



IMPLANTACIÓN DE APLICACIONES WEB

# Git y GitHub

IES LAS FUENTEZUELAS  
REGINA ALBÍN RODRÍGUEZ



## Tabla de contenido

|         |   |    |
|---------|---|----|
| 1       | Introducción .....  | 3  |
| 2       | Qué es Git y para qué sirve.....  | 3  |
| 3       | Qué es y cómo funciona GitHub .....   | 3  |
| 4       | Términos básicos .....  | 3  |
| 5       | Prerrequisitos .....  | 4  |
| 6       | Flujo de trabajo de Git .....   | 4  |
| 7       | Cómo crear un repositorio en GitHub.....  | 6  |
| 8       | Instalación de Git.....   | 7  |
| 9       | Configuración de Git.....   | 10 |
| 10      | Cómo crear repositorios desde cero y almacenar cambios en nuestro repositorio local | 12 |
| 10.1    | git init .....  | 12 |
| 10.2    | gitignore .....   | 13 |
| 10.3    | git add .....   | 15 |
| 10.4    | git rm .....  | 15 |
| 10.5    | git commit .....  | 16 |
| 10.6    | Git remote. Conexión repositorio local con uno remoto .....                         | 17 |
| 10.7    | git mv .....  | 20 |
| 10.8    | git clone .....   | 20 |
| 10.9    | Repositorio remoto compartido .....   | 23 |
| 10.10   | Git Branch.....   | 23 |
| 10.10.1 | ¿Qué es una rama de Git?.....   | 23 |
| 10.11   | git push .....  | 26 |
| 10.12   | git pull .....  | 27 |
| 10.13   | git merge.....  | 28 |
| 10.14   | Eliminar un branch.....   | 29 |
| 11      | Bibliografía.....   | 30 |

# 1 Introducción

Todos los desarrolladores suelen utilizar algún tipo de **sistema de control de versiones (VCS)**, una herramienta que les permita colaborar con otros desarrolladores en un proyecto sin peligro de que sobrescriban el trabajo de los demás, y volver a las versiones anteriores de la base de código si existe un problema descubierto más tarde. El VCS más popular (al menos entre los desarrolladores web) es **Git**, junto con **GitHub**, un sitio que proporciona alojamiento para los repositorios y varias herramientas para trabajar con ellos.

## 2 Qué es Git y para qué sirve

Git es un Sistema de Control de Versiones Distribuido (DVCS) utilizado para guardar diferentes versiones de un archivo (o conjunto de archivos) para que cualquier versión sea recuperable cuando se desee.

Git también facilita el registro y comparación de diferentes versiones de un archivo o proyecto. Esto significa que los detalles sobre qué se cambió, quién cambió qué, o quién ha iniciado una propuesta, se pueden revisar en cualquier momento. Git nos permite hacer un seguimiento de las modificaciones realizadas en los archivos en nuestro ordenador y en nuestro repositorio remoto.

Git se usa para:

- El manejo de repositorios y ramas.
- Poder trabajar en equipo.
- Volver a un archivo en el momento en el que estaba correcto.

## 3 Qué es y cómo funciona GitHub

GitHub es una corporación americana, es un sitio web + infraestructura que proporciona un servidor Git más una serie de herramientas realmente útiles para trabajar con repositorios git individuales o en equipo, como informar problemas con el código, herramientas de revisión, características de administración de proyectos tal como asignación de tareas, estados de tareas, y más.

## 4 Términos básicos

**GIT**→Es un sistema de control de versiones: un software que permite registrar el historial de cambios de un proyecto.

**Repositorio**→ Es todo proyecto que está siendo seguido por GIT, ya tiene un historial de GIT en el que se están registrando sus cambios.

**Commit**→Es cada uno de los cambios registrados en el historial de GIT. Cada uno de los desarrolladores manda los commits de los cambios que ha hecho. No es automático, cada desarrollador tiene que decir qué hizo y por qué.

**Ramas**→ Son nuevos caminos que toma el proyecto. La rama principal se llama *máster* y es donde está el proyecto que sale a producción. Cada vez que se saca una nueva característica o que se quiere corregir algo se saca una rama, de tal manera que se pueda trabajar en un ambiente aislado. Es una copia exacta del proyecto, pero está separada. De este modo, si algo se rompe en ese proyecto, no comprometes al proyecto original. Si todo va bien, puedes unificar esa rama con el proyecto principal, si va mal, puedes eliminarla sin ningún problema.

**Clon**→Es una copia exacta del repositorio. Cuando un programador se integra a un equipo de trabajo lo primero que debe hacer es clonar el repositorio en su equipo local. De esa manera cada miembro del equipo tiene un clon del repositorio en su equipo local.

## 5 Prerrequisitos

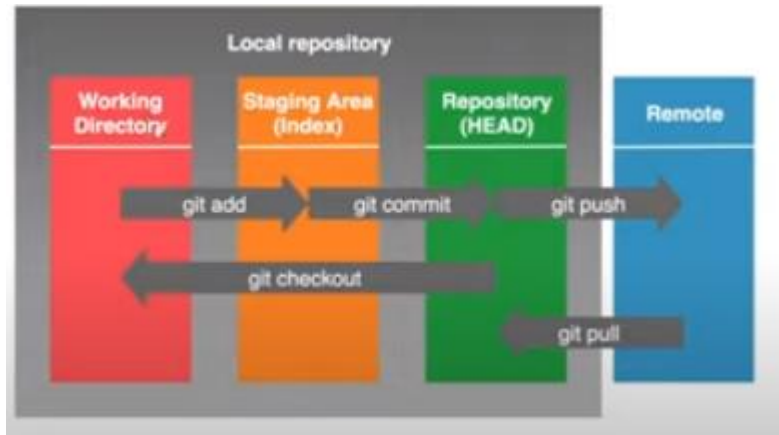
Para usar Git y GitHub, necesitas:

- Un ordenador con Git instalado. (Consulta la [página de descargas de Git](#)).
- Una herramienta para usar Git. Dependiendo de cómo te guste trabajar, puedes usar un cliente Git con GUI (Git Bash, GitHub Desktop, SourceTree o Git Kraken) o simplemente usar una ventana de la terminal.
- Una [cuenta de GitHub](#).

## 6 Flujo de trabajo de Git

Git tiene tres estados principales en los que se pueden encontrar tus archivos: confirmado (committed), modificado (modified), y preparado (staged). Confirmado: significa que los datos están almacenados de manera segura en tu base de datos local. Modificado: significa que has modificado el archivo, pero todavía no lo has confirmado a tu base de datos. Preparado: significa que has marcado un archivo modificado en su versión actual para que vaya en tu próxima confirmación.

Esto nos lleva a las tres secciones principales de un proyecto de Git: El directorio de Git (Git directory), el directorio de trabajo (working directory) y el área de preparación (staging area).



Para poder utilizar la línea de comandos necesitamos entender cómo funciona un repositorio local y un repositorio remoto.

Cuando trabajamos a nivel local lo que realizamos es lo siguiente. En nuestro directorio de trabajo (nuestro ordenador) se van a albergar todos los cambios o cosas nuevas que tengamos o queramos versionar a nivel local. Veamos esto con un ejemplo: vamos a crear una carpeta llamada *git-test*, la cual va a albergar todos los cambios o cosas nuevas que realicemos a nivel local. Dentro de esta creamos otra carpeta *folder 1* y archivo llamado *text-file.txt*. La carpeta *git-test* va a ser nuestro Working directory, es decir, va a ser el directorio donde vamos a estar trabajando a nivel local.

La idea es que cualquier cambio que realizamos en nuestro código, archivo,... almacenado en el Working directory, debe quedar reflejado en nuestro repositorio local, pero para ello tenemos que pasar por un paso intermedio llamado Staging Area.

Posteriormente, podremos mandar los cambios de nuestro repositorio local al repositorio remoto. Este repositorio remoto, lo podemos identificar como Git Hub.

Si trabajamos en local (comenzamos en la imagen por la izquierda), inicializamos el directorio de trabajo (working directory). Podemos trabajar (editar ficheros) en el directorio de trabajo.

Con el comando *Git add* enviamos los cambios a staging, que es un estado intermedio en el que se van almacenando los archivos a enviar en el *commit*. Finalmente con *commit* lo enviamos al repositorio local.

Si queremos colaborar con otros, con *push* subimos los archivos a un repo remoto y mediante *pull* podríamos traer los cambios realizados por otros en remoto hacia nuestro directorio de trabajo.

Si comenzamos trabajando en remoto, lo primero que hacemos es un clon (copia) de la información en el directorio local.

Comandos básicos:

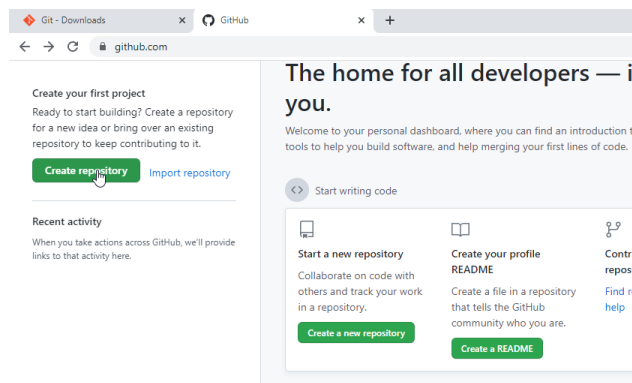
- *git init* → Crear un repositorio local desde cero
- *git clone* → Es para descargar un repositorio que ya existe.
- *git add <file>* → Añadir un/os archivo/s al Staging Area (Index)
- *git commit -m "Descripción"* → commit cambios en el repositorio local (Head)

- `git log` → Verificar los logs de commit (es una ayuda)
- `git status` → Verificar el estado de nuestros archivos
- `git push` → Para pasar los cambios de nuestro repositorio al repositorio remoto
- `git diff` → para ver los cambios que ha habido en nuestros archivos
- `git checkout <branch>` → Cambio entre ramas

## 7 Cómo crear un repositorio en GitHub

Para ello lo primero que vamos a hacer es creamos una [cuenta de GitHub](#) y debemos tener creada en nuestro ordenador la carpeta Git-test, que se indicó anteriormente.

Comenzamos a crear nuestro repositorio.



Vamos a crearlo privado, ya que queremos indicar quien va a poder acceder

Owner: albirregi / Repository name: git-test

Description (optional): Repositorio para realizar pruebas

Public (selected) / Private

Initialize this repository with: Add a README file

Add .gitignore: .gitignore template: None

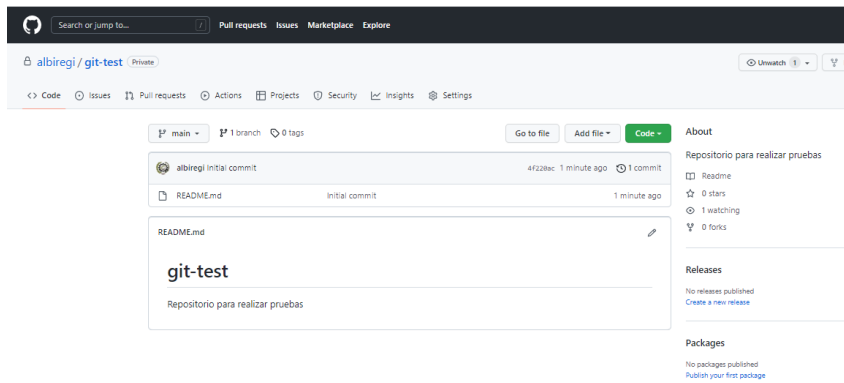
Choose a license: License: None

This will set main as the default branch. Change the default name in your settings.

You are creating a private repository in your personal account.

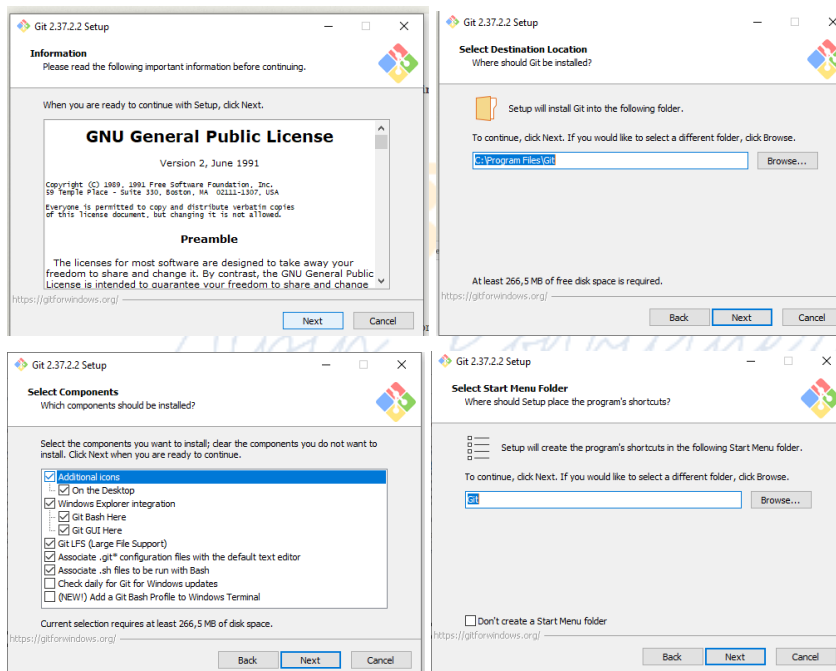
Create repository

Repositorio creado

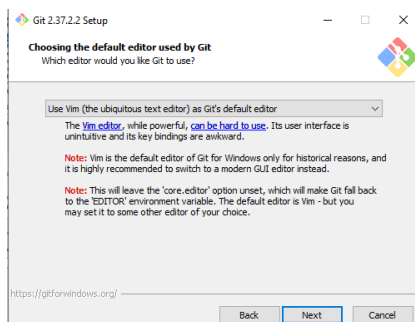


## 8 Instalación de Git

Descargar la versión de Windows <https://git-scm.com/downloads>

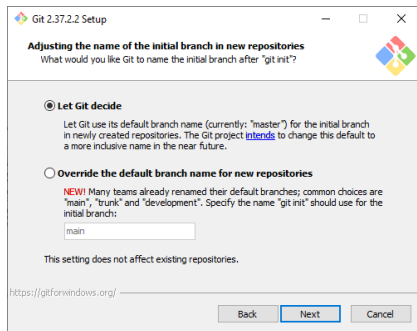


A continuación nos indica si queremos utilizar Vim como editor de texto

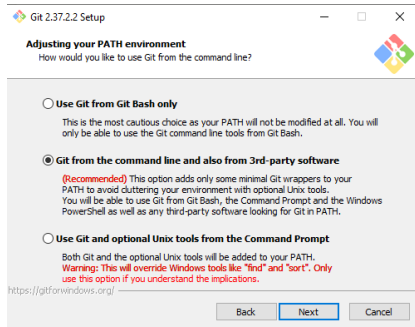


Para indicar el manejo que va a hacer Git con los repositorios que tenemos en el ordenador.

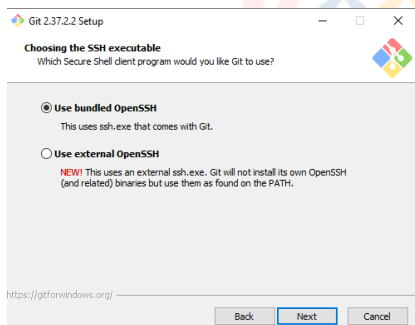
## Git y GitHub



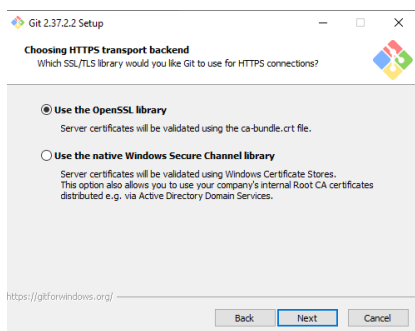
Para utilizar Git desde línea de comando y software de terceros.



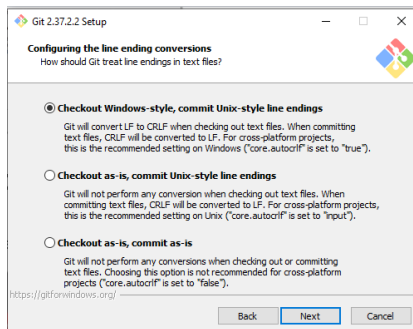
Utilizar SSH (Quizás esta ventana no salga, si al registrarnos en Git, no hemos elegido opciones de seguridad, pero no importa).



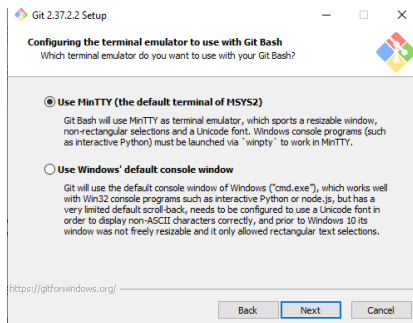
Utilizar librería SSL



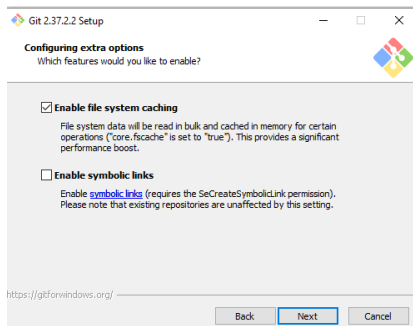
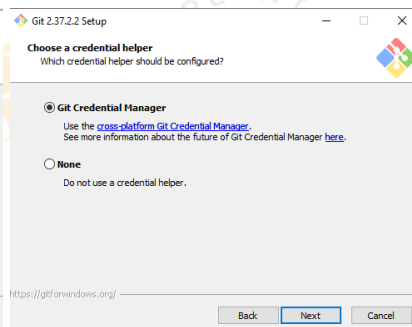
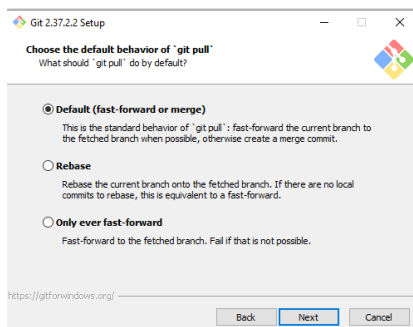




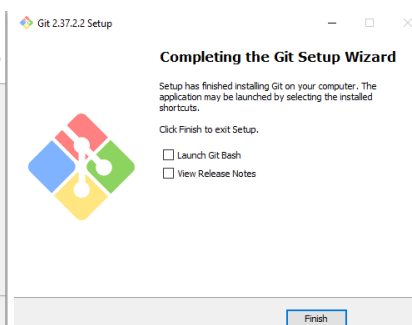
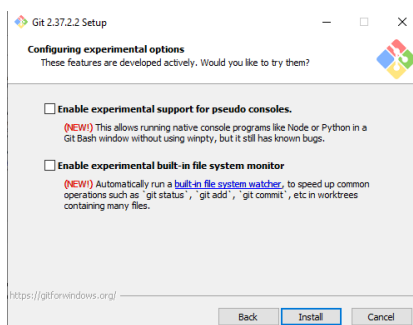
## Terminal que se va a utilizar



## Opción de PULL

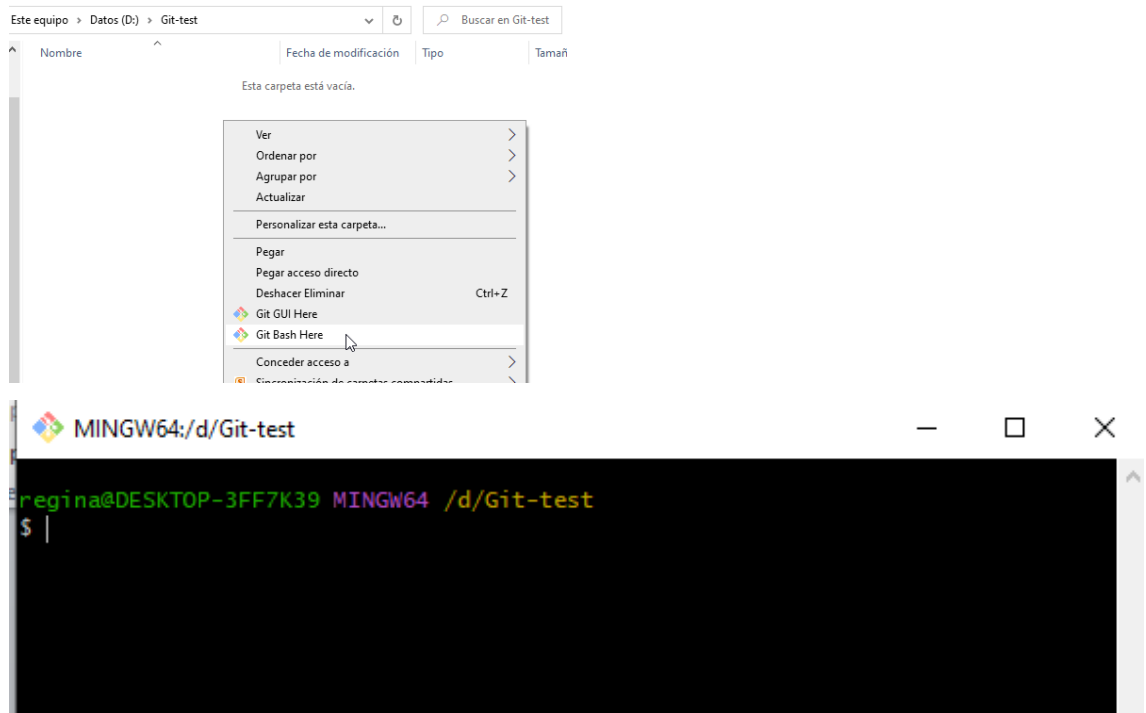


No vamos a elegir ninguna opción, ya que no vamos a realizar ningún tipo de experimentación.

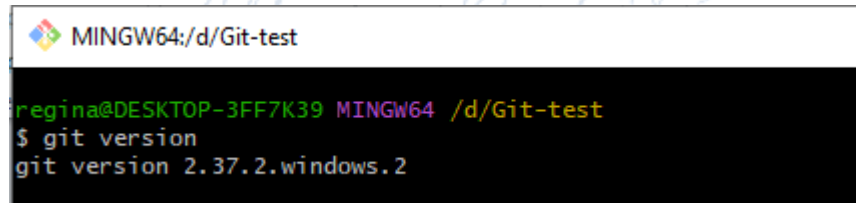


## 9 Configuración de Git

Nos situamos en la carpeta en la que vamos a trabajar y botón derecho en *Git Bash Here* para entrar en la consola



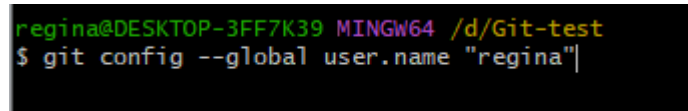
Podemos verificar la versión de Git que tenemos instalada



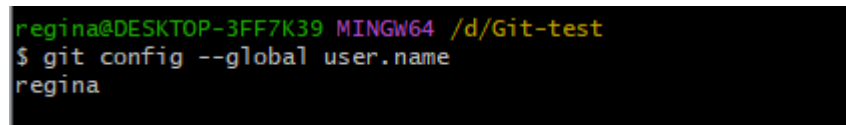
```
git version
```

Es necesario crear un usuario y cuenta de correo que vamos a utilizar para que Git pueda seguir nuestro rastro y controlar cada cambio que realicemos.

Vamos a crear nuestro usuario



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



Para ver el nombre de usuario

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

Indicar el correo que vamos a utilizar con Git

```
regina@DESKTOP-3FF7K39 MINGW64 /d/Git-test
$ git config --global user.email "albiiregi@gmail.com"
```

```
git config --global user.email "correo_personal"
```

Para ver el correo que estamos utilizando

```
git config --global user.email
```

Para validar los cambios/modificaciones que tenemos en nuestro Git

```
regina@DESKTOP-3FF7K39 MINGW64 /d/Git-test
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager-core
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
user.name=regina
user.email=albiiregi@gmail.com
```

```
git config --list
```

Para ver los archivos y carpetas que tenemos en la carpeta en la que estamos trabajando. Vamos a crear primero en la carpeta *Git-test* una carpeta llamada *Folder 1* para ver este comando.

```
regina@DESKTOP-3FF7K39 MINGW64 /d/Git-test
$ ls
'folder 1/'
```

```
ls
```

Para ver con más detalle lo almacenado en nuestra carpeta: Permisos, nombre usuario, fecha de creación

```
regina@DESKTOP-3FF7K39 MINGW64 /d/Git-test
$ ll
total 0
drwxr-xr-x 1 regina 197121 0 Aug 19 14:04 'folder 1/'
```

```
ll
```

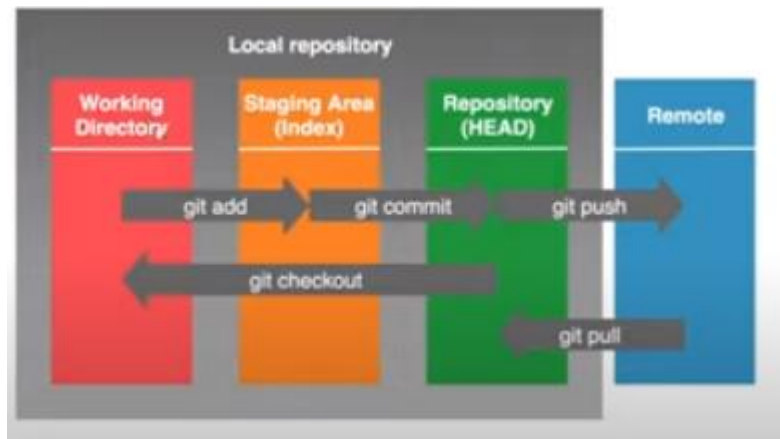
Para ver los archivos y directorios ocultos

```
regina@DESKTOP-3FF7K39 MINGW64 /d/Git-test
$ ls -alh
total 16K
drwxr-xr-x 1 regina 197121 0 Aug 19 14:04 ./
drwxr-xr-x 1 regina 197121 0 Aug 19 12:52 ../
drwxr-xr-x 1 regina 197121 0 Aug 19 14:04 'folder 1'/'
```

```
ls -alh
```

## 10 Cómo crear repositorios desde cero y almacenar cambios en nuestro repositorio local

En este apartado vamos a trabajar con el cuadro gris, que representa el repositorio local.



### 10.1 git init

Primero vamos a iniciar nuestro Git local

```
git init
```

```
regina@DESKTOP-3FF7K39 MINGW64 /d/Git-test
$ git init
Initialized empty Git repository in D:/Git-test/.git/

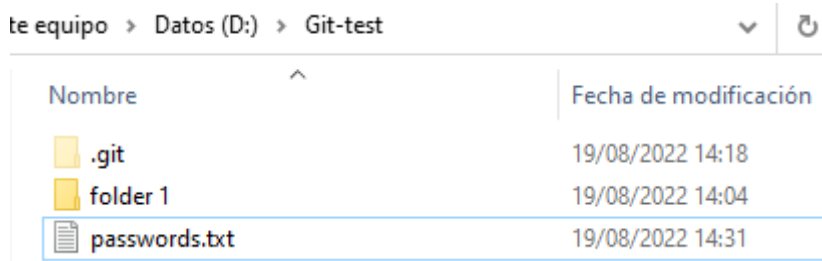
regina@DESKTOP-3FF7K39 MINGW64 /d/Git-test (master)
$
```

Con esto ya tenemos un repositorio hecho "master". En nuestra carpeta se ha creado una carpeta oculta llamada `.git`

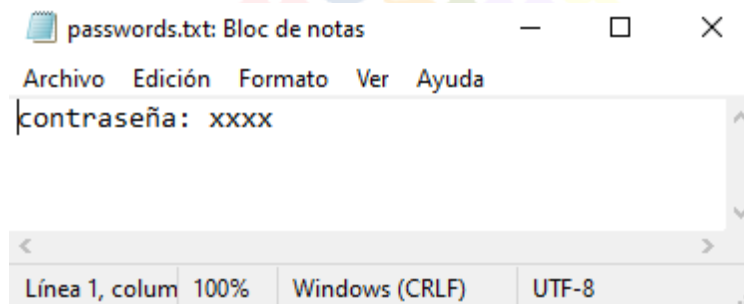
```
regina@DESKTOP-3FF7K39 MINGW64 /d/Git-test (master)
$ ls -alh
total 20K
drwxr-xr-x 1 regina 197121 0 Aug 19 14:18 ./
drwxr-xr-x 1 regina 197121 0 Aug 19 12:52 ../
drwxr-xr-x 1 regina 197121 0 Aug 19 14:18 .git/
drwxr-xr-x 1 regina 197121 0 Aug 19 14:04 'folder 1'/
```

## 10.2 gitignore

Vamos a crear un archivo llamado *passwords* donde vamos a almacenar contraseñas, pero no queremos que se muestre en el repositorio remoto.



En el que vamos a escribir nuestras contraseñas

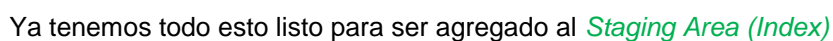
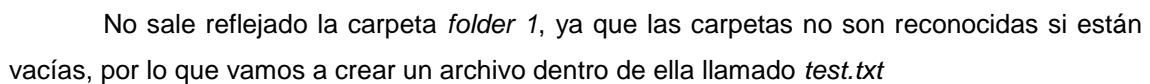
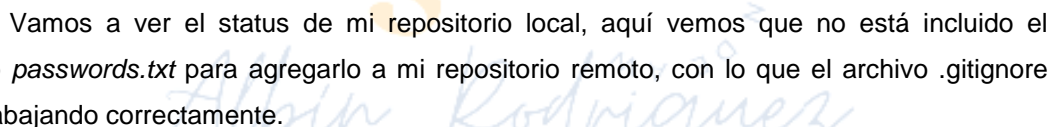
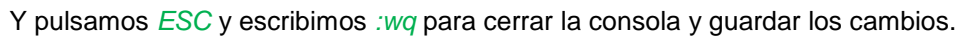


Para ello vamos a crear un archivo *.gitignore* para que cuando indiquemos los cambios a nuestro repositorio remoto, no mande los cambios realizados en este archivo. Vamos a abrir el editor *vim*

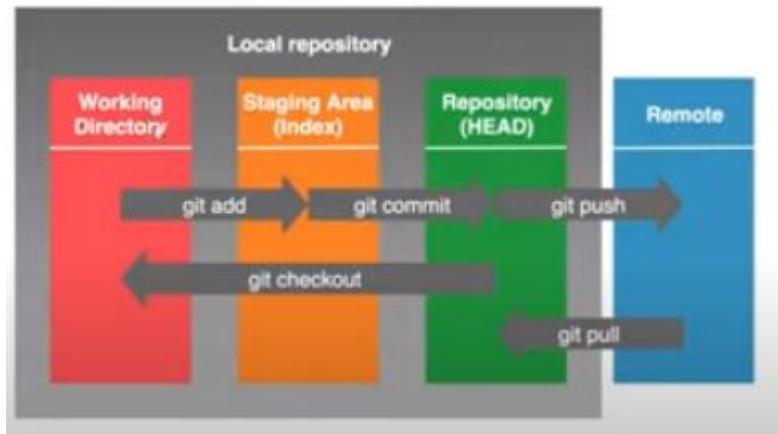
```
regina@DESKTOP-3FF7K39 MINGW64 /d/Git-test (master)
$ vim .gitignore
```

vim .gitignore

Aquí escribimos los archivos y directorios que no queremos que se muestren en el repositorio remoto cuando realicemos un *commit*.



## 10.3 git add



Con este comando añadimos todo al Staging Area independientemente de la carpeta en la que nos encontremos todos los archivos y carpetas que han sido creados o han tenido alguna modificación.

```
regina@DESKTOP-3FF7K39 MINGW64 /d/Git-test (master)
$ git add -A
```

git add -A

Al ver el estado de mi git, veremos que no es necesario hacer ningún add, solo nos indica que ahora hay que hacer commit para almacenar los cambios en el repositorio local - Repository (HEAD)-.

```
regina@DESKTOP-3FF7K39 MINGW64 /d/Git-test (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   .gitignore
        new file:   folder 1/test.txt
```

## 10.4 git rm

Si por ejemplo el archivo test.txt no queremos hacerle un commit porque queremos modificarlo antes, para ello escribimos lo siguiente.

```
regina@DESKTOP-3FF7K39 MINGW64 /d/Git-test (master)
$ git rm --cached 'folder 1'/test.txt
rm 'folder 1/test.txt'
```

Al hacer status veremos que este archivo ya no está para el **commit** sino para el **add**

```
regina@DESKTOP-3FF7K39 MINGW64 /d/Git-test (master)
$ git status
On branch master

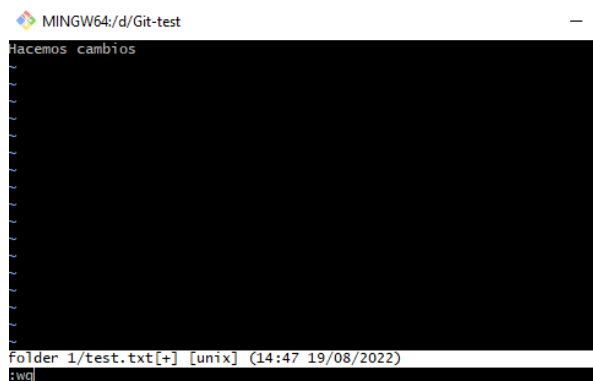
No commits yet

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

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        folder 1/
```

Hacemos unos cambios

```
regina@DESKTOP-3FF7K39 MINGW64 /d/Git-test (master)
$ vim 'folder 1'/test.txt
```



A continuación hacemos un git add. Podemos indicar el archivo en particular al que queremos hacer un add o poner add -A como antes

```
regina@DESKTOP-3FF7K39 MINGW64 /d/Git-test (master)
$ git add 'folder 1'/test.txt
warning: in the working copy of 'folder 1/test.txt', LF will be replaced by CRLF
the next time Git touches it
```

Si escribimos el comando status, veremos que ya *test.txt* está listo para commit

```
regina@DESKTOP-3FF7K39 MINGW64 /d/Git-test (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   .gitignore
        new file:   folder 1/test.txt
```

## 10.5 git commit

Ahora vamos a proceder a hacer la transición del Staging Area al Repositorio Local para ello utilizamos el comando *git commit -m "comentario"*. En el comentario debemos añadir alguna información que nos sirva para los logs del commit.



```
regina@DESKTOP-3FF7K39 MINGW64 /d/Git-test (master)
$ git commit -m "commit inicial"
[master (root-commit) 8f8d4fa] commit inicial
2 files changed, 2 insertions(+)
create mode 100644 .gitignore
create mode 100644 folder 1/test.txt
```

```
git commit -m "commit inicial"
```

```
regina@DESKTOP-3FF7K39 MINGW64 /d/Git-test (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Observamos que ya todos los archivos y carpetas están en el repositorio local listos para ser enviados al repositorio remoto.

Para ver los commit que se han ido realizando utilizaremos el siguiente comando.

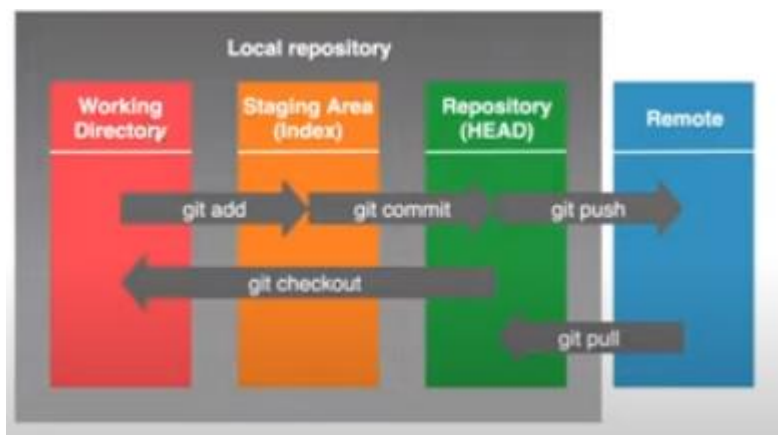
```
regina@DESKTOP-3FF7K39 MINGW64 /d/Git-test (master)
$ git log
commit 8f8d4fa6cf0b846a0e194fdb68f94fb7e101403e (HEAD -> master)
Author: regina <albiiregi@gmail.com>
Date: Fri Aug 19 16:43:12 2022 +0200

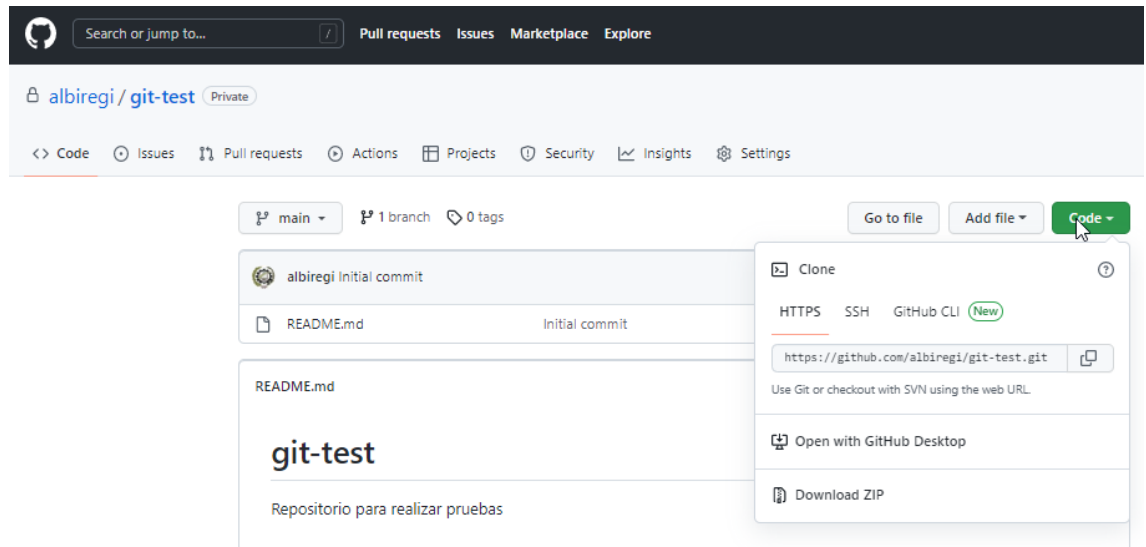
    commit inicial
```

```
git log
```

## 10.6 Git remote. Conexión repositorio local con uno remoto

En este apartado vamos a guardar nuestro repositorio local en Git (Rectángulo azul).





Dirección https con la que nos vamos a comunicar con el repositorio remoto.

Con el siguiente comando vamos a enviar nuestros cambios de manera remota.

```
regina@DESKTOP-3FF7K39 MINGW64 /d/Git-test (master)
$ git remote add origin https://github.com/albiregi/git-test.git
```

```
git remote add origin https://github.com/albiregi/git-test.git
```

A continuación indicamos a Git que queremos que nuestra rama se llame *main*.

```
regina@DESKTOP-3FF7K39 MINGW64 /d/Git-test (master)
$ git branch -M main
```

```
git branch -M main
```

Por último, escribimos el siguiente comando, estamos ligando nuestro repositorio local con el remoto.

```
regina@DESKTOP-3FF7K39 MINGW64 /d/Git-test (main)
$ git push -u origin main
```

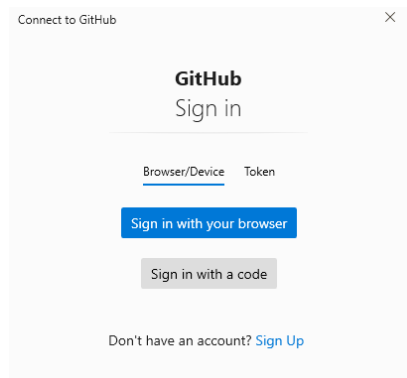
```
git push -u origin main
```

```
regina@DESKTOP-3FF7K39 MINGW64 /d/git-test (main)
$ git push -u origin main
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (5/5), 375 bytes | 375.00 KiB/s, done.
Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/albiregi/git-test.git
4f220ac..11b9f76 main -> main
branch 'main' set up to track 'origin/main'.
```

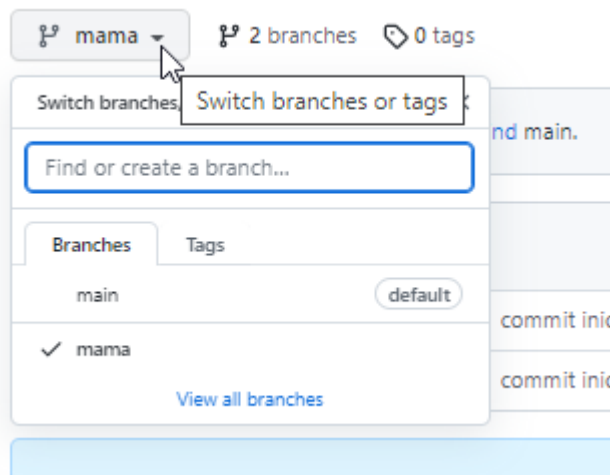
Si diera error cambiar el nombre de la rama o forzar a que sea en esa rama

```
git push -u origin main --force
```

Ahora GitHub me pide que me identifique



Pulsamos F5, elegimos la rama que hemos creado



Y ya vemos en el repositorio remoto lo que teníamos en el local.



Resumen

```
git remote add origin https://github.com/albiregi/git-test.git
git branch -M main
git push -u origin main
```

## 10.7 git mv

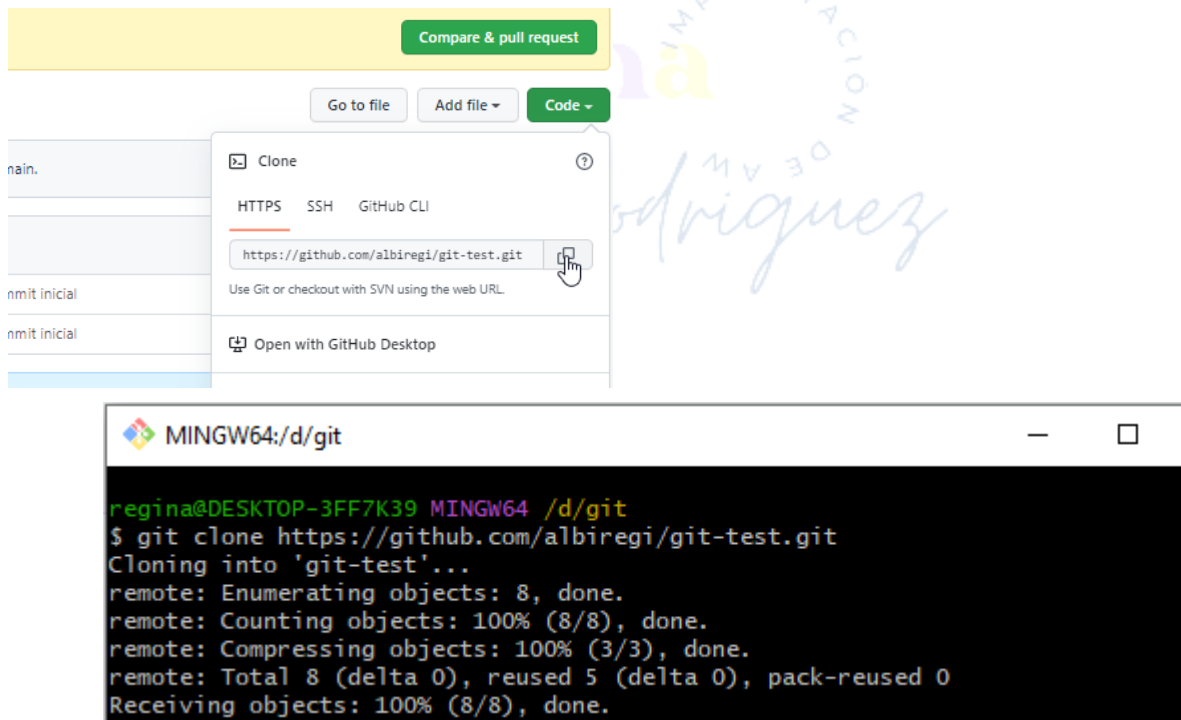
Para cambiar el nombre de un fichero o directorio de tu repositorio.

```
git mv test.txt prueba.txt
git commit -am "He modificado el nombre del fichero test.txt"
git push
```

## 10.8 git clone

Primero vamos a eliminar del ordenador la carpeta Git-test que es donde tenía mi repositorio. Creamos una carpeta llamada Git y abrimos ahí [Git Bash Here](#).

Vamos a clonar nuestro repositorio que tenemos en GitHub en nuestro ordenador.



Al situarnos en la carpeta `git` que habíamos creado, veremos que está alojado el repositorio que acabamos de clonar.



Ahora puedo ver el contenido de este repositorio y los logs que he ido realizando

```
regina@DESKTOP-3FF7K39 MINGW64 /d/git
$ cd git-test

regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (main)
$ git log
commit 4f220ac31c0aabe084d3e4db716c8547b36eebae (HEAD -> main, origin/main, origin/HEAD)
Author: albiregi <111572134+albiregi@users.noreply.github.com>
Date:   Fri Aug 19 12:59:10 2022 +0200

    Initial commit
```

Vamos a modificar el contenido del fichero que hay dentro de *folder 1* y escribimos Segundo cambio después de clonar.

```
regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (main)
$ vim "folder 1"/test.txt
```

Observamos que hay que hacer de nuevo Add para añadirlo a nuestro Staging Area.

```
regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   folder 1/test.txt
        deleted:    passwords.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        folder 1/.test.txt.swp

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

El comando git diff nos permite ver los cambios que ha habido en nuestros archivos.

```
regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (main)
$ git diff
diff --git a/folder 1/test.txt b/folder 1/test.txt
index e69de29..9daa114 100644
--- a/folder 1/test.txt
+++ b/folder 1/test.txt
@@ -0,0 +1,2 @@
+Hacemos cambios
+Segundo cambio despues de clonar
```

En verde se muestran los cambios que se han realizado. **Nota:** +Hacemos cambios aparece en ver porque se modificó durante la realización de los apuntes.

```
regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (main)
$ git add -A

regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (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:   folder 1/.test.txt.swp
    modified:   folder 1/test.txt
    deleted:    passwords.txt
```

Hacemos el commit

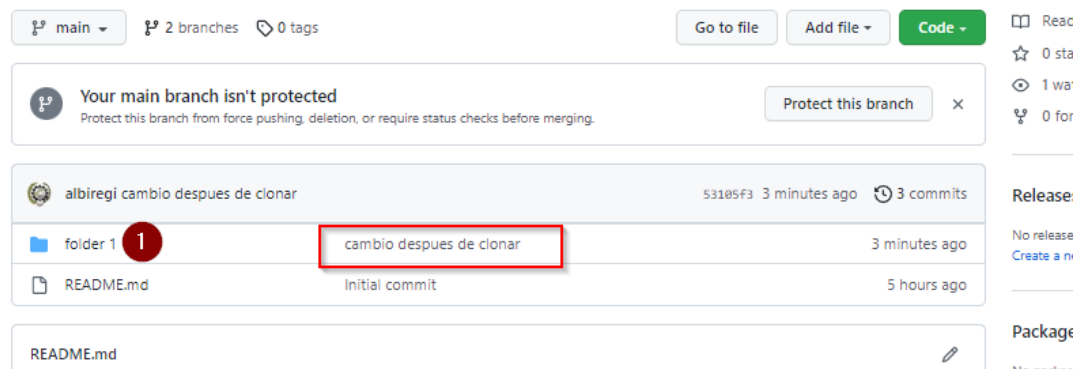
```
regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (main)
$ git commit -m "cambio despues de clonar"
[main 53105f3] cambio despues de clonar
3 files changed, 2 insertions(+), 1 deletion(-)
create mode 100644 folder 1/.test.txt.swp
delete mode 100644 passwords.txt

regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (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
```

Subimos los cambios del repositorio local al remoto.

```
regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (main)
$ git push origin main
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 607 bytes | 607.00 KiB/s, done.
Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/albiregi/git-test.git
  11b9f76..53105f3  main -> main
```



Observamos que ya está actualizado el repositorio remoto. Si abrimos 1 veremos los archivos que contienen donde test.txt tiene los cambios realizados.

## 10.9 Repositorio remoto compartido

Supongamos que en un repositorio están trabajando varias personas y una de ellas realiza un commit y no tenemos ese cambio almacenado en nuestro ordenador. ¿Qué debemos hacer? Guardar ese cambio con el comando *git pull*.

```
regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (main)
$ git pull origin main
From https://github.com/albiregi/git-test
* branch          main      -> FETCH_HEAD
Already up to date.
```

Supongamos que la rama en la que estamos trabajando es *main*. Esto nos indica que no ha habido ningún cambio, en el caso de que los hubiera, los habríamos descargado a nuestro repositorio local.

## 10.10 Git Branch

En un entorno colaborativo, es habitual que varios desarrolladores compartan y trabajen sobre el mismo código fuente. Mientras que algunos desarrolladores corregirán errores, otros implementarán nuevas funciones, etc. Con tantas cosas sucediendo, es necesario que exista un sistema para administrar diferentes versiones de la misma base de código.

La ramificación permite a cada desarrollador ramificarse a partir del código base original y aislar su trabajo de los demás. También ayuda a Git a fusionar fácilmente versiones más adelante.

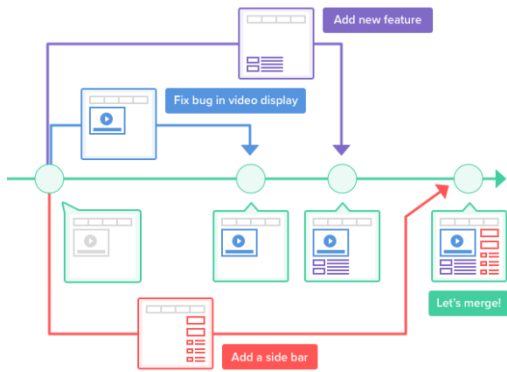
### 10.10.1 ¿Qué es una rama de Git?

Una rama de Git es esencialmente una línea de desarrollo independiente. Puede aprovechar la bifurcación cuando trabaje en nuevas funciones o corrija errores porque aísla su trabajo del de otros miembros del equipo.

Una rama de git es una línea de desarrollo independiente tomada del mismo código fuente.

Se pueden fusionar diferentes ramas en cualquier rama siempre que pertenezcan al mismo repositorio.

El siguiente diagrama ilustra cómo el desarrollo puede llevarse a cabo en paralelo usando ramas.



Rama verde → main    Rama roja → nuevo-feature

Tenemos un equipo que están trabajando en la rama main. Lo que vamos a hacer es crear una nueva rama llamada nuevo-feature para crear una mejora al proyecto común, después hacemos un merge, para que todos los cambios queden en la rama main donde todos estamos trabajando.

```
regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (main)
$ git branch nuevo-feature
```

git branch nuevo-feature

Creamos la nueva rama, observamos que estamos trabajando en la rama main.

Nos situamos en la nueva rama.

```
regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (main)
$ git checkout nuevo-feature
Switched to branch 'nuevo-feature'

regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (nuevo-feature)
```

git checkout nuevo-feature

Comando para ver las ramas que tenemos creadas localmente.

```
regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (nuevo-feature)
$ git branch
main
* nuevo-feature
```

git branch

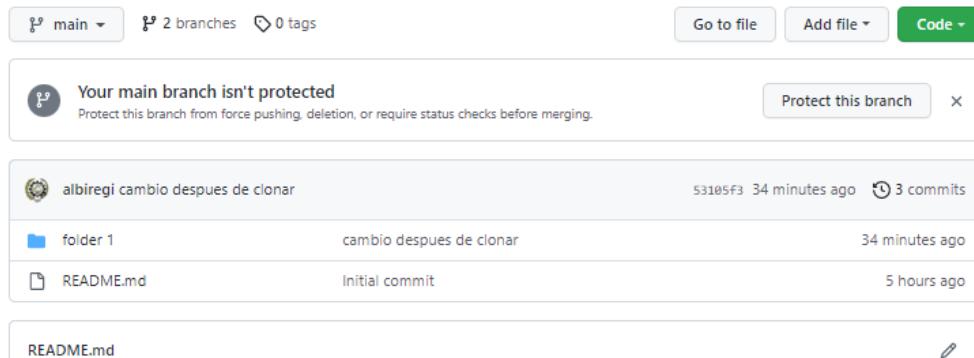
Para ver las ramas locales y remotas

```
regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (nuevo-feature)
$ git branch -a
main
* nuevo-feature
remotes/origin/HEAD -> origin/main
remotes/origin/main
```

git branch -a



Si miramos en github, veremos que la rama nuevo-feature no está aún, ya que no hemos hecho ningún *push*.

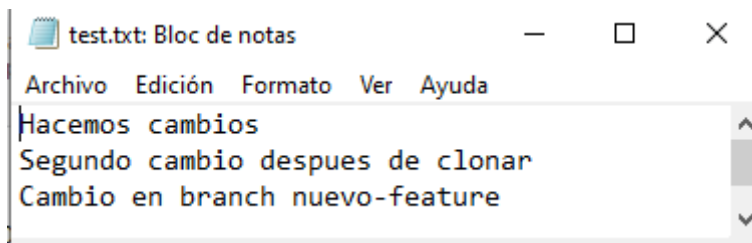


Vemos que tenemos los mismos archivos que en la rama main.

```
regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (nuevo-feature)
$ ls
README.md 'folder 1/'
```

Realizamos una modificación en el archivo test.txt del directorio 'folder 1'.

Podemos comprobar que este cambio queda reflejado localmente.



Ahora hay que añadir este nuevo cambio al Staging área y hacemos commit

```
regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (nuevo-feature)
$ git add -A

regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (nuevo-feature)
$ git commit -m "Cambio en nueva brach"
[nuevo-feature ea48fc3] Cambio en nueva brach
1 file changed, 2 insertions(+), 1 deletion(-)
```

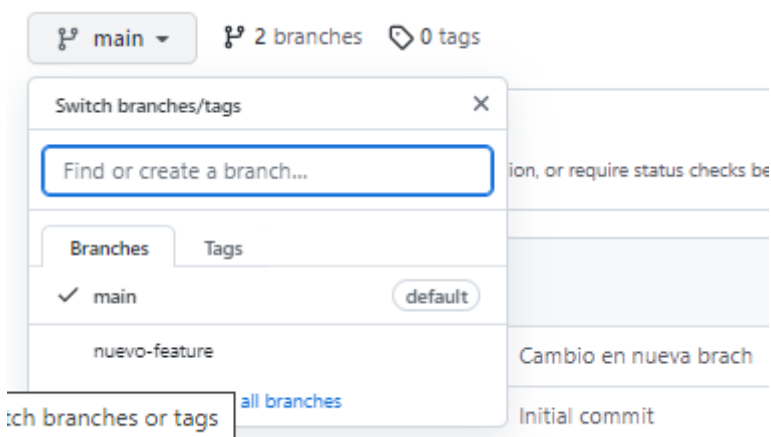
```
regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (nuevo-feature)
$ git status
On branch nuevo-feature
nothing to commit, working tree clean
```

## 10.11 git push

Ya solo nos queda reflejar estos cambios en el repositorio remoto, pero en la rama *nuevo-feature*.

```
regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (nuevo-feature)
$ git push origin nuevo-feature
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 426 bytes | 426.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'nuevo-feature' on GitHub by visiting:
remote:   https://github.com/albiregi/git-test/pull/new/nuevo-feature
remote:
To https://github.com/albiregi/git-test.git
 * [new branch]   nuevo-feature -> nuevo-feature
```

git push origin nuevo-feature



Observad que el contenido del archivo test.txt de la rama nuevo-feature es distinto al de la rama main.

Podemos observar que la rama nuevo-feature, ya está en el repositorio remoto.

```
regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (nuevo-feature)
$ git branch -a
main
* nuevo-feature
remotes/origin/HEAD -> origin/main
remotes/origin/main
remotes/origin/mama
remotes/origin/nuevo-feature
```

```
git brach -a
```

## 10.12 git pull

Lo último que nos queda es combinar los cambios de la rama *nuevo-feature* para que todos los cambios queden en la línea principal, *main*. Para ello, lo primero que debemos realizar es cambiarnos a la rama principal, *main*.

```
regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (nuevo-feature)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (main)
$
```

```
git checkout main
```

Asegurarnos que no ha habido ningún cambio en la rama *main* y ya podemos hacer el pull.

```
regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (main)
$ git pull origin main
From https://github.com/albiregi/git-test
* branch      main      -> FETCH_HEAD
Already up to date.
```

```
git pull origin main
```

## 10.13 git merge

Vamos utilizar el comando merge para validar las branch con las que se ha hecho un merge. Como vemos ahora mismo no hay ninguna.

```
regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (main)
$ git branch --merged
* main
```

```
git branch --merged
```

Hacemos el merge en main de la rama nuevo-feature.

```
regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (main)
$ git merge nuevo-feature
Updating 53105f3..ea48fc3
Fast-forward
 folder 1/test.txt | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)
```

```
git merge nuevo-feature
```

```
regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (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
```

Los cambios están en local, pero no en remoto.

Ultimo paso, hacer un push para reflejar los cambios locales en remoto

```
regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (main)
$ git push origin main
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/albiregi/git-test.git
 53105f3..ea48fc3  main -> main
```

```
git push origin main
```

Observamos que los cambios están reflejados remotamente en main.



## 10.14 Eliminar un branch

Una vez que se ha realizado el merge, lo aconsejable es eliminar la rama para evitar sobrecargar nuestro repositorio.

Vemos los branch que tenemos.

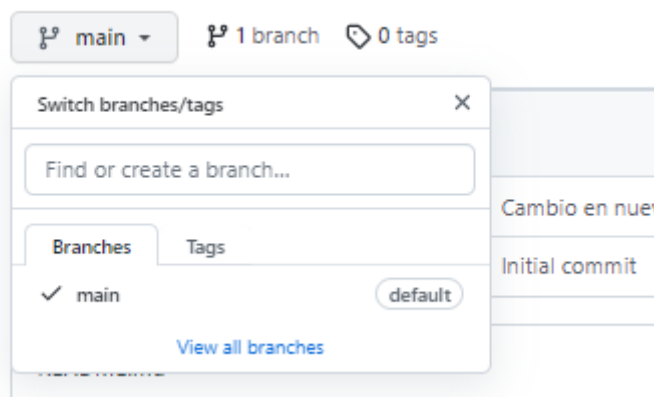
```
regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (main)
$ git branch -a
* main
  nuevo-feature
  remotes/origin/HEAD -> origin/main
  remotes/origin/main
  remotes/origin/nuevo-feature
```

Vamos a borrar la rama tanto local como remota nuevo-feature.

```
regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (main)
$ git push origin --delete nuevo-feature
To https://github.com/albiiregi/git-test.git
- [deleted]          nuevo-feature
```

```
git push origin --delete nuevo-feature
```

Observamos que la rama remota nuevo-feature ha sido borrada.



```
regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (main)
$ git branch -a
* main
nuevo-feature
remotes/origin/HEAD -> origin/main
remotes/origin/main
```

Nos queda eliminar la rama a nivel local.

```
regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (main)
$ git branch -d nuevo-feature
Deleted branch nuevo-feature (was ea48fc3).

regina@DESKTOP-3FF7K39 MINGW64 /d/git/git-test (main)
$ git branch -a
* main
remotes/origin/HEAD -> origin/main
remotes/origin/main
```

```
git branch -d nuevo-feature
```

## 11 Bibliografía

<https://backlog.com/git-tutorial/using-branches/>  
<https://www.youtube.com/watch?v=0tx9PkKSHQo>

Albin Rodriguez