



## Contenido

|   |    |
|---|----|
| 1. ¿QUÉ ES GIT?.....  | 2  |
| 2. INSTALACIÓN DE GIT EN WINDOWS.....                           | 2  |
| 3. CONFIGURACIÓN BÁSICA GIT.....                                | 3  |
| 4. FILOSOFÍA DEL TRABAJO CON GIT.....                           | 4  |
| 5. ESTADOS DE ARCHIVOS EN GIT.....                              | 4  |
| 6. ¿QUÉ ES GITHUB?.....   | 5  |
| 7. CREANDO UN REPOSITORIO DE GIT.....                           | 6  |
| 7.1. Inicializando un repositorio en una carpeta existente..... | 6  |
| 7.2. Clonando un repositorio existente.....                     | 6  |
| 8. REVISAR EL ESTADO DE ARCHIVOS.....                           | 8  |
| 9. RASTREAR ARCHIVOS NUEVOS.....                                | 8  |
| 10. PREPARAR ARCHIVOS MODIFICADOS.....                          | 9  |
| 11. ELIMINAR ARCHIVOS DEL AREA DE PREPARACIÓN.....              | 11 |
| 12. DESHACER CAMBIOS EN EL DIRECTORIO DE TRABAJO.....           | 12 |
| 13. CONFIRMAR CAMBIOS.....                                      | 13 |
| 14. ACTUALIZAR REPOSITORIO REMOTO.....                          | 14 |
| 15. ARCHIVO .gitignore.....                                     | 16 |
| 16. GESTIÓN DE RAMAS.....                                       | 18 |
| 16.1. Enviar la rama al repositorio remoto.....                 | 21 |
| 16.2. Fusionando una rama.....                                  | 22 |
| 16.3. Eliminar una rama.....                                    | 23 |
| 17. ANEXO I.....  | 24 |
| 18. ANEXO II.....   | 24 |
| 19. ANEXO III.....  | 27 |

## 1. ¿QUÉ ES GIT?

Git es un software de control de versiones distribuido desarrollado por Linus Torvalds, para optimizar el trabajo en proyectos que cuenten con un gran número de archivos.

Aunque, es posible utilizar **Git** como un **CVS local**, lo usual, es alojar los proyectos en un servidor para trabajar de forma colaborativa.

Existen un montón de **plataformas de desarrollo colaborativo** (también conocidas como **forjas**) que admiten el uso de **Git** como **sistema de control de versiones**. Entre estas destacan:

- GitLab
- Bitbucket
- Codegiant
- GitHub
- SourceForge

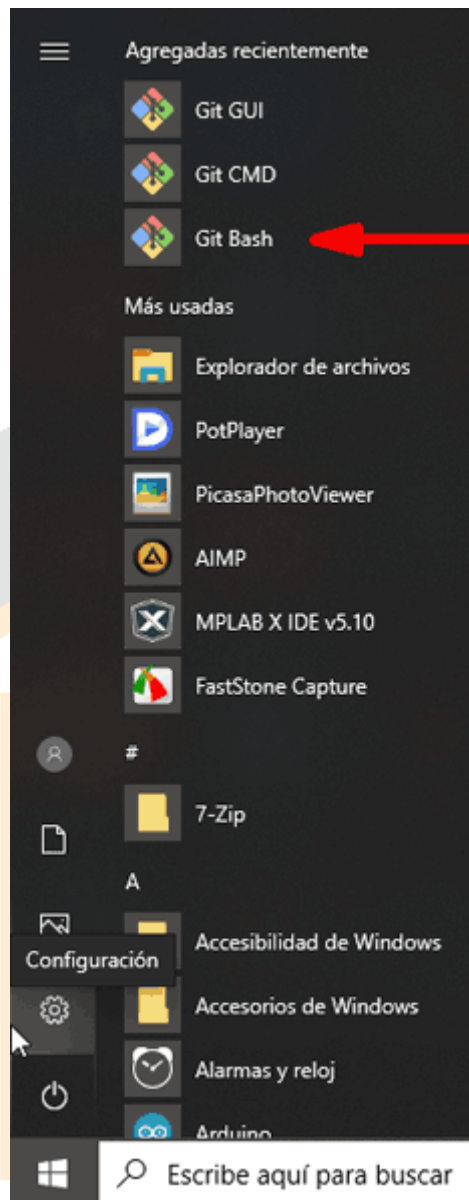
## 2. INSTALACIÓN DE GIT EN WINDOWS

Para **instalar Git en Windows** es necesario ir a la página de descargas para Windows en el sitio de Git. Aquí se debe seleccionar el que corresponda con la arquitectura del ordenador.



Una vez la descarga ha finalizado se ejecuta el instalador. La primera ventana que aparece muestra la licencia. Se pulsa en "Next >" para continuar y en las sucesivas pantallas se van dejando todas las opciones por defecto.

Para comprobar que todo se ha instalado correctamente puedes ir al menú inicio y comprobar que la aplicación aparezca allí.



### 3. CONFIGURACIÓN BÁSICA GIT

inmediatamente después de instalar Git, lo primero que hay que hacer es lanzar un par de comandos de configuración.

```
git config --global user.name "Tu nombre aquí"  
git config --global user.email "tu_email_aquí@example.com"
```

Con estos comandos indicas tu nombre de usuario (usas tu nombre y apellidos generalmente) y el *email*. Esta configuración sirve para que cuando hagas *commits* en el repositorio local, éstos se almacenen con la referencia a ti mismo, meramente informativa. Gracias a ello, más adelante cuando obtengas información de los cambios realizados en los archivos del "repo"

local, te va a aparecer como responsable de esos cambios a este usuario y correo que has indicado.

#### 4. FILOSOFÍA DEL TRABAJO CON GIT

En Git, al igual que otros **sistemas de control de versiones**, los proyectos se almacenan en repositorios. Un **repositorio** es el lugar donde se almacenan los archivos del proyecto junto a todo el historial de cambios que **Git** gestiona.

A efectos prácticos no es más que la carpeta en la que se van a almacenar los archivos que se quieren gestionar.

La mayoría de los **CVS** almacenan la información como una lista de cambios en los archivos, es decir, manejan la información que almacenan como un conjunto de archivos y las modificaciones hechas a cada uno de ellos a través del tiempo.

**Git**, por el contrario, **almacena instantáneas del proyecto cada vez que confirmas algún cambio**. Es como si, cada vez que confirmes un cambio, **Git** tome una fotografía de todos los archivos del proyecto y lo almacene tal cual.

#### 5. ESTADOS DE ARCHIVOS EN GIT

**Esto es lo más importante que se debes recordar.** En Git un archivo puede estar en uno de tres estados:

- **Confirmado** (*committed*): indica que los datos están almacenados de manera segura en tu base de datos local.
- **Modificado** (*modified*): indica que has modificado el archivo, pero todavía no lo has confirmado a tu base de datos.
- **Preparado** (*staged*): significa que has marcado un archivo modificado en su versión actual para que vaya en tu próxima confirmación.

Esto implica que en un proyecto de **Git** existan **tres secciones principales**:

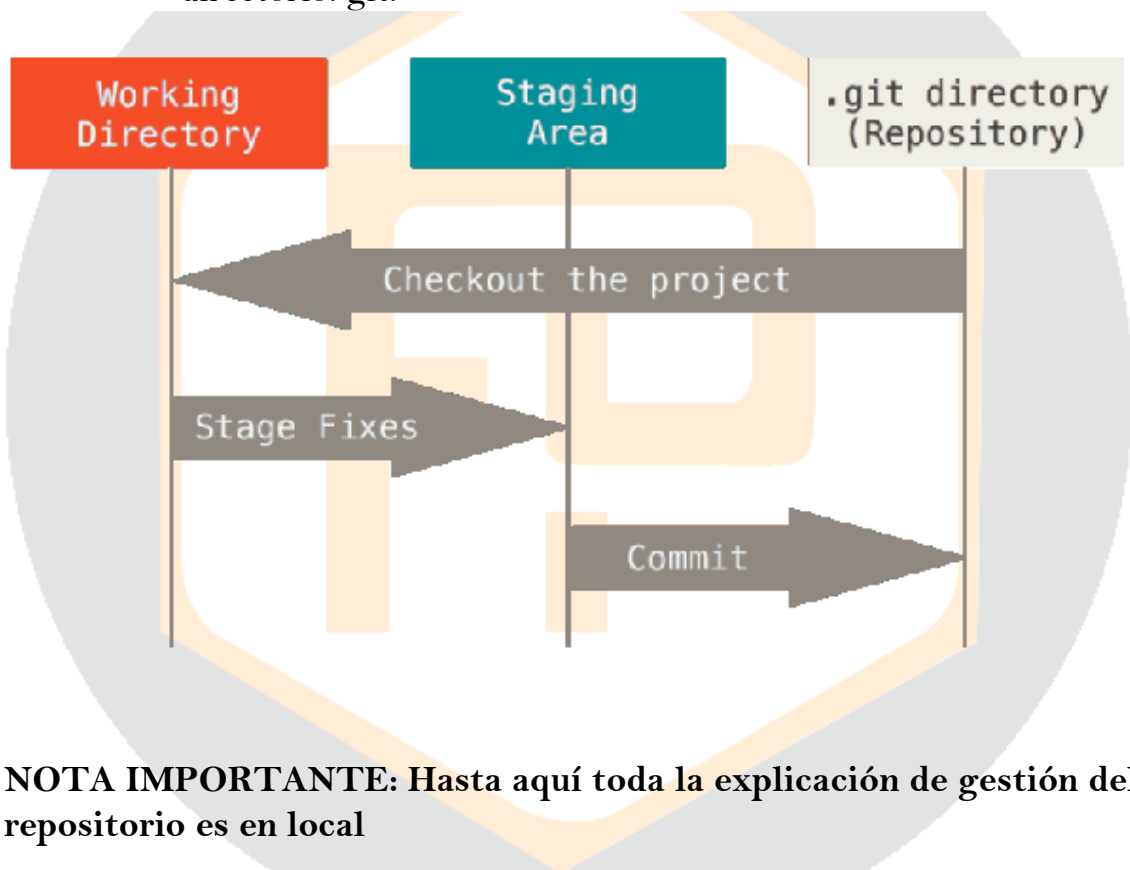
- **Directorio .git**: es donde se almacena la base de datos con toda la información que **Git** necesita para gestionar el proyecto.
- **Directorio de trabajo**: no es más que una copia de una versión del proyecto. Estos archivos son extraídos de la base de datos

del **directorio .git** y se colocan en la carpeta del proyecto para que puedas trabajar con ellos.

- **Área de preparación** (*Staging area*): en sí es un fichero que indica que los archivos que irán en la siguiente confirmación.

Teniendo esto en cuenta, el **flujo de trabajo en Git** para una confirmación sería:

1. **Modificar** una serie de archivos en el directorio de trabajo.
2. **Añadir** los archivos modificados al área de preparación.
3. **Confirmar** los cambios. Esto toma los archivos tal y como están en tu área de preparación y los almacena en la base de datos del directorio. git.



**NOTA IMPORTANTE:** Hasta aquí toda la explicación de gestión del repositorio es en local

## 6. ¿QUÉ ES GITHUB?

**GitHub** es una compañía sin fines de lucro que ofrece un servicio de *hosting* de repositorios almacenados en la nube. Esencialmente, es una **plataforma de desarrollo colaborativo** (también denominadas “forja”) para alojar y gestionar proyectos **utilizando el sistema de control de versiones Git**.

Se utiliza **principalmente para la creación de código fuente** de programas, pero no se limita solo a eso. De hecho, en esta plataforma te puedes encontrar proyectos de todo tipo, desde **libros** hasta **tutoriales**.

Esto se debe a su interfaz clara y limpia que permite a cualquier persona utilizarlo fácilmente.

Además de esto, cualquier persona puede inscribirse y hospedar un repositorio público completamente gratuito. Esto hace que **GitHub** sea **especialmente popular con proyectos de código abierto** (*open source*).

Pero no acaba ahí, la verdad es que actualmente **GitHub** ofrece rasgos similares a los de una **red social**:

- Permite **seguir proyectos y personas**.
- Soporte para **notificaciones**.
- Permite **crear una página propia para cada proyecto**.
- Etc.

## 7. CREANDO UN REPOSITORIO DE GIT

Existen dos maneras de obtener un **repositorio en Git**. La primera es tomar un proyecto o carpeta local existente y crear el repositorio a partir de aquí. La segunda es **clonar un repositorio** existente en otro servidor, como puede ser **GitHub o GitLab** en una carpeta local.

### 7.1. Inicializando un repositorio en una carpeta existente

Se debe ir al directorio del proyecto y ejecutar el siguiente comando:

*git init*

Esto crea una **sub-carpeta** llamada **.git**, la cual contiene la base de datos y los archivos necesarios para que **Git** sea capaz de gestionar el proyecto.

### 7.2. Clonando un repositorio existente

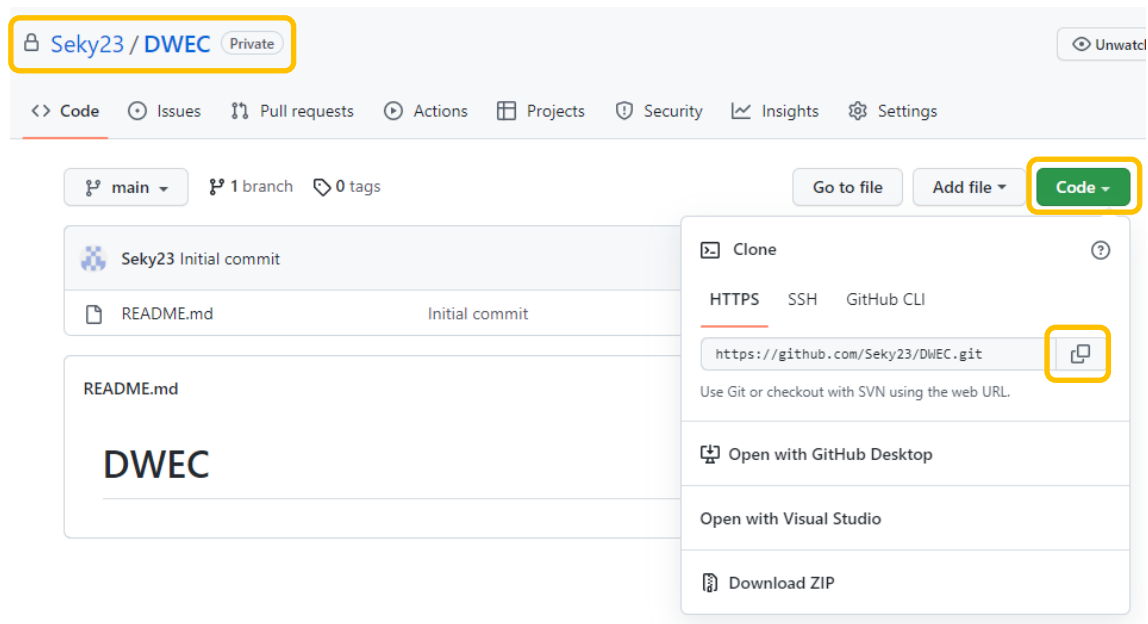
Si se desea obtener una copia de un **repositorio Git** existente en un **servidor remoto** se debe utilizar el comando **git clone**. Lo puedes utilizar para descargar un repositorio en el que desees contribuir o para comenzar a trabajar en un repositorio recién creado en **GitHub**.

Este comando **crea una copia local** del repositorio remoto, por lo tanto tendrás acceso a todo el historial de versiones de todos los archivos del proyecto.

La estructura del comando sería la siguiente:

*git clone [url]*

Donde *[url]* es la dirección del repositorio en el **servidor remoto**.



```
Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023
$ git clone https://github.com/Seky23/DWEC.git
```

```
Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023
$ git clone https://github.com/Seky23/DWEC.git
Cloning into 'DWEC'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.

Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023
$ ls
'1. SELECCIÓN DE ARQUITECTURAS Y HERRAMIENTAS DE PROGRAMACIÓN' / DWEC/

Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023
$ cd DWEC

Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/DWEC (main)
$ ls -la
total 5
drwxr-xr-x 1 Sergio 197121 0 Jul 28 22:36 ./
drwxr-xr-x 1 Sergio 197121 0 Jul 28 22:36 ../
drwxr-xr-x 1 Sergio 197121 0 Jul 28 22:36 .git/
-rw-r--r-- 1 Sergio 197121 6 Jul 28 22:36 README.md
```



Como puedes apreciar, la carpeta contiene un directorio **.git** (donde está la **base de datos del repositorio**) y el área de trabajo con los archivos de la última versión del proyecto.

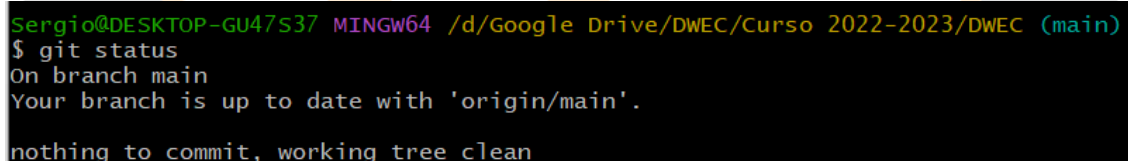
**Importante:** el **área de trabajo de un repositorio** es la propia carpeta donde se almacena el proyecto, sin incluir el directorio **.git**

## 8. REVISAR EL ESTADO DE ARCHIVOS

Los archivos en **Git** pueden o no estar bajo seguimiento. Los archivos **bajo seguimiento o rastreados** (en inglés *tracked*) son aquellos que estaban en la última instantánea del proyecto (commit). Los archivos **sin seguimiento o sin rastrear** (en inglés *untracked*) son todos los otros, es decir, cualquier archivo nuevo que se agregue al directorio de trabajo.

Cuando clonas un repositorio todos los archivos están rastreados y sin modificar. Al editar algunos archivos estos pasan al estado de **modificado**. Cuando termines las modificaciones tienes que poner los archivos en **modo preparado para confirmar los cambios**.

Para **determinar en qué estado se encuentran los archivos** de tu proyecto debes ejecutar el comando **git status**. Si ejecutas este comando en un repositorio recién clonado obtendrás un resultado similar al de la siguiente imagen.



```
Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/DWEC (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

Esto significa que no hay archivos **modificados** y tampoco **sin rastrear**. Esto es perfectamente lógico en un repositorio recién clonado en el que no se ha modificado nada.

También indica en qué rama te encuentras y el **estado de esa rama** con respecto a la existente en el servidor remoto. Como aún no has realizado modificaciones, muestra que está actualizada.

## 9. RASTREAR ARCHIVOS NUEVOS

Ahora se va a agregar un nuevo archivo de texto plano llamado **"hola\_github.txt"** a tu proyecto. La carpeta del proyecto ahora luce como esta imagen.



```

Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/DWEC (main)
$ touch hola_github.txt

Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/DWEC (main)
$ ls -a
./ ../ .git/ README.md hola_github.txt

Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/DWEC (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

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

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

```

Como se puede ver, el archivo `hola_github.txt` aparece bajo la cabecera “Archivos sin seguimiento” (en inglés *Untracked files*). Bajo esta cabecera solo aparecen archivos nuevos o modificados y Git no los incluirá en el próximo *commit* a no ser que se le indique expresamente.

Para que Git comience a rastrear un nuevo archivo es necesario utilizar el comando `git add`. Por ejemplo, ejecutando el siguiente comando se comenzará a rastrear el archivo `hola_github.txt`

```

Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/DWEC (main)
$ git add hola_github.txt

Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/DWEC (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   hola_github.txt

```

Si vuelves a revisar el estado de los archivos. Puedes ver que ahora el archivo `hola_github.txt` está siendo rastreado y está listo para ser confirmado en el próximo *commit* (*stage*).

## 10. PREPARAR ARCHIVOS MODIFICADOS

Ahora se va a editar un archivo que esté siendo rastreado, en este caso el archivo `README.md`.

```
MINGW64:/d/Google Drive/DWEC/Curso 2022-2023/DWEC
GNU nano 6.2 README.md Modified
# Repositorio utilizado para el módulo DWEC

^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute   ^C Location
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^_ Go To Line

Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/DWEC (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   hola_github.txt

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:   README.md
```

Como se aprecia, el archivo **README.md** aparece en la sección “**Cambios no rastreados para el *commit***” (en inglés, *Changes not staged for commit*). Eso significa que es un archivo rastreado que ha sido modificado, pero aún no está preparado. Para ello, es necesario utilizar el comando **git add**. (Con los siguientes ejemplos se rastrean todos los archivos que estén sin preparar)

`git add .`

`git add -A`

Si después ejecutas **git status** el resultado sería:

```
Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/DWEC (main)
$ git add .
warning: LF will be replaced by CRLF in README.md.
The file will have its original line endings in your working directory

Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/DWEC (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README.md
        new file:   hola_github.txt
```

Esto indica que ambos archivos están preparados y formarán parte de tu **próxima confirmación** (*commit*).

**git add** es un comando que cumple varios propósitos. Es preferible pensar en él como un comando para “**añadir este contenido a la próxima confirmación**” más que para “añadir este archivo al proyecto”.

## 11. ELIMINAR ARCHIVOS DEL AREA DE PREPARACIÓN

Para eliminar archivos del área de preparación de manera individual, escribe lo siguiente (por ejemplo):

```
git reset README.md
```

```
git restore --staged README.md
```

Esto eliminará el archivo README.md del área de preparación. Para ver este cambio, escribe nuevamente el comando `git status`.

Si deseas eliminar todos los archivos del área de preparación, entonces ejecuta lo siguiente:

```
git reset
```

```
Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README.md
        new file:   hola_github.txt

Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (main)
$ git reset README.md
Unstaged changes after reset:
M       README.md

Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   hola_github.txt

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:   README.md
```

## 12. DESHACER CAMBIOS EN EL DIRECTORIO DE TRABAJO

Para que un archivo marcado como modificado en el directorio de trabajo vuelva a como se encontraba antes de ser marcado como modificado hay que ejecutar

```
git restore README.md
```

En la siguiente imagen se ve el contenido antes y después de su restauración

```
Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   hola_github.txt

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:   README.md

Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (main)
$ cat README.md
# Repositorio utilizado para el módulo DWEC

Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (main)
$ git restore README.md

Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   hola_github.txt

Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (main)
$ cat README.md
# DWEC
```

### 13. CONFIRMAR CAMBIOS

Una vez tienes el área de preparación con los archivos que quieres, puedes confirmar los cambios. Para esto es necesario ejecutar el comando:

*git commit -m "mensaje"*

“-m” especifica un mensaje que se debe pasar describiendo la confirmación. Dado que este es nuestro primer commit, escribiremos Initial Commit.

```
Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   hola_github.txt

Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (main)
$ git commit -m "Initial Commit"
[main c75b0f0] Initial Commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 hola_github.txt
```

Si ahora ejecutas `git status`, verás que se indica que el directorio de trabajo está limpio ya que se han confirmado todos los archivos y no se ha modificado ninguno desde entonces.

luego podemos ver el commit que habíamos ejecutado, incluyendo el número hash del commit.

*git log*

```
Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (main)
$ git commit -m "Initial Commit"
[main c75b0f0] Initial Commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 hola_github.txt

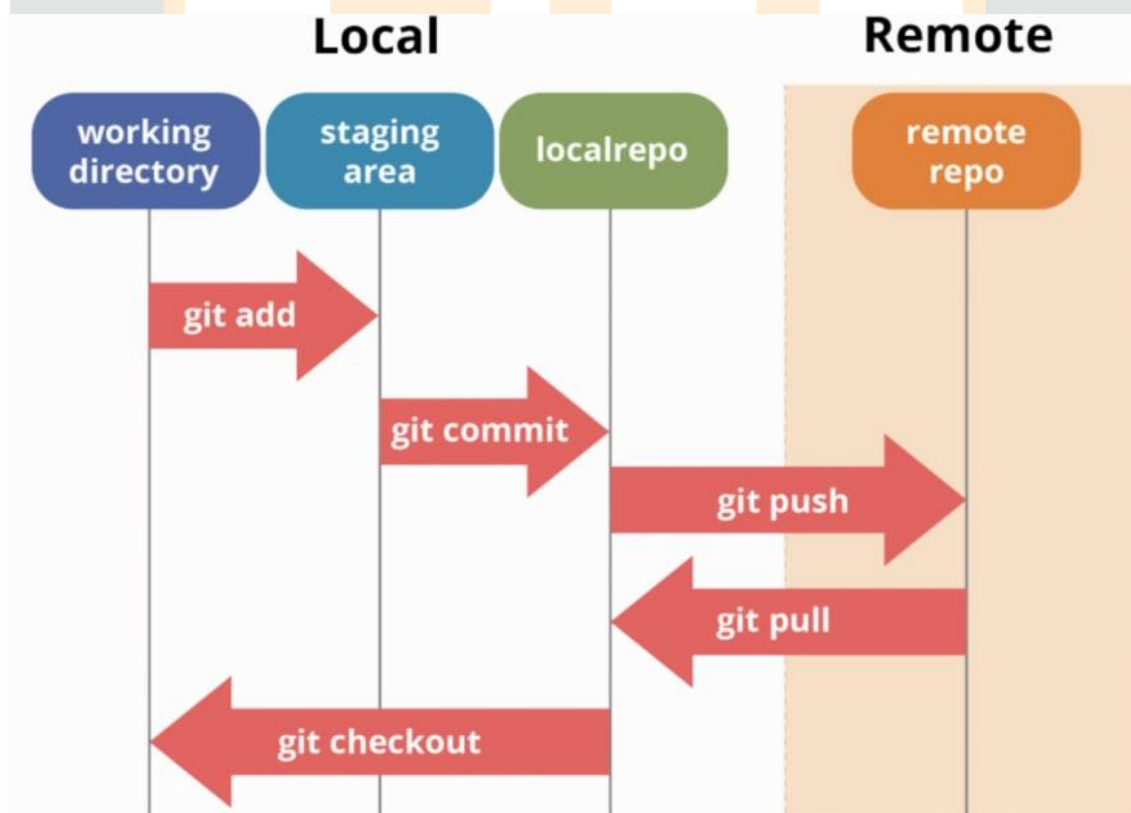
Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (main)
$ git log
commit c75b0f0348ced1779bd321946fb89b7c5150a79e (HEAD -> main)
Author: Seky23 <seky23@hotmail.com>
Date: Tue Aug 2 13:30:22 2022 +0200

    Initial Commit

commit 9506ec940b279a8a0aaf576973822fcae0a1614e (origin/main, origin/HEAD)
Author: Seky23 <34790383+Seky23@users.noreply.github.com>
Date: Thu Jul 28 22:26:46 2022 +0200

    Initial commit
```

#### 14. ACTUALIZAR REPOSITORIO REMOTO



hasta ahora todos los cambios realizados han sido locales. Es decir, que todas las modificaciones están en tu ordenador. De hecho, si ejecutas el comando **git status** obtendrás una salida que indica que tu rama está adelantada a la de **origin** por **varios commits**.

```
Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

El término **origin** lo utiliza **Git** para referirse al **servidor remoto** del que **has clonado tu repositorio**.

Ahora debemos enviar los cambios confirmados al repositorio remoto para que otras personas tengan acceso a ellos.

Dado que lo común es que hay varios desarrolladores trabajando en un solo proyecto, **primero tenemos que extraer cualquier cambio que se haya realizado en el repositorio remoto antes de enviar nuestros cambios** para evitar conflictos.

Ejecuta el siguiente comando:

*git pull origin main*  
*git pull https://github.com/Seky23/DWEC.git main*

```
Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (main)
$ git pull origin main
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 692 bytes | 34.00 KiB/s, done.
From https://github.com/Seky23/DWEC
* branch      main      -> FETCH_HEAD
   9506ec9..5db5290  main  -> origin/main
Merge made by the 'ort' strategy.
 repo.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 repo.txt
```

El comando para enviar todos los cambios locales a un **servidor remoto** es **git push**, su estructura sería:

*git push [nombre-remoto] [nombre-rama]*

Donde:



- `[nombre-remoto]`: es el servidor a actualizar.
  - `[nombre-rama]`: es la rama a actualizar.
- Para el **nombre del servidor** puedes utilizar **directamente la URL** o el término **origin**. Por lo tanto, puedes utilizar cualquiera de estos comandos:

```
git push https://github.com/Seky23/DWEC.git main
git push origin main
```

- Al ejecutar el **comando Git** te pedirá que indiques el nombre de usuario y la contraseña para acceder al **servidor remoto**. Esos datos corresponden a los que utilizaste para crear tu **cuenta de GitHub**.
- El resultado obtenido sería el siguiente:

```
Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (main)
$ git push https://github.com/Seky23/DWEC.git main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 282 bytes | 282.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Seky23/DWEC.git
 9506ec9..c75b0f0  main -> main

Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (main)
```

Si ahora vas al navegador y abres el repositorio podrás ver los cambios efectuados en el **repositorio remoto**

## 15. ARCHIVO .gitignore

el archivo "gitignore", que sirve para decirle a Git qué archivos o directorios completos debe ignorar y no subir al repositorio de código.

En el gitignore se especificarán todas las rutas y archivos que no se requieren y con ello, el proceso de control de versiones simplemente ignorará esos archivos.

Piensa que no todos los archivos y carpetas son necesarios de gestionar a partir del sistema de control de versiones. Hay código que no necesitas enviar a Git, ya sea porque sea privado para un desarrollador en concreto y

no lo necesiten (o lo deban) conocer el resto de las personas. Pueden ser también archivos binarios con datos que no necesitas mantener en el control de versiones, como diagramas, instaladores de software, etc.

implemente tienes que crear un archivo que se llama ".gitignore" en la carpeta raíz de tu proyecto. Como puedes observar, es un archivo oculto, ya que comienza por un punto ".".

**Nota:** Los archivos cuyo nombre comienza en punto "." son ocultos solamente en Linux y Mac. En Windows los podrás ver perfectamente con el explorador de archivos.

Dentro del archivo .gitignore colocarás texto plano, con todas las carpetas que quieres que Git simplemente ignore, así como los archivos.

La notación es muy simple. Por ejemplo, si indicamos la línea

```
bower_components/
```

Estamos evitando que se procese en el control de versiones todo el contenido de la carpeta "bower\_components".

Si colocamos la siguiente línea:

```
*.DS_Store
```

Estaremos evitando que el sistema de control de versiones procese todos los archivos acabados de .DS\_Store, que son ficheros de esos que crea el sistema operativo del Mac (OS X) automáticamente.

Hay muchos tipos de patrones aplicables a la hora de especificar grupos de ficheros, con comodines diversos, que puedes usar para poder indicar, de manera muy específica, lo que quieres que Git no procese al realizar el control de versiones. Puedes encontrar más información en la [documentación de gitignore](#), pero generalmente no lo necesitarás porque lo más cómodo es crear el código de este archivo por medio de unas plantillas que ahora te explicaremos.

Dado que la mayoría de las veces los archivos que necesitas ignorar son siempre los mismos, atendiendo a tu sistema operativo, lenguajes y tecnologías que uses para desarrollar, es muy sencillo crear un archivo .gitignore por medio de una especie de plantillas.

Existe una herramienta online que yo uso siempre que se llama [gitignore.io](https://gitignore.io). Básicamente permite escribir en un campo de búsqueda los nombres de todas las herramientas, sistemas, frameworks, lenguajes, etc. que puedas

estar usando. Seleccionas todos los valores y luego generas el archivo de manera automática.

Por ejemplo una alternativa sería escribir las siguientes palabras clave: OSX, Windows, Node, Polymer, SublimeText.

Escribes cada una de las palabras y pulsas la tecla enter para ir creando los "chips". Te debe aparecer algo como esto:



Una vez generado el código de tu gitignore, ya solo te queda copiarlo y pegarlo en tu archivo .gitignore, en la raíz de tu proyecto.

**Nota:** Las líneas que comienzan por "#" en un .gitignore son simplemente comentarios.

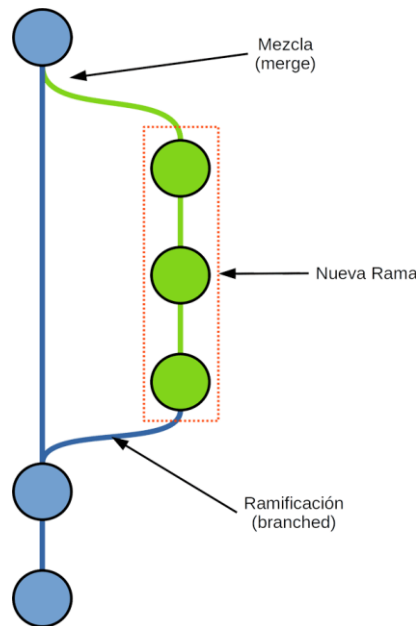
## 16. GESTIÓN DE RAMAS

Ahora, imagina que quieres agregar una nueva funcionalidad, pero no estás del todo seguro. Bien podrías continuar como hasta ahora y si después no te gusta el resultado regresar a una de las “**instantáneas**” anteriores.

Pero, existe un método más práctico y es el de **ramificar el proyecto**.

La idea es tomar el estado actual de la rama *main* y crear una nueva rama a partir de ese estado. De esta forma puedes hacer cambios en esta nueva rama y luego, si quieres, puedes combinarla con la rama *main*.

Al proceso de combinar dos ramas se le denomina “**mezclar**” (*merge* en inglés). Este proceso se encarga de hacer efectivos los cambios realizados en una rama sobre la otra.



Hasta ahora hemos estado trabajando en nuestra rama maestra o principal, pero no es así como deberías trabajar en git como desarrollador porque la rama maestra debe ser una versión estable del proyecto en el que estás trabajando. Lo usual es que cada colaborador cree una rama cuando va a realizar alguna modificación.

Luego, cuando sus cambios son comprobados (es decir, cuando sean validados mediante pruebas) se mezclan los resultados.

El comando para crear una nueva rama llamada solucion-alfa es el siguiente:

```
git branch solucion-alfa
```

Ahora si ejecutas

```
git branch
```

luego verás todas las ramas del repositorio, y **la rama en la que tú estás ubicado se encuentra resaltada con un asterisco del lado izquierdo**

```
Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (main)
$ git branch solucion-alfa

Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (main)
$ git branch
* main
  solucion-alfa
```

Si deseas cambiarte a la rama recientemente creada por ti, escribe lo siguiente:

*git checkout solucion-alfa*

Ahora, si escribes git branch verás que ahora te encuentras en la rama *solucion-alfa*.

```
Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (main)
$ git checkout solucion-alfa
Switched to branch 'solucion-alfa'

Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (solucion-alfa)
$ git branch
  main
* solucion-alfa

Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (solucion-alfa)
$ ls
README.md  actualizar_github.txt  hola_github.txt  repo.txt
```

Ahora debemos realizar los cambios en el proyecto. Creamos el fichero *archivo-alfa*.

```
Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (solucion-alfa)
$ touch archivo-alfa

Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (solucion-alfa)
$ ls
README.md  actualizar_github.txt  archivo-alfa  hola_github.txt  repo.txt

Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (solucion-alfa)
$ git status
On branch solucion-alfa
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    archivo-alfa

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

Ahora repetimos el proceso para confirmar estos cambios:

*git status*  
*git add -A*  
*git commit -m "solución-alfa"*

```
Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (solucion-alfa)
$ git status
On branch solucion-alfa
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    archivo-alfa

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

Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (solucion-alfa)
$ git add -A

Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (solucion-alfa)
$ git commit -m "solucion alfa"
[solucion-alfa 6a078c2] solucion alfa
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 archivo-alfa
```

Este commit solo cambiará los archivos en la rama solución-alfa local, no habiendo alterado aún la rama main local ni el repositorio remoto.

### 16.1. Enviar la rama al repositorio remoto

Ingresa el siguiente comando:

```
git push -u origin solución-alfa
```

donde origin es el repositorio remoto y solución-alfa es la rama que le queremos enviar.

```
Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (solucion-alfa)
$ git push -u origin solucion-alfa
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 236 bytes | 236.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'solucion-alfa' on GitHub by visiting:
remote:   https://github.com/Seky23/DWEC/pull/new/solucion-alfa
remote:
To https://github.com/Seky23/DWEC.git
 * [new branch]      solucion-alfa -> solucion-alfa
branch 'solucion-alfa' set up to track 'origin/solucion-alfa'.
```

Ahora en el navegador podrás ver que en el repositorio remoto hay una rama mas, y si escribes

```
git branch -a
```

```
Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (solucion-alfa)
$ git branch -a
main
* solucion-alfa
remotes/origin/HEAD -> origin/main
remotes/origin/main
remotes/origin/solucion-alfa
```

Ahora vemos que en nuestro repositorio remoto tenemos la rama solución-alfa **¿Porqué debemos enviar la rama al repositorio remoto? Porque en algunas empresas es allí donde ejecutan sus pruebas unitarias y en otras para asegurarse de que el código se ejecute bien antes de fusionarse con la rama maestra.**

Dado que todas la prueban ha sido exitosas (no entraremos en detalles de eso aquí), ahora podemos fusionar la rama solución-alfa con la rama principal.

## 16.2. Fusionando una rama

Primero, debemos ubicarnos (checkout) en la rama maestra local

*git checkout main*

```
Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (solucion-alfa)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (main)
$
```

Extraemos todos los cambios de la rama maestra remota:

*git pull origin main*

```
Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (main)
$ git pull origin main
From https://github.com/Seky23/DWEC
* branch      main      -> FETCH_HEAD
Already up to date.
```

Ahora veremos todas las ramas que hemos fusionado hasta ahora:

*git branch --merged*

```
Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (main)
$ git branch --merged
* main
```

la rama solucion-alfa no figurará ya que aún no la hemos fusionado.

Para fusionar solucion-alfa con la principal:

*git merge solucion-alfa*

(Ten en cuenta que ahora estamos en la rama maestra)

```
Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (main)
$ git merge solucion-alfa
Updating e554a7b..6a078c2
Fast-forward
 archivo-alfa | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 archivo-alfa

Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (main)
$ git branch --merged
* main
  solucion-alfa
```



Ahora que ha sido fusionada, podemos enviar los cambios a la rama maestra del repositorio remoto.

*git push origin main*

```
Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (main)
$ git push origin main
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Seky23/DWEC.git
e554a7b..6a078c2  main -> main
```

### 16.3. Eliminar una rama

Para verificar la fusión realizada en la sección anterior, podemos ejecutar:

*git branch --merged*

```
Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (main)
$ git branch --merged
* main
  solucion-alfa
```

Si solución-alfa se muestra aquí, eso significa que hemos fusionado todos los cambios y que la rama ya puede ser descartada.

*git branch -d solución-alfa*

```
Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (main)
$ git branch -d solucion-alfa
Deleted branch solucion-alfa (was 6a078c2).
```

Ahora la rama ha sido eliminada localmente.

Pero como la hemos enviado al repositorio remoto, aún continua ahí. Esto puede ser visto ejecutando:

*git branch -a*

```
Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (main)
$ git branch -a
* main
  remotes/origin/HEAD -> origin/main
  remotes/origin/main
  remotes/origin/solucion-alfa
```

Para eliminar la rama del repositorio remoto, escribe:

*git push origin --delete solucion-alfa*

```
Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (main)
$ git push origin --delete solucion-alfa
To https://github.com/Seky23/DWEC.git
- [deleted]          solucion-alfa

Sergio@DESKTOP-GU47S37 MINGW64 /d/Google Drive/DWEC/Curso 2022-2023/dwec (main)
$ git branch -a
* main
  remotes/origin/HEAD -> origin/main
  remotes/origin/main
```

## 17. ANEXO I

1. Instalar Git LFS

<https://git-lfs.github.com/>

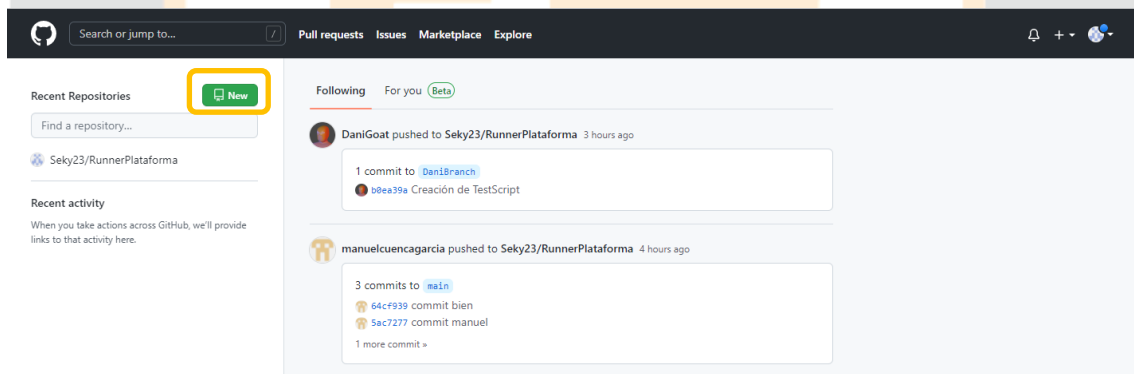
2. Abrir la consola de git y ejecutar

git lfs install

## 18. ANEXO II

3. Crear un repositorio en github

<https://github.com/>



## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner \*

Seky23 ▾

Repository name \*

Great repository names are short and memorable. Need inspiration? How about [crispy-happiness?](#)

Description (optional)

☐

Public

Anyone on the internet can see this repository. You choose who can commit.

☒

Private

You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

☐

Add a README file

This is where you can write a long description for your project. [Learn more.](#)

Add .gitignore

Choose which files not to track from a list of templates. [Learn more.](#)

.gitignore template: Unity ▾

Choose a license

A license tells others what they can and can't do with your code. [Learn more.](#)

License: None ▾



#### 4. Clonamos el repositorio en el lugar del equipo windows donde se desea crear el proyecto unity

The screenshot shows a GitHub repository page for 'Seky23 / DWEC'. The repository is private and has 1 commit. The main branch is selected. The repository contains a README.md file. The page shows the repository name, owner, and various statistics like stars, forks, and watchers. The README content is visible, showing the title 'DWEC'. The page also includes navigation links for Code, Issues, Pull requests, Actions, Projects, Security, Insights, and Settings. The footer shows the URL: https://github.com/Seky23/DWEC/settings.

GitHub interface showing the repository **Seky23 / RunnerPlataforma** (Private).

Navigation tabs: <> Code, Issues, Pull requests 1, Actions, Projects, Security, Insights, Settings.

Repository details: main branch, 5 branches, 0 tags. Buttons: Go to file, Add file, Code.

File list:

| File            | Commit                     |
|-----------------|----------------------------|
| Assets          | Commit Variable            |
| Packages        | First Commit               |
| ProjectSettings | First Commit               |
| UserSettings    | First Commit               |
| .gitattributes  | Commit desde Visual Studio |
| .gitignore      | Initial commit             |
| .vsconfig       | Commit desde Visual Studio |
| LICENSE         | Initial commit             |
| a.txt           | commit bien                |
| prueba.txt      | Creación fichero de prueba |

Clone menu options:

- Clone (HTTPS, SSH, GitHub CLI)
- Open with GitHub Desktop
- Open with Visual Studio
- Download ZIP

Clone URL: `https://github.com/Seky23/RunnerPlataforma`

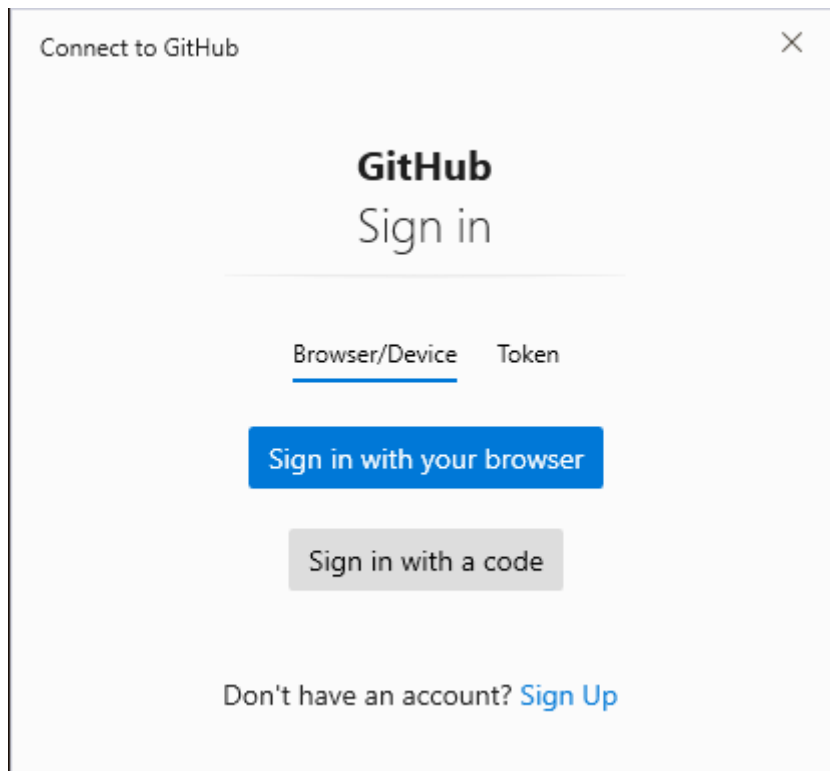
```
MINGW64:/c/Users/Sergio/Documents/Unity/colaborativo
$ ls
EjemploSolucion/
EjemploSolucionTareaLevel2.unitypackage/
EjemploSolucionTareaLevel2.unitypackage.gz
'Examen Metroid'/
'Juego Bolos'/
LittleNightmareBeforeChristmas/
MiPrimerVideojuego/
MyProject/
'Oculus rift'/
'PaintBall 3D'/
'PaintBall 3D antes de la solución final'/
'Pixel Metroid 2D Depurado'/
'Pixel Metroid 2D Disparo'/
PixelMetroid2D/
'PixelMetroid2D Diseño Nivel 1'/
'PixelMetroid2D Final Videos'/
'Runner Plataforma Colaborativo'/
RunnerPlataforma/
RunnerPlataformaMovil/
ejercicioRealidadAumentada/
localGlobal/
'prueba bolos'/

Sergio@DESKTOP-GU47S37 MINGW64 ~/Documents/Unity
$ mkdir colaborativo

Sergio@DESKTOP-GU47S37 MINGW64 ~/Documents/Unity
$ cd colaborativo/

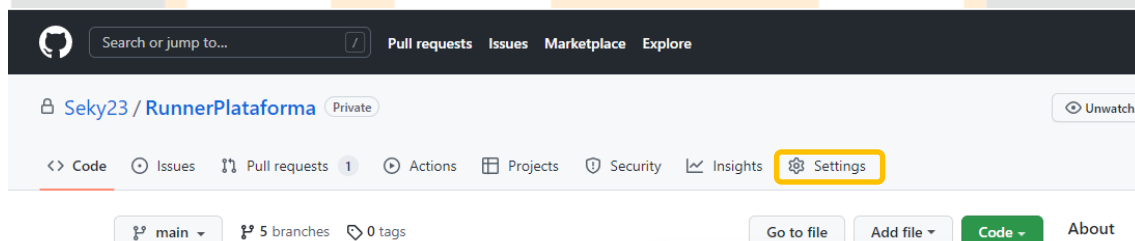
Sergio@DESKTOP-GU47S37 MINGW64 ~/Documents/Unity/colaborativo
$ ls

Sergio@DESKTOP-GU47S37 MINGW64 ~/Documents/Unity/colaborativo
$ git clone https://github.com/Seky23/RunnerPlataforma.git
```



## 19. ANEXO III

Compartimos el repositorio con el resto de usuarios que trabajarán con él



- General
- Access
- Collaborators
- Code and automation
- Branches
- Tags
- Actions
- Webhooks
- Pages
- Security
- Code security and analysis
- Deploy keys
- Secrets
- Integrations

Who has access

PRIVATE REPOSITORY

Only those with access to this repository can view it.

Manage

DIRECT ACCESS

6 have access to this repository. 6 collaborators.

Manage access


Add people

Select all

Type

Find a collaborator...


☐

 cgm94

Collaborator

Remove

☐

 christianpgo

Collaborator

Remove

