

## Git y Github

### Índice

Git y Github	1
Git	3
Instalación de GIT	3
¿Qué versión tengo de Git?	3
Creando un repositorio local	3
Repositorio Local	4
Configurando nuestro repositorio	4
Agregando archivos al repositorio	5
Confirmando archivos	6
Como guarda Git nuestros datos	7
Repasando	7
GitHub	8
Crear un repositorio remoto	8
Vinculando el repositorio remoto con el local	11
Creando un token en Github	11
Subiendo archivos a Github	15
Clonar un repositorio existente	16
Actualizar cambios en mi repositorio local	18
Ramas o Branches	19
Creando Ramas	19

Git checkout	20
Guardar cambios y subirlos al repositorio remoto	20
Fusión de dos ramas	20
Comandos Básicos	21
Otros Comandos interesantes	22

# Git

Git es un Software de control de versiones diseñado por [Linus Torvalds](#). Y, entonces ¿a qué le llamamos sistema de control de versiones?

Un Sistema de Control de versiones (VCS por sus siglas en inglés - Version Control System) es una herramienta de software que monitorea y gestiona cambios en un sistema de archivos. Asimismo, un VCS ofrece herramientas de colaboración para compartir e integrar dichos cambios en otros usuarios del VCS.

Como un VCS opera a nivel de sistema de archivos, esté monitoreará las acciones de adición, eliminación y modificación que se producen en archivos y directorios no vacíos.

Git es uno de los VCS que existen (otros son: Mercurial, SVN y Preforce), sin embargo es el más utilizado en el ambiente de desarrollo de software.

## Instalación de GIT

¿Cómo saber si Git está instalado? Si estamos usando una Mac o tenemos un sistema operativo basado en Linux, lo más probable es que ya esté instalado. En caso de tener Windows, debemos instalar Git Bash.

Podemos descargar la versión del sistema operativo desde este link <https://git-scm.com/downloads>

### ¿Qué versión tengo de Git?

Para ello utilizamos el comando `git --version`. Si este comando nos genera algún error es que no tenemos instalado Git.

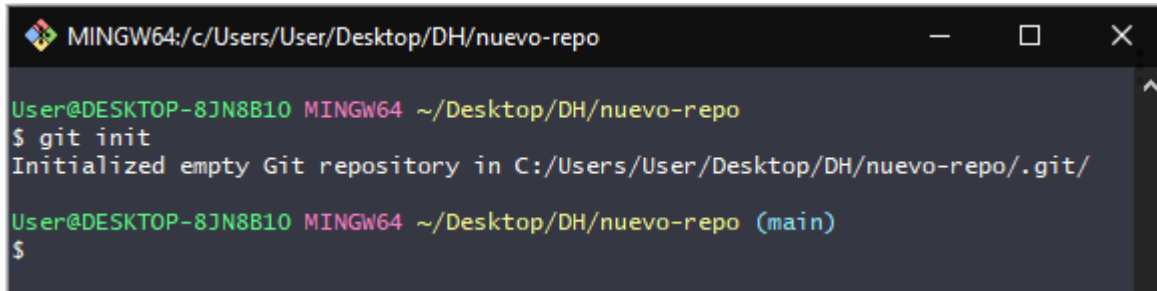
## Creando un repositorio local

Como dijimos anteriormente, Git, como VCS, monitoreará las acciones de adición, eliminación y modificación en archivos. ¿Cuáles archivos?. Monitoreará todos los archivos y directorios no vacíos dentro del directorio que le especifiquemos. Ese directorio contendrá nuestro repositorio de trabajo (también llamado working directory).

## Repositorio Local

Repositorio será entonces un conjunto de directorios y archivos donde se almacenará un determinado proyecto. Cada repositorio guarda un historial de versiones que nos permitirá volver en el tiempo, a un punto determinado de nuestro proyecto.

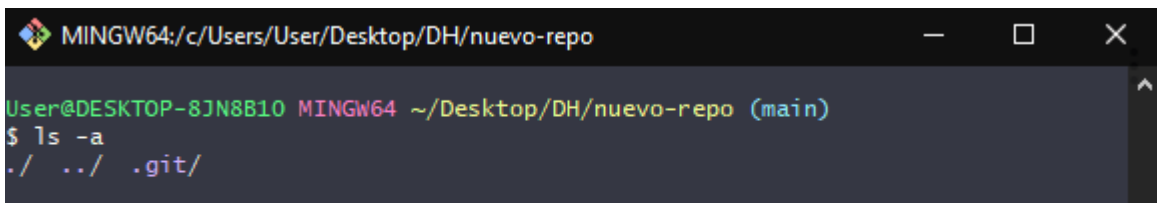
Para crear un repositorio desde cero, podemos ejecutar el comando `git init`

A screenshot of a Windows terminal window titled 'MINGW64:/c/Users/User/Desktop/DH/nuevo-repo'. The prompt is 'User@DESKTOP-8JN8B10 MINGW64 ~/Desktop/DH/nuevo-repo'. The command '\$ git init' has been entered, and the output is 'Initialized empty Git repository in C:/Users/User/Desktop/DH/nuevo-repo/.git/'. The prompt now shows '(main)' after the directory path.

```
MINGW64:/c/Users/User/Desktop/DH/nuevo-repo
User@DESKTOP-8JN8B10 MINGW64 ~/Desktop/DH/nuevo-repo
$ git init
Initialized empty Git repository in C:/Users/User/Desktop/DH/nuevo-repo/.git/
User@DESKTOP-8JN8B10 MINGW64 ~/Desktop/DH/nuevo-repo (main)
$
```

Podemos comprobar que nuestro repositorio ya está activo, viendo el nombre de la rama principal (main) o (master). Si estamos en Linux o Mac, tal vez no sea visible.

En todo caso, con el comando `ls -a` podemos listar los archivos ocultos de este directorio y allí nos debería aparecer la carpeta `.git`

A screenshot of a Windows terminal window titled 'MINGW64:/c/Users/User/Desktop/DH/nuevo-repo'. The prompt is 'User@DESKTOP-8JN8B10 MINGW64 ~/Desktop/DH/nuevo-repo (main)'. The command '\$ ls -a' has been entered, and the output is './ ../.git/'.

```
MINGW64:/c/Users/User/Desktop/DH/nuevo-repo
User@DESKTOP-8JN8B10 MINGW64 ~/Desktop/DH/nuevo-repo (main)
$ ls -a
./ ../.git/
```

## Configurando nuestro repositorio

Una vez que ya tenemos inicializado el repositorio, antes de crear carpetas y archivos es aconsejable configurar nuestro usuario y mail para que git nos reconozca como autores de las versiones del proyecto (este paso se puede realizar también una vez que ya tengamos archivos y directorios creados).

Para ello utilizaremos los comandos:

```
git config user.name "nombreUsuario"
```

```
git config user.email "nombreUsuario@gmail.com"
```

Es recomendable que si ya tenemos una cuenta de github (o gitlab o bitbucket), configuremos el repositorio con el nombre de usuario y contraseña correspondiente.

Si queremos que ese usuario y contraseña quede predeterminado para todos los repositorios que voy a crear, utilizaremos el flag `--global`:


```
git config --global user.name "nombreUsuario"
```

```
git config --global user.email "nombreUsuario@gmail.com"
```

## Agregando archivos al repositorio

Una vez que creamos archivos y/o directorios en nuestro repositorio, el proceso de seguimiento comienza. Git necesita que le digamos qué archivos queremos guardar en nuestra primera versión. Por lo que los archivos creados estarán sin seguimiento hasta que especifiquemos lo contrario.

Podemos ver el status de nuestro repositorio con el comando `git status`:



```
MINGW64:/c/Users/User/Desktop/DH/nuevo-repo
User@DESKTOP-8JN8B10 MINGW64 ~/Desktop/DH/nuevo-repo (main)
$ git status
On branch main

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)
User@DESKTOP-8JN8B10 MINGW64 ~/Desktop/DH/nuevo-repo (main)
$ |
```

Como podemos apreciar, todos los archivos aparecerán en estado “untracked” o sin seguimiento.

Los archivos que queremos que estén en la próxima versión necesitarán pasar primero por el estado de “en seguimiento” o “staging”, antes de ser confirmados. Para ello utilizaremos el comando `git add`:

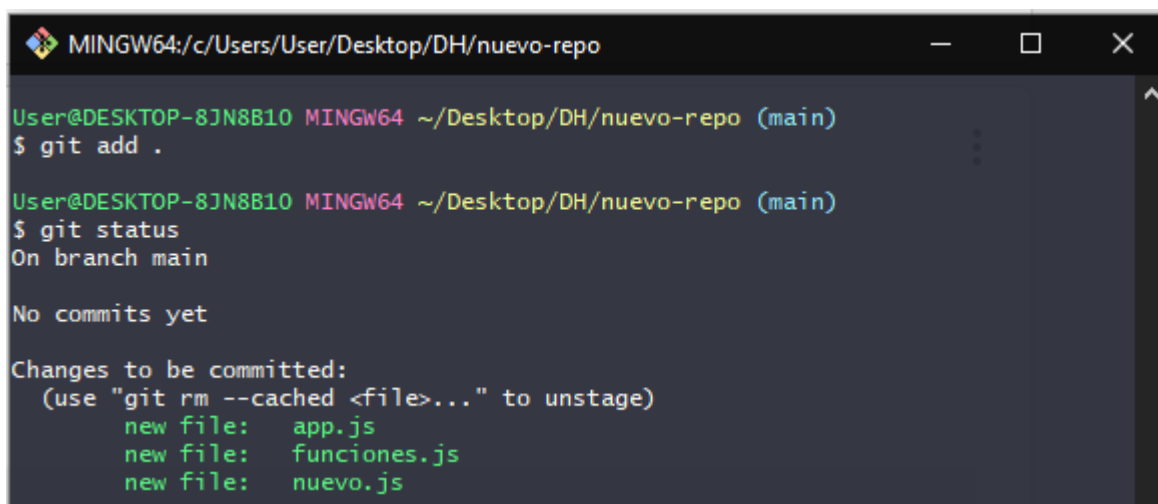
```
git add nombreadarchivo
```

Para agregar un sólo archivo, o

```
git add .
```

Para agregar todos los archivos sin seguimiento al staging área.

Si ejecutamos `git add .` y luego `git status`, veremos que ya nuestros archivos están en el staging area:



```
MINGW64:/c/Users/User/Desktop/DH/nuevo-repo
User@DESKTOP-8JN8B10 MINGW64 ~/Desktop/DH/nuevo-repo (main)
$ git add .

User@DESKTOP-8JN8B10 MINGW64 ~/Desktop/DH/nuevo-repo (main)
$ git status
On branch main


No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   app.js
        new file:   funciones.js
        new file:   nuevo.js
```

En este punto, ya estaremos en condiciones de confirmar estos archivos en una versión.

## Confirmando archivos

Para agregar archivos y directorios a una versión utilizaremos el comando `git commit`, seguido del flag `-m` y especificaremos un mensaje significativo sobre la versión. Por ejemplo:



```
MINGW64:/c/Users/User/Desktop/DH/nuevo-repo
User@DESKTOP-8JN8B10 MINGW64 ~/Desktop/DH/nuevo-repo (main)
$ git commit -m "Iniciando proyecto"
[main (root-commit) 58c81ad] Iniciando proyecto
3 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 app.js
create mode 100644 funciones.js
create mode 100644 nuevo.js

User@DESKTOP-8JN8B10 MINGW64 ~/Desktop/DH/nuevo-repo (main)
$ |
```

La primera versión de nuestro proyecto se guardará en pequeños objetos llamados commits, los que tendrán un número de identificación llamado hash, la fecha y el autor de esa versión. Para poder ver nuestra primera versión, y luego el historial de versiones

utilizamos el comando git log

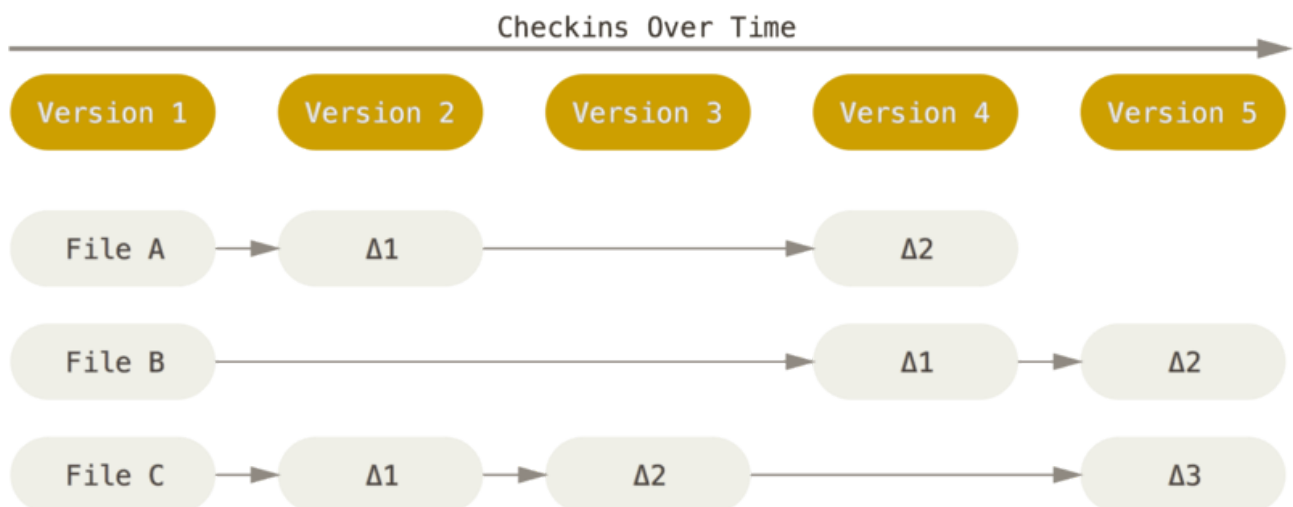
```
MINGW64:/c/Users/User/Desktop/DH/nuevo-repo
User@DESKTOP-8JN8B10 MINGW64 ~/Desktop/DH/nuevo-repo (main)
$ git log
commit 58c81ad82c97cfd5e871d155487a4d94ac3e3509 (HEAD -> main)
Author: nombreusuario <mail@gmail.com>
Date: Thu Sep 8 23:19:42 2022 -0300

    Inicializando proyecto

User@DESKTOP-8JN8B10 MINGW64 ~/Desktop/DH/nuevo-repo (main)
$ |
```

## Como guarda Git nuestros datos

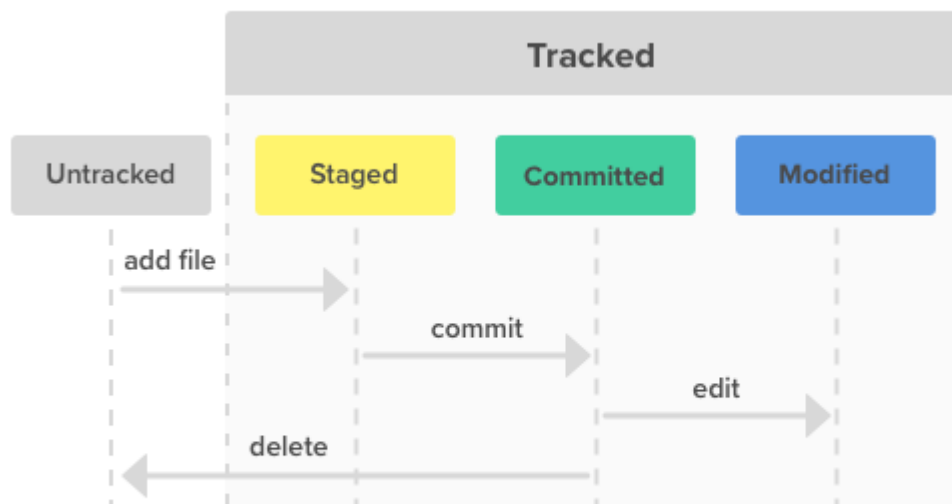
Con cada commit que hacemos, git básicamente toma una foto del aspecto de todos nuestros archivos en ese momento y guarda una referencia a esa foto. Si los archivos no se han modificado, Git no almacenará estos archivos de nuevo. Git maneja los cambios en un repositorio como una secuencia de fotos instantáneas.



## Repasando

El manejo de Git se basa en tres estados principales por los que pueden pasar los archivos una vez creados o modificados: confirmado (committed), modificado (modified) y

en seguimiento (staged o en staging area).



El flujo de trabajo básico de Git es algo así:

1. Modificamos o creamos una serie de archivos
2. Agregamos los archivos al área de seguimiento o staging
3. Confirmamos los cambios, y esa fotografía instantánea de los archivos se almacena de manera permanente en tu directorio de trabajo.

## GitHub

Github es una plataforma colaborativa en la nube que nos va a permitir llevar un control de versión sobre nuestros proyectos.

Hasta ahora creamos un repositorio local, pero con Github vamos a poder crear a partir de ese repositorio local, uno remoto, o copiar uno remoto a nuestra computadora.

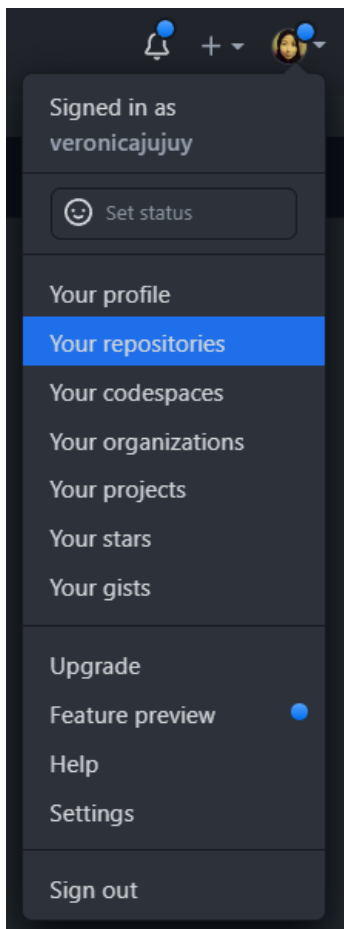
### Crear un repositorio remoto

Primero debemos tener un usuario creado en github: <https://github.com/>

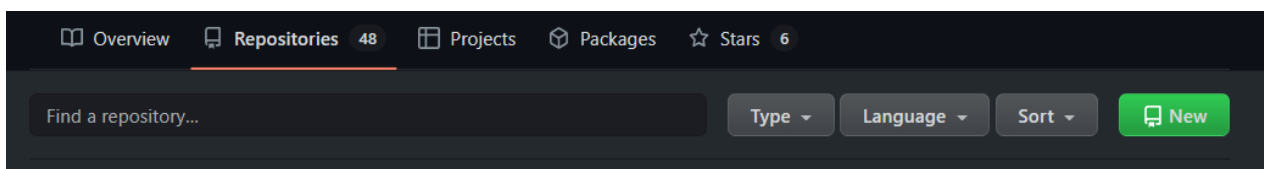
Ahora vamos a crear un repositorio.



Clickeamos al costado de la foto de nuestro perfil y hacemos clic en “Your repositories”



Acá hacemos clic en el botón new:





En la ventana que se abre, elegiremos un nombre de repositorio, y especificaremos si es que este debe ser público o privado. En general para las prácticas dejaremos todos los repositorios como públicos. Por ahora no agregaremos nada mas y presionaremos en en botón create repository

## Create a new repository

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

---


**Owner \*** **Repository name \***


 veronicajujuy / nuevo-repositorio 

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

**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:** None ▾

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

**License:** None ▾

---



 You are creating a public repository in your personal account.

---

[Create repository](#)

Ahora tendremos nuestro repositorio vacío:

**Quick setup — if you've done this kind of thing before**


 Set up in Desktop or **HTTPS** **SSH** <https://github.com/veronicajujuy/nuevo-repositorio.git> 

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

---

**...or create a new repository on the command line**

```
echo "# nuevo-repositorio" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/veronicajujuy/nuevo-repositorio.git
git push -u origin main
```

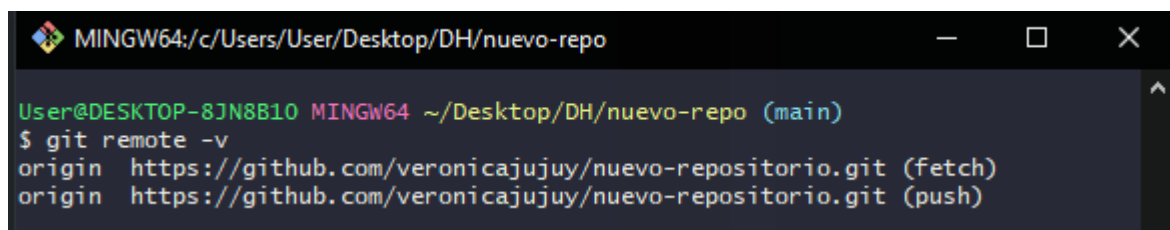


## Vinculando el repositorio remoto con el local

Ahora ya podremos vincular el repositorio que tenemos en nuestra computadora con el que acabamos de crear en Github. Para ello, en la carpeta que contiene nuestro repositorio local, debemos copiar el comando que nos recomienda github:

```
git remote add origin https://github.com/veronicajujuy/nuevo-repositorio.git
```

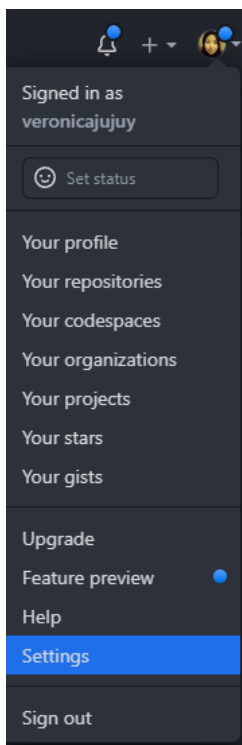
Para chequear que efectivamente se haya vinculado el repositorio escribimos `git remote -v`. Debería mostrarnos el repositorio conectado:

A screenshot of a terminal window titled 'MINGW64: c:/Users/User/Desktop/DH/nuevo-repo'. The prompt is 'User@DESKTOP-8JN8B10 MINGW64 ~/Desktop/DH/nuevo-repo (main)'. The command '\$ git remote -v' has been executed, resulting in two lines of output: 'origin https://github.com/veronicajujuy/nuevo-repositorio.git (fetch)' and 'origin https://github.com/veronicajujuy/nuevo-repositorio.git (push)'.

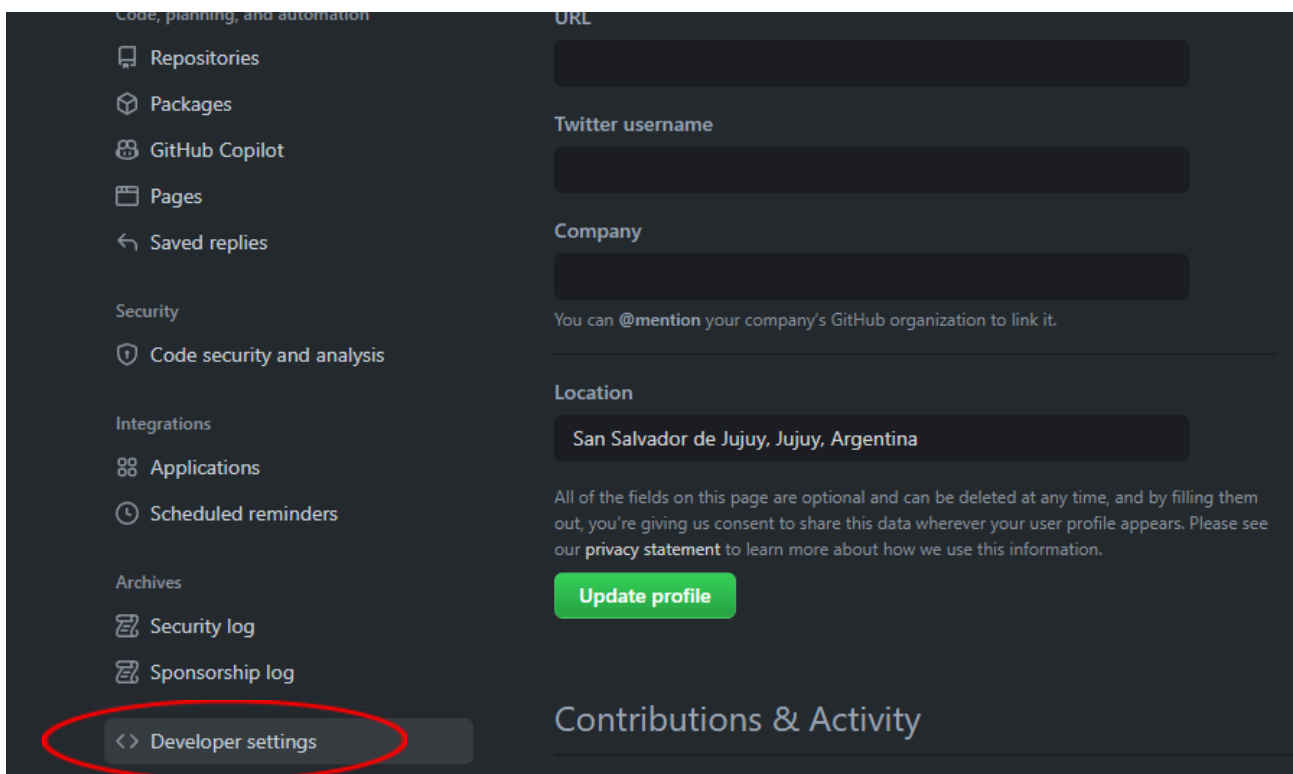
```
MINGW64: c:/Users/User/Desktop/DH/nuevo-repo
User@DESKTOP-8JN8B10 MINGW64 ~/Desktop/DH/nuevo-repo (main)
$ git remote -v
origin https://github.com/veronicajujuy/nuevo-repositorio.git (fetch)
origin https://github.com/veronicajujuy/nuevo-repositorio.git (push)
```

## Creando un token en Github

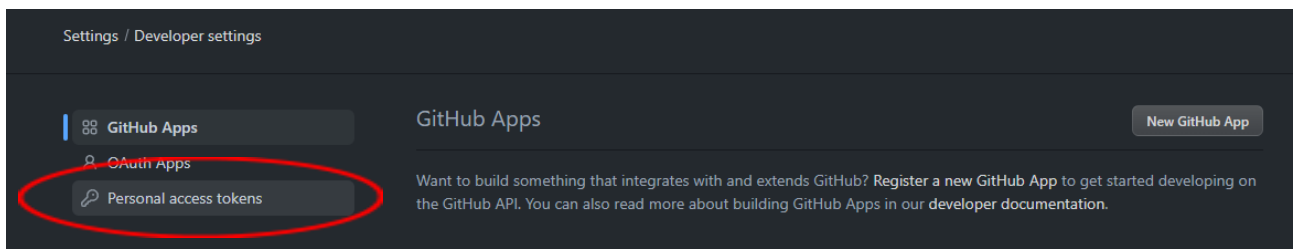
Antes de subir los archivos a Github, debemos crear un nuevo token en Github. Un token de acceso personal, son una alternativa al uso de contraseñas para la autenticación en Github. Para ello vamos al menú desplegable de nuestra cuenta y de allí a settings:



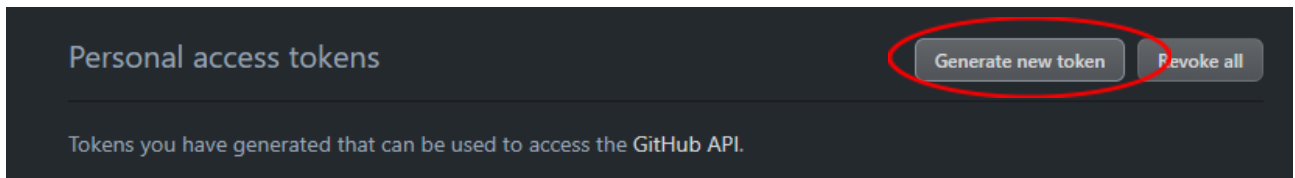
Una vez en settings, bajamos hasta la última opción del menú de la izquierda, “Developer Settings”



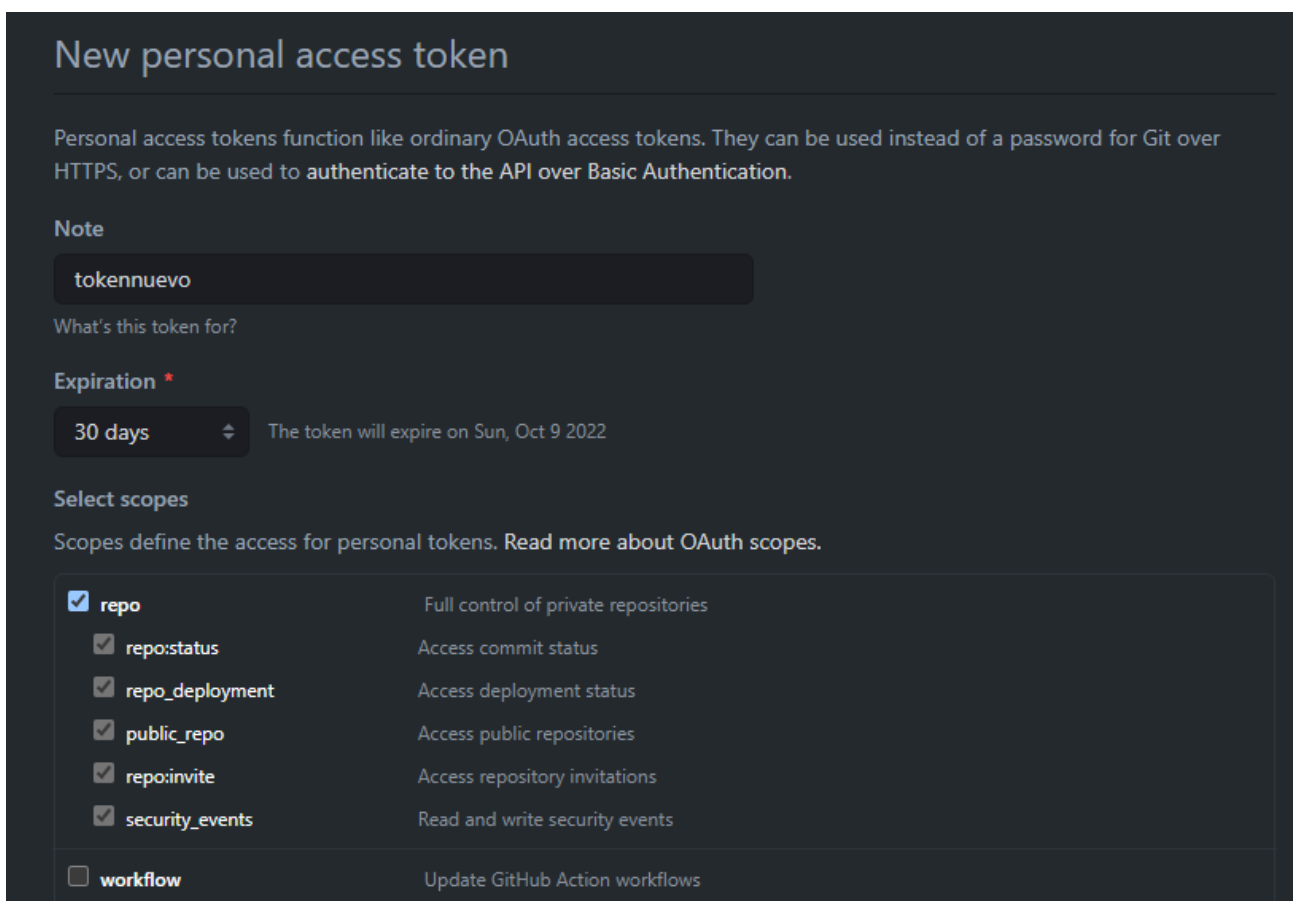
Luego hacemos clic en la opción Personal access tokens



Y luego en Generate new token:



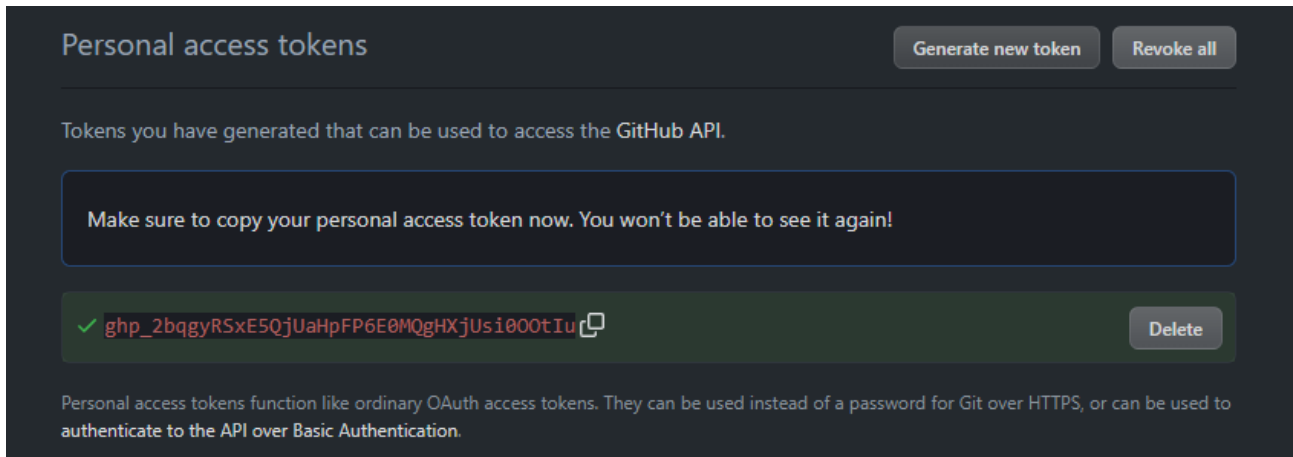
Nos pedirá la contraseña de github para continuar. Luego en la siguiente pantalla le pondremos un nombre al nuevo token. Una fecha de expiración. Y seleccionaremos el alcance del token. En nuestro caso solo necesitaremos elegir repo.



Es muy importante que elijamos al menos un scope o ámbito antes de guardar el token, sino no tendremos los permisos necesarios para subir cambios a los repositorios.

Ahora haremos clic en Generate token

En la siguiente pantalla nos mostrará por ÚNICA VEZ el nuevo token creado



Es imprescindible que copiemos este código y lo peguemos en algún archivo de texto en un lugar seguro.

## Subiendo archivos a Github

Ahora podemos subir los archivos creados en nuestro repositorio local al repositorio remoto que creamos en Github. Para ello utilizaremos el comando git push:

```
git push origin main
```

Es probable que nos pida nuestro nombre de usuario y contraseña. Le proporcionamos el nombre de usuario y cuando nos pida la contraseña, copiaremos el token creado anteriormente.

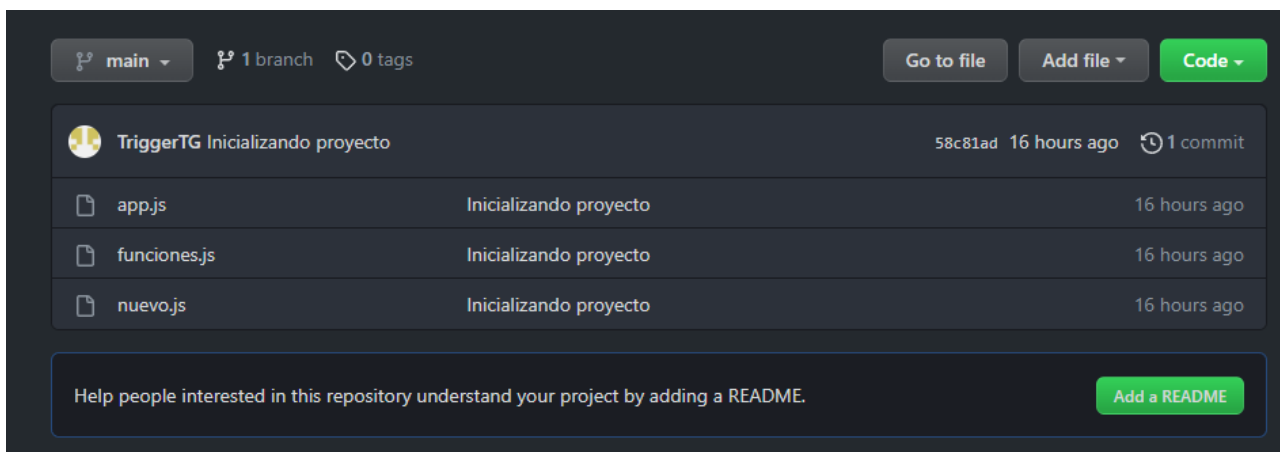
Importante! Generalmente cuando tipeamos la contraseña o copiamos el token, no nos muestra los clásicos \*\*\* de las contraseñas, sino que solo queda el prompt titilando. Es normal solo debemos asegurarnos de haber copiado la contraseña y luego hacer enter.

```
MINGW64:/c/Users/User/Desktop/DH/nuevo-repo

User@DESKTOP-8JN8B10 MINGW64 ~/Desktop/DH/nuevo-repo (main)
$ git push origin main
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% (3/3), 238 bytes | 238.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/veronicajujuy/nuevo-repositorio.git
 * [new branch]      main -> main

User@DESKTOP-8JN8B10 MINGW64 ~/Desktop/DH/nuevo-repo (main)
$
```

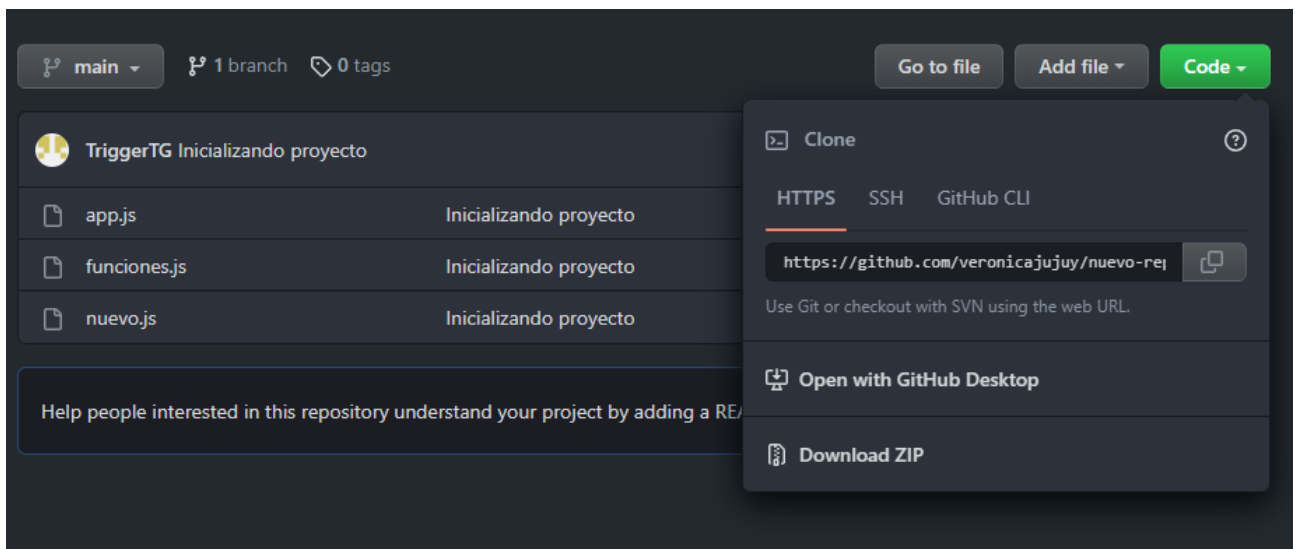
Veremos una leyenda como ésta, que nos especifica que los cambios ya fueron subidos al repositorio remoto. Si vamos al repositorio en Github veremos los mismos archivos que tengo en mi computadora



Ahora cada vez que hagamos cambios en nuestro repositorio local, haremos git add, git commit para crear una nueva versión y git push para subir a Github.

## Clonar un repositorio existente

Otra forma de vincular un repositorio local con uno remoto es clonar un repositorio existente a nuestra computadora local. Para ello, buscamos un repositorio ya existente y presionamos en el botón “Code”, acá nos abrirá una ventana con las opciones para clonar el repositorio:



Acá nos brinda tres formas de clonar el repositorio, por HTTPS, con el protocolo SSH y a través del Github CLI. Nosotros utilizaremos HTTPS. Copiamos el código y vamos al git bash.

Nos posicionamos en el directorio donde vamos a clonar el repositorio y tipeamos

```
git clone <url-repositorio>
```

Nos aparecerá una leyenda como esta:

```
MINGW64:/c/Users/User/Desktop/DH
User@DESKTOP-8JN8B10 MINGW64 ~/Desktop/DH
$ git clone https://github.com/veronicajujuy/nuevo-repositorio.git
Cloning into 'nuevo-repositorio'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.

User@DESKTOP-8JN8B10 MINGW64 ~/Desktop/DH
$ |
```

Ahora listamos los directorios y tiene que aparecer uno con el nombre del repositorio:



```
MINGW64:/c/Users/User/Desktop/DH

User@DESKTOP-8JN8B10 MINGW64 ~/Desktop/DH
$ ls
nuevo-repositorio/

User@DESKTOP-8JN8B10 MINGW64 ~/Desktop/DH
```

Ingresamos a ese directorio y allí recién estaremos dentro del repositorio clonado.

```
MINGW64:/c/Users/User/Desktop/DH/nuevo-repositorio

User@DESKTOP-8JN8B10 MINGW64 ~/Desktop/DH/nuevo-repositorio (main)
$ ls
app.js  funciones.js  nuevo.js

User@DESKTOP-8JN8B10 MINGW64 ~/Desktop/DH/nuevo-repositorio (main)
$
```

## Actualizar cambios en mi repositorio local

Si estamos trabajando en equipo, en Github seguramente habrá archivos de otros miembros. Cada vez que necesite actualizar mi repositorio local con los cambios producidos en el repositorio remoto debemos utilizar el comando `git pull origin main`.

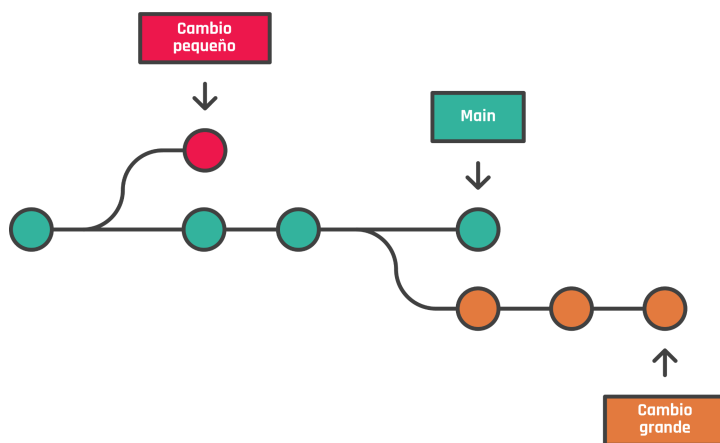
```
MINGW64:/c/Users/User/Desktop/DH/nuevo-repositorio

User@DESKTOP-8JN8B10 MINGW64 ~/Desktop/DH/nuevo-repositorio (main)
$ git pull origin main
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 281 bytes | 15.00 KiB/s, done.
From https://github.com/veronicajujuy/nuevo-repositorio
* branch      main      -> FETCH_HEAD
   58c81ad..620f7ac main    -> origin/main
Updating 58c81ad..620f7ac
Fast-forward
 app.js | 1 +
1 file changed, 1 insertion(+)
```

# Ramas o Branches

En Git las ramas o branches son espacios o entornos independientes que nos permitirán trabajar sobre un proyecto sin alterar o borrar el conjunto de archivos originales del proyecto.

Hasta ahora vimos que el que creamos tiene por defecto una rama principal llamada Main o Master —dependiendo de la versión de Git— en la cual vamos guardando las versiones de nuestro trabajo cada vez que hacemos un commit.



## Creando Ramas

Para crear una nueva rama utilizamos el comando `git branch`. Git branch nos permite crear, enumerar, cambiar el nombre y eliminar ramas. No permite cambiar entre ramas o volver a unir un historial bifurcado.

```
git branch
```

Enumera todas las ramas de tu repositorio, es similar a `git branch --list`.

```
git branch <branch>
```

Crea una nueva rama llamada `<branch>`.

```
git branch -d <branch>
```

Elimina la rama llamada `<branch>`. Git evita que eliminemos la rama si tiene cambios que aún no se han fusionado con la rama Main.

```
git branch -D <branch>
```

Fuerza la eliminación de la rama especificada, incluso si tiene cambios sin fusionar.

## Movernos de una rama a otra

Para movernos de una rama a otra, ejecutaremos el comando

```
git checkout <nombre_rama>
```

Generalmente, Git solo permitirá que nos movamos a otra rama si no tenemos cambios.

Si tenemos cambios, para cambiarnos de rama, debemos:

- Eliminarlos (deshaciendo los cambios).
- Confirmarlos (haciendo un git commit).

## Guardar cambios y subirlos al repositorio remoto

Una vez que terminamos de realizar los cambios que queremos en nuestra branch, ejecutamos los mismos comandos que vimos hasta ahora: git add, git commit, git status y git log. Pero cuando queramos subir esos cambios, debemos utilizar git push con el nombre de la rama en que estamos posicionados:

```
git push origin <branch>
```

Así también, para traer los cambios de esa rama utilizamos el git pull agregando desde donde queremos traer los cambios:

```
git pull origin <branch>
```

## Fusión de dos ramas

Cuando necesitamos traer el flujo de trabajo de una rama hacia otra nos posicionamos en la rama que deseamos agregar los commits realizados en la rama en donde se trabajaron dichos commits. una vez allí tipeamos

```
git merge branch
```

Por ejemplo si necesitamos traer a Main lo realizado en hotfix hacemos(Posicionados en Main):

```
git merge hotfix
```

## Comandos Básicos

**git status:** Nos permite ver el estado de nuestros archivos.

**git add:** Nos permite añadir un archivo a la Staging o área de preparación.

- `git add nombre_archivo`
- `git add .` (añade todos los archivos)

**git commit:** envía al local repository todos los archivos que fueron agregados.

**git commit -m “mensaje”:** permite agregar archivos a un commit

**git remote -v:** Nos permite chequear si el repositorio local esta vinculado a un remoto

**git remote add origin URL:** permite enlazar un repositorio local con uno remoto (se debe haber creado con anterioridad).

**git reset head:** quita los archivos de la zona staged y/o los devuelve a su estado anterior.

**git rm:** Nos permite eliminar el archivo del working directory pero no lo elimina del historial ya almacenado en git.

**git rm --cached:** Nos permite mover los archivos al estado anterior o untrucked.

**git rm --force:** Nos permite eliminar los archivos de git y del disco duro.

**git config --global init.defaultBranch main**

**git branch “nombre”**

**git checkout “nombre” para cambiarse**

**git checkout -b “nombre”** crea y se cambia en el mismo paso.

**git show** muestra el último commit enlazado con dos ramas.

## Otros Comandos interesantes

**git show**: muestra los cambios que han existido sobre un archivo.

**git diff**: muestra la diferencia entre una version y otra.

Por ejemplo:

```
git diff commit1 commit2
```

**git diff –staged**: vemos los cambios por etapas entre dos versiones.

**git log**: obtenemos el ID de los commits y mostramos el historial de commits de forma local.

**git log –graph** muestra un gráfico sobre los commits en la branch.

**git branch -d branch** : elimina la rama local.