un archivo, si se cambio un archivo, si se creo una carpeta, si se movio un archivo a otra carpeta; todo este tipo de cambios que se hagan entre un commit y otro va a estar registrado en el próximo commit.

Entonces, el commit; registra los cambios que se realizaron en los archivos en comparación con la versión anterior.





## GIT Y EL USO DE COMANDOS.



El commit tiene un identificador único, lo que se traduce como un algoritmo llamado Hash, y este ayudara a identificar:



Y se asociaría a ese commit, también el nombre del usuario, la fecha y el email de quien haya ejecutado ese commit.

## CREAR UN COMMIT.

Verificamos el status con el comando git status y damos enter

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   mi-archivo.txt
```

Ahora vamos a agregar un commit:

Escribimos comando git commit, entonces escribimos el comando:

git commit -m "Agregar archive de texto" (significa que vamos a agregar un commit, usando -m que es el mensaje y este será: "Agregar archivo de texto" damos enter y se observa asi:

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ git commit -m "Agregar archive de texto"
[main (root-commit) 04231d7] Agregar archivo de texto
1 file changed, 1 insertion(+)
 create mode 100644 mi-archivo.txt

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ |
```

## GIT Y EL USO DE COMANDOS.

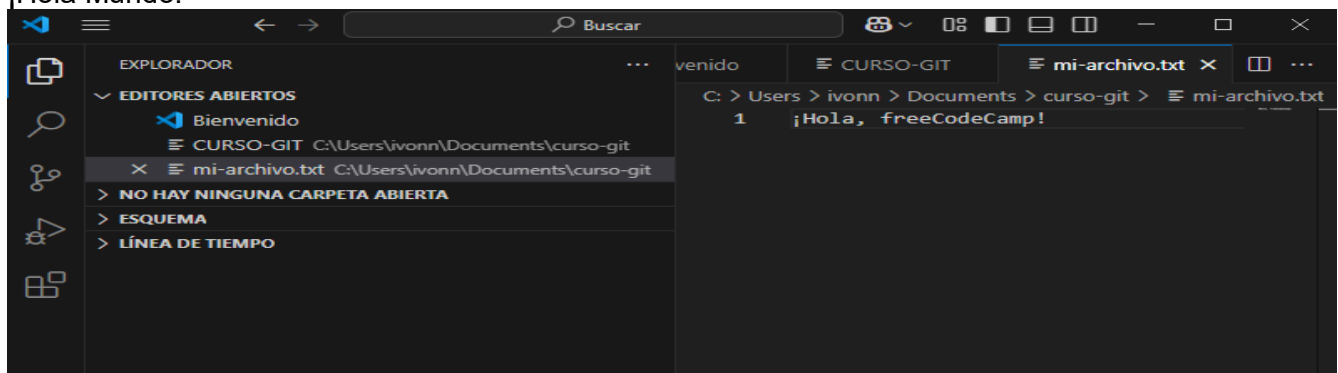
Para verificar que mi commit fue registrado, usamos el comando git log, que nos va a mostrar una secuencia o el historial de commits que se hayan registrado:

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ git log
commit 04231d7a3299cdc890d01f81884529a6baf33823 (HEAD -> main)
Author: Cia NERV <ivhr1718@gmail.com>
Date: Sat Mar 29 02:46:37 2025 -0600

    Agregar archivo de texto
```

Observamos la estructura que tiene el commit: Presenta un identificador sha (la serie de números y letras) que están en la rama de main, presenta el autor que es Cia NERV con el email registrado y también se observa la fecha en que se hizo: sábado 29 de marzo con la hora. Y ya tenemos nuestro commit.

Otra manera de hacer un commit es con VSC, vamos a él y modificamos la palabra que teníamos de ¡Hola Mundo!



Y la modificamos ahí mismo por: ¡Hola FreeCodeCamp!, le damos guardar al archivo en VSC y lo verificamos en GitHub. Escribiendo el comando git status y nos muestra que se hizo un cambio:

```
MINGW64:/c/Users/ivonn/Documents/curso-git

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working
  directory)
        modified:   mi-archivo.txt

Untracked files:
  (use "git add <file>..." to include in what will be committ
  ed)
        CURSO-GIT

no changes added to commit (use "git add" and/or "git commit
-a")

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$
```

## GIT Y EL USO DE COMANDOS.

El archivo fue modificado y aun no se ha agregado, ahora usamos comando git add .

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working
directory)
        modified:   mi-archivo.txt

Untracked files:
  (use "git add <file>..." to include in what will be committ
ed)
        CURSO-GIT

no changes added to commit (use "git add" and/or "git commit
-a")

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ git add .

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   CURSO-GIT
        modified:   mi-archivo.txt

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$
```

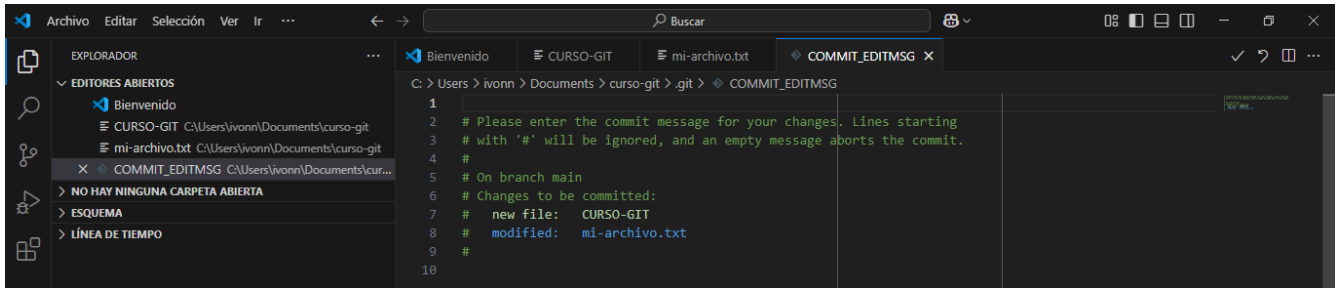
Y el cambio fue hecho y listo para ser agregado al commit.

Ahora agregamos otro commit con un editor de texto y si esta bien configurado VSC, solo ponemos el comando de git commit y VSC nos estará esperando para realizarlo

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ git commit
hint: Waiting for your editor to close the file... |
```

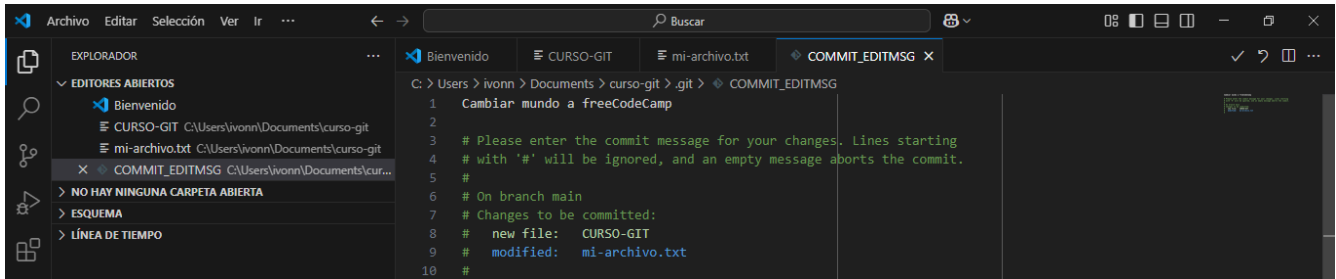
Lo visualizamos en VSC

## GIT Y EL USO DE COMANDOS.



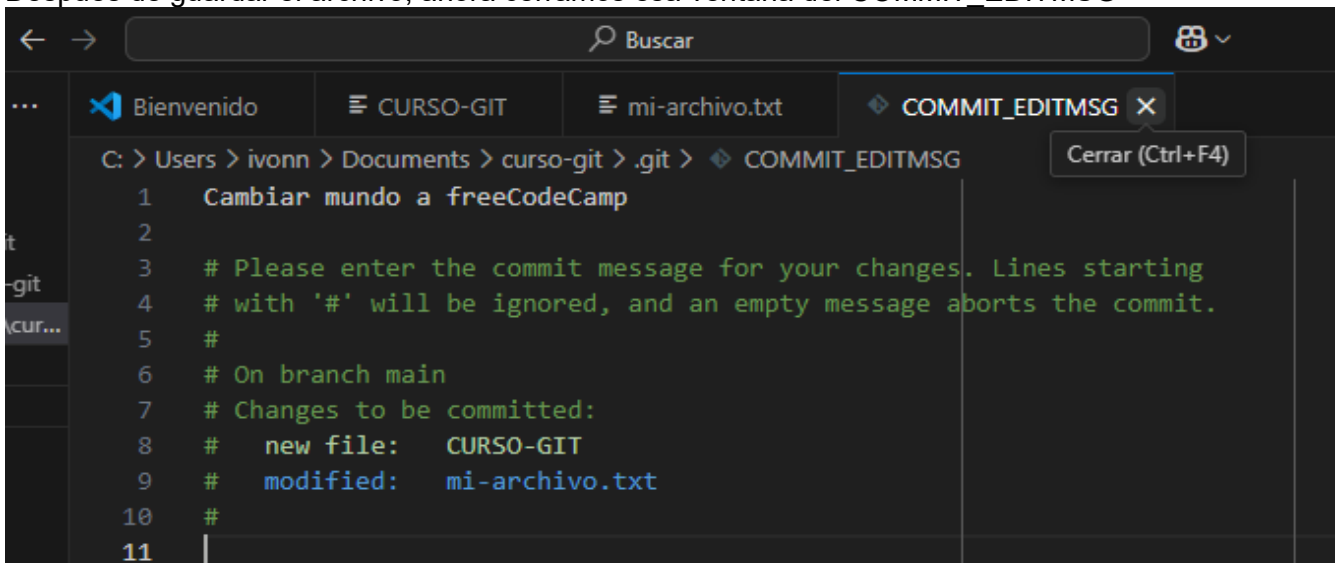
```
C: > Users > ivonn > Documents > curso-git > .git > COMMIT_EDITMSG
1
2 # Please enter the commit message for your changes. Lines starting
3 # with '#' will be ignored, and an empty message aborts the commit.
4 #
5 # On branch main
6 # Changes to be committed:
7 #   new file:   CURSO-GIT
8 #   modified:   mi-archivo.txt
9 #
10
```

Se agrego lo siguiente: Cambiar mundo a freeCodeCamp y guardamos con Ctrl+S en VSC



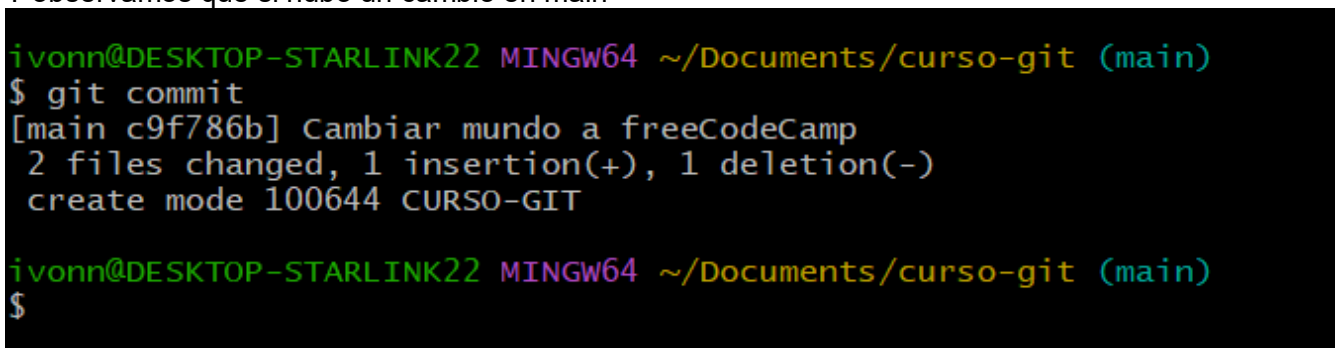
```
C: > Users > ivonn > Documents > curso-git > .git > COMMIT_EDITMSG
1 Cambiar mundo a freeCodeCamp
2
3 # Please enter the commit message for your changes. Lines starting
4 # with '#' will be ignored, and an empty message aborts the commit.
5 #
6 # On branch main
7 # Changes to be committed:
8 #   new file:   CURSO-GIT
9 #   modified:   mi-archivo.txt
10
```

Despues de guardar el archivo, ahora cerramos esa ventana del COMMIT\_EDITMSG



```
C: > Users > ivonn > Documents > curso-git > .git > COMMIT_EDITMSG
1 Cambiar mundo a freeCodeCamp
2
3 # Please enter the commit message for your changes. Lines starting
4 # with '#' will be ignored, and an empty message aborts the commit.
5 #
6 # On branch main
7 # Changes to be committed:
8 #   new file:   CURSO-GIT
9 #   modified:   mi-archivo.txt
10
11
```

No se observa cambio alguno, pero nos vamos a Git Bash  
Y observamos que si hubo un cambio en main




```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ git commit
[main c9f786b] Cambiar mundo a freeCodeCamp
2 files changed, 1 insertion(+), 1 deletion(-)
create mode 100644 CURSO-GIT

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$
```

## GIT Y EL USO DE COMANDOS.


Limpiamos pantalla usando el comando clear y damos enter.

Ahora queremos ver los commits que hayamos hecho o se hayan realizado, escribimos comando: git log y nos muestra los commits que se han hecho:

 MINGW64:/c/Users/ivonn/Documents/curso-git

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ git log
```

Damos enter

 MINGW64:/c/Users/ivonn/Documents/curso-git

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ git log
commit c9f786bc3202d2a96cdf9cf945e0c572f6a66c4c (HEAD -> main)
Author: Cía NERV <ivhr1718@gmail.com>
Date: Sat Mar 29 02:59:59 2025 -0600


    Cambiar mundo a freeCodeCamp

commit 04231d7a3299cdc890d01f81884529a6baf33823
Author: Cía NERV <ivhr1718@gmail.com>
Date: Sat Mar 29 02:46:37 2025 -0600

    Agregar archivo de texto
```

## EDITOR DE TEXTO PARA LOS COMMITS.

Para asociar Visual Code (VSC) sino fue configurado por defecto cuando se instalo y ser usado con git, se utiliza este comando: git config --global core.editor "code --wait" y dar enter y ya quedo enlazado.

 MINGW64:/c/Users/ivonn/Documents/curso-git

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ git config --global core.editor "code --wait"
```

Si lo que se quiere es enlazar por ejemplo el editor para usar con git de Sublime text

<https://docs.github.com/es/get-started/git-basics/associating-text-editors-with-git>

# GIT Y EL USO DE COMANDOS.

## Usar Sublime Text como editor [↗](#)

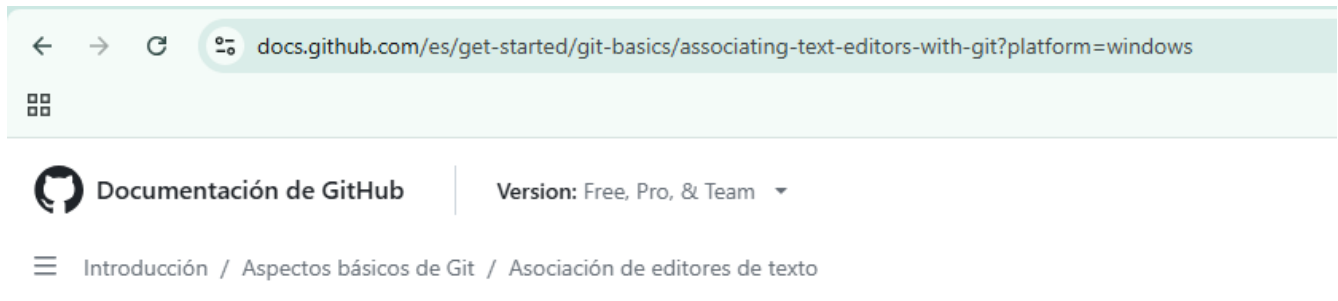
- 1 Instale [Sublime Text](#). Para obtener más información, consulta [Instalación](#) en la documentación de Sublime Text.
- 2 Abra Git Bash.
- 3 Escriba el siguiente comando:

```
git config --global core.editor "'C:/Program Files (x86)/sublime text 3/subl.exe' -w"
```

## Usar Notepad++ como editor [↗](#)

- 1 Instale Notepad++ desde <https://notepad-plus-plus.org/>. Para obtener más información, consulta [Introducción](#) en la documentación de Notepad++.
- 2 Abra Git Bash.
- 3 Escriba el siguiente comando:

```
git config --global core.editor "'C:/Program Files (x86)/Notepad++/notepad++.exe' - multiInst -notabbar -nosession -noPlugin"
```



## Uso de Visual Studio Code como editor [↗](#)

- 1 Instala [Visual Studio Code](#) (VS Code). Para obtener más información, consulta la documentación de VS Code, la sección [Configuración de VS Code](#).
- 2 Abra Git Bash.
- 3 Escriba el siguiente comando:

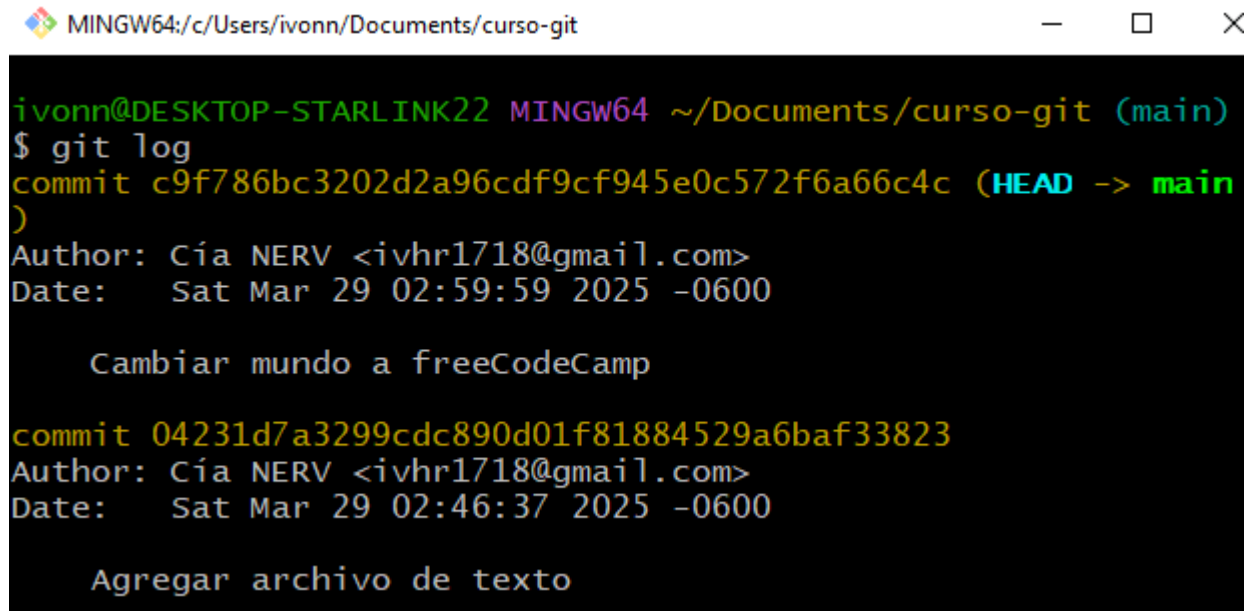
```
git config --global core.editor "code --wait"
```

# GIT Y EL USO DE COMANDOS.

## MODIFICAR UN COMMIT.

Se recomienda usar este comando de -amend solo en repositorio local

Tenemos estos repositorios, lo observamos con el comando git log y damos enter



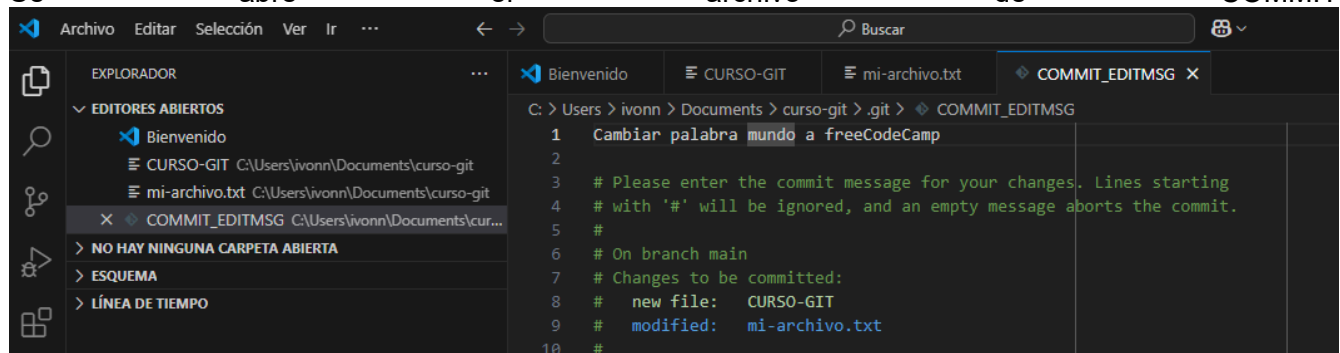
```
MINGW64:/c:/Users/ivonn/Documents/curso-git
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ git log
commit c9f786bc3202d2a96cdf9cf945e0c572f6a66c4c (HEAD -> main)
Author: Cía NERV <ivhr1718@gmail.com>
Date: Sat Mar 29 02:59:59 2025 -0600

    Cambiar mundo a freeCodeCamp

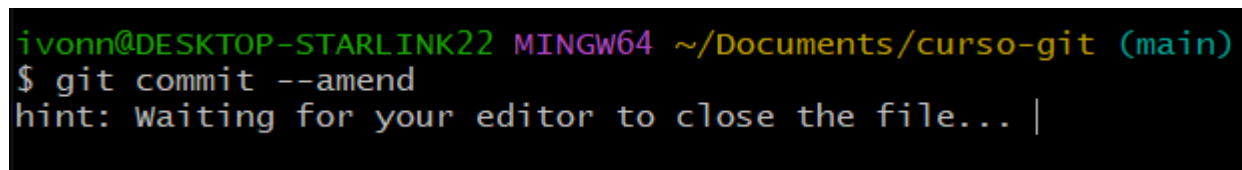
commit 04231d7a3299cdc890d01f81884529a6baf33823
Author: Cía NERV <ivhr1718@gmail.com>
Date: Sat Mar 29 02:46:37 2025 -0600

    Agregar archivo de texto
```

Se abre el archivo de COMMIT



Modificamos cambiar mundo a freeCodeCamp, le ponemos ahora: Cambiar palabra mundo a freeCodeCamp, le damos guardar en Visual Coce y cerramos el archivo.



```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ git commit --amend
hint: Waiting for your editor to close the file... |
```

Al hacer la modificación en la línea de texto darle guardar y cerrar la pestaña de COMMIT en VSC aparece esto en Git Bash

## GIT Y EL USO DE COMANDOS.

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ git commit --amend
[main 2248ad0] Cambiar el mundo a freeCodeCamp
Date: Sat Mar 29 02:59:59 2025 -0600
2 files changed, 1 insertion(+), 1 deletion(-)
create mode 100644 CURSO-GIT

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$
```

Verificamos el ultimo commit con el comando: git log

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ git log
commit 2248ad06e526d4f2af2245a32a6b355bf89f97b1 (HEAD -> main)
Author: Cía NERV <ivhr1718@gmail.com>
Date: Sat Mar 29 02:59:59 2025 -0600

    Cambiar el mundo a freeCodeCamp

commit 04231d7a3299cdc890d01f81884529a6baf33823
Author: Cía NERV <ivhr1718@gmail.com>
Date: Sat Mar 29 02:46:37 2025 -0600

    Agregar archivo de texto

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$
```

Y se muestra la modificación en el commit, con la frase: Cambiar el mundo a freeCodeCamp.

## REVERTIR O DESHACER EL ÚLTIMO COMMIT.

Si quiero revertir comentario en Commit.

Este comando nos permite reiniciar el comentario de los commits (reset) de una forma suave (Soft), lo que significa que no se van a modificar los archivos en el directorio, lo que se va a deshacer es el registro del commit y con HEAD es una forma de referirnos al ultimo commit que se hizo en la rama main y queremos retroceder (~1) 1 commit en el historial, el más reciente queremos eliminarlo.



## GIT Y EL USO DE COMANDOS.

```
MINGW64:/c/Users/ivonn/Documents/curso-git
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ git reset --soft HEAD~1

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ |
```

Damos enter no se vera nada, pero ponemos el comando: git log y nos muestra el commit

```
MINGW64:/c/Users/ivonn/Documents/curso-git
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ git reset --soft HEAD~1

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ git log
commit 04231d7a3299cdc890d01f81884529a6baf33823 (HEAD -> main)
Author: Cía NERV <ivhr1718@gmail.com>
Date: Sat Mar 29 02:46:37 2025 -0600

    Agregar archivo de texto
```

En lugar de dos, como se veían con anterioridad, si se quiere restaurar, el mismo git nos dice que comando utilizar:

```
ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   CURSO-GIT
    modified:   mi-archivo.txt

ivonn@DESKTOP-STARLINK22 MINGW64 ~/Documents/curso-git (main)
$ |
```

## GIT Y EL USO DE COMANDOS.

### COMANDOS:

Ver los cambios en un proyecto que se ha creado, modificado o eliminado.	git status
Agregar los cambios al repositorio	git add
Confirmar los cambios/ registrar los cambios y añadirlos al historial	git commit.
Ver los cambios en un archivo	git show
Ver la diferencia entre dos versiones	git diff commitA commitB.
Ver el historial de cambios de un archivo	git log -- [filename].
Deshacer los cambios	git reset

Actualizar el mensaje de un commit	git commit --amend
Revertir cambios	git revert
Forzar las actualizaciones de los cambios	git push --force

- ❖ **Rama en Git:** Una línea de desarrollo independiente
- ❖ **Ventaja principal de utilizar ramas en Git:** permitir trabajar en características nuevas sin afectar la rama principal
- ❖ **Comando para crear una nueva rama en Git:** git branch nombre\_de\_rama
- ❖ **Comando se utiliza para renombrar una rama existente en Git:** git branch -m nombre\_actual nuevo\_nombre
- ❖ **Para cambiar a una nueva rama inmediatamente después de crearla, se puede usar el siguiente comando:** git checkout -b <nuevo\_nombre>
- ❖ **Al intentar renombrar una rama, qué sucede si el nuevo nombre ya existe:** Git mostrará un error
- ❖ **Comando para renombrar una rama existente en Git:** git branch -m nombre\_actual nuevo\_nombre
- ❖ **Si deseas cambiar a una nueva rama inmediatamente después de crearla, se puede utilizar:** git checkout -b <nuevo\_nombre>

### COMANDOS: Mostrar, moverse y eliminar las ramas. Commit en stash

- ❖ **Comando para listar todas las ramas disponibles:** git branch
- ❖ **Si deseas eliminar una rama llamada "bugfix-123", cuál sería el comando correcto:** git branch -d bugfix-123

## GIT Y EL USO DE COMANDOS.

- ❖ **Para ver la rama en la que actualmente te encuentras trabajando, se utiliza:** `git branch --current`
- ❖ **Diferencia entre un commit y un git stash:** Un git stash guarda cambios temporalmente, mientras que un commit los guarda permanentemente.
- ❖ **Comando que guarda los cambios actuales de trabajo en el stash:** `git stash`
- ❖ **Comando que se usa para crear una nueva rama a partir de un stash existente:** `git stash branch <branch-name>`
- ❖ **Al aplicar un stash específico identificado por "stash@{2}", se implementaría:** `git apply stash@{2}`
- ❖ **Para ver la lista de todos los stashes guardados, se utiliza el comando de:** `git stash list`
- ❖ **Comando que se utiliza para aplicar los cambios guardados en el stash y eliminarlos de la lista de stash al mismo tiempo:** `git stash pop`
- ❖ **Para aplicar los cambios guardados en el stash a la rama actual y sin eliminar el stash, se implementa el comando de:** `git stash apply`
- ❖ **Comando elimina el stash aplicado recientemente:** `git stash drop`

### COMANDOS: merge ramas, unir ramas, conflictos, unión automática.

- ❖ **Comando para hacer merge de una rama en otra:** `git merge branch_name`
- ❖ **Comando correcto para hacer un merge de la rama feature en la rama main:** `git checkout main` y luego `git merge feature`
- ❖ **Al fusionar (merge) una rama en la rama principal:** Combina los cambios de la rama en la rama principal.
- ❖ **Conflicto en un merge es donde:** las modificaciones simultáneas en la misma parte del archivo desde diferentes ramas.
- ❖ **Comando que se utiliza para abortar un merge en caso de conflictos:** `git merge --abort`
- ❖ **La unión (merge) en Git es:** Combinar los cambios de dos ramas en una sola
- ❖ **Un conflicto en Git:** Representa cuando dos ramas contienen cambios incompatibles en el mismo archivo
- ❖ **Si hay conflictos durante el merge, para resolverlos primero hay que:** Editar manualmente los archivos conflictivos
- ❖ **Después de resolver los conflictos y guardar los cambios, se implementa el comando para finalizar el merge con:** `git commit`
- ❖ **El comando merge fast-forward en Git es:** Un merge que simplemente mueve el puntero de la rama a la punta de la rama fusionada
- ❖ **Lo que caracteriza a merge fast-forward es:** Que la rama simplemente avanza al último commit de la rama fusionada.
- ❖ **Ventaja de usar un merge fast-forward:** La de mantener un historial de commits lineal y limpio
- ❖ **En Git un merge de unión automática es:** Un merge que Git puede realizar automáticamente sin conflictos
- ❖ **En Git se utiliza para realizar una unión automática de ramas el comando de:** `git merge`
- ❖ **Situación en la que se puede hacer un merge fast-forward en Git:** Cuando la rama de destino no ha tenido nuevos commits desde que se creó la rama fuente.
- ❖ **Que pasa con el historial de commits cuando se realiza un merge fast-forward:** El historial de commits permanece lineal.
- ❖ **En Git un conflicto de fusión en Git nos representa:** Que hay divergencias en los archivos que deben resolverse

## GIT Y EL USO DE COMANDOS.

- ❖ **Comandos que se implementan para resolver conflictos:** git add y git commit
- ❖ **Después de resolver un conflicto de fusión, se utilizará para completar la fusión, el comando de:** git commit
- **Cuando la rama de destino no tiene nuevos commits desde que se creó la rama fuente, que tipo de merge sucede:** El Fast-forward merge

### Tags Etiquetas y git reset.

¿Qué es una etiqueta en Git?

**Un marcador temporal específico en la historia del proyecto**

¿Qué comando de Git se utiliza para crear una etiqueta ligera?

**git tag**

¿Cómo se eliminan etiquetas en Git?

**git tag -d <nombre\_etiqueta>**

¿Cuál es la diferencia principal entre una etiqueta ligera y una anotada en Git?

**Una etiqueta ligera es solo un marcador, mientras una anotada incluye un mensaje**

¿Qué opción se usa para crear una etiqueta anotada en Git?

**-a**

¿Cómo lista todas las etiquetas en el repositorio?

**git tag --list**

¿Qué comando se usa para eliminar una etiqueta local llamada v1.0 en Git?

**git tag -d v1.0**

¿Cuál es el comando que muestra los detalles de una etiqueta anotada en Git?

**git show <nombre-de-la-etiqueta>**

¿Qué comando se utiliza para empujar todas las etiquetas locales a un repositorio remoto?

**git push --tags**

¿Qué comando se utiliza para regresar de manera permanente a un commit específico en Git?

**git reset <commit>**

¿Qué opción del comando git reset borra los cambios en el área de trabajo?

**--hard**

¿Qué ocurre cuando se utiliza git reset --soft?

**Los cambios se conservan en el área de trabajo y en el índice.**

¿Qué comando elimina el commit más reciente y borra los cambios en el área de trabajo?

**git reset --hard HEAD~1**

## GIT Y EL USO DE COMANDOS.

¿Qué diferencia hay entre `git reset --mixed` y `git reset --hard`?

`git reset --mixed` conserva los cambios en el área de trabajo, `git reset --hard` los borra.

### Git Stash y Git Rebase - Para realizar cambios de emergencia

#### Git Stash:

¿Qué es *Git Stash*?

Es un área que permite almacenar temporalmente los cambios realizados en un proyecto sin enviarlos al repositorio mediante un commit.

¿Qué comando nos permite realizar un guardado rápido en nuestros archivos del proyecto?

`git stash` o `git save`

¿Con qué comando podemos visualizar de forma numerada los guardados rápidos almacenados?

`git stash list`

¿De qué manera podemos aplicar uno de los guardados rápidos almacenados mediante su nombre de lista?

`git stash apply stash@{2}`

¿Qué acción realiza el comando *git stash pop*?

Permite hacer entrada al guardado, aplicar los cambios y eliminar los archivos de dicha área.

¿Qué hace el comando *git stash show*?

Muestra un resumen de los cambios realizados en el stash.

¿Con qué comando podemos borrar los cambios guardados en stash sin aplicarlos?

`git stash drop nombre-del-stash`

¿De qué manera podemos limpiar todo en el área de stash?

`git stash clear`

¿Qué acción se ejecuta cuando añadimos la opción *--keep-index* al comando *stash save*?

Le indica a Git que no guarde nada que no se haya almacenado ya mediante el comando `git add`.

¿De qué forma se puede realizar un guardado rápido en el que se almacene cualquier archivo que no está bajo el control de la última versión del proyecto?

`git stash --include-untracked`

## **GIT Y EL USO DE COMANDOS.**

### **Git Rebase.**

#### **¿Qué es git rebase?**

Es un proceso de reorganización enfocado en integrar modificaciones de una rama a otra, organizando la base de una rama de un commit a otra, manteniendo un historial del proyecto lineal.

#### **¿Qué función realiza la opción squash en el comando git rebase?**

Permite combinar uno o más commits en uno solo.

#### **¿Cómo funciona la opción reword aplicada en git rebase?**

Después de su ejecución, la reorganización se detiene y permite modificar el mensaje de commit.

#### **¿Qué hace el comando git rebase edit?**

Permite modificar el commit, es decir, agregar o eliminar información de éste.

#### **¿Qué acción realiza la opción pick?**

Indica que se incluye la confirmación del commit.

#### **¿Cuál es la mayor desventaja de realizar un rebase?**

Hacer un rebase en una rama o repositorio público fusiona la rama principal en el repositorio local, creando una versión diferente o una discordancia en el proyecto.

#### **¿Cuál es la diferencia entre git merge y git rebase?**

Git merge deja intactas las ramas existentes conservando todo el historial, mientras que git rebase reorganiza nuestro proyecto consiguiendo un historial limpio y lineal sin bifurcaciones.

#### **¿Cuál es la diferencia entre usar la opción fixup y la opción squash?**

Squash permite combinar dos o más commits en uno solo con la posibilidad de escribir un nuevo mensaje de confirmación, fixup descarta este mensaje y sólo combina los commits en uno solo.

#### **¿Qué hace el comando git rebase --i?**

Accede a la versión interactiva del comando, en la cual podemos seleccionar, modificar, mover o eliminar los commits según las propias necesidades.

#### **¿Qué diferencia hay entre realizar un rebase standard y un rebase interactivo?**

El rebase standard toma automáticamente las confirmaciones de la rama de trabajo actual y las aplica al encabezado de la rama anterior, mientras que el rebase interactivo abre un editor en el cual se pueden adaptar los commits a nuestras necesidades.

## **GitHub avanzado.**

### **Fork, Clone y Pull Requests.**

#### **¿Cuál es la principal diferencia entre un fork y un clone en Git?**

**Un fork copia el repositorio en tu cuenta de GitHub, mientras que el clone copia el repositorio localmente**

#### **¿Qué comando se usa para vincular un repositorio clonado con el repositorio original (upstream)?**

**git remote add upstream <URL>**

#### **¿Cuál de las siguientes afirmaciones sobre pull requests es correcta?**

## **GIT Y EL USO DE COMANDOS.**

Un pull request permite discutir y revisar cambios antes de integrarlos al proyecto principal

**¿Cuál es el propósito principal de hacer un fork antes de colaborar en un proyecto de GitHub?**

Crear una copia del repositorio que no afecta el proyecto original

### **Actualización del Fork y flujos de trabajo**

**¿Qué comando se usa para obtener los últimos cambios del repositorio original (upstream) sin fusionarlos inmediatamente en tu fork?**

**git fetch upstream**

**¿Cuál es el propósito de git merge al actualizar un fork?**

Combinar los cambios obtenidos mediante git fetch en tu rama actual

**¿Cuál de los siguientes flujos de trabajo en GitHub se basa en mantener una rama principal estable y desarrollar en ramas separadas para cada característica?**

Feature Branch

**¿Qué estrategia es más adecuada para un equipo pequeño que trabaja en varios cambios simultáneamente, manteniendo control sobre versiones y lanzamientos?**

Gitflow

### **Feature Branch y Revisión del trabajo**

**¿Cuál es el propósito principal de usar una "feature branch" en Git?**

Implementar y probar una nueva característica de forma aislada del desarrollo principal

**¿Qué comando se utiliza para crear y cambiar a una nueva rama en Git?**

**git checkout -b <nombre\_rama>**

**¿Cuál de las siguientes afirmaciones sobre pull requests es correcta?**

Un pull request permite que otros revisen y comenten el código antes de fusionarlo

**¿Qué elemento es fundamental al revisar el código de un compañero?**

Proporcionar comentarios constructivos que mejoren el código

### **Limpieza y recuperación d4-.**

**¿Cuál es una razón importante para limpiar las ramas de un repositorio?**

Mejorar la organización y evitar confusión en el desarrollo

**¿Qué comando se utiliza para eliminar una rama local en Git?**

**git branch -d <nombre\_rama>**

**¿Cuál de los siguientes comandos permite recuperar una rama eliminada si todavía existe en el historial de Git?**

**git reflog**

**¿Qué ocurre si usas git branch -D <nombre\_rama> en lugar de git branch -d <nombre\_rama>?**

La rama se elimina forzosamente, sin importar su estado