

Apuntes de Git y GitHub

¿Qué es Git?

Git es un sistema de control de versiones distribuido.

Permite registrar los cambios realizados en un proyecto, volver a versiones anteriores, trabajar en equipo y fusionar código fácilmente.

Ciclo de vida de los archivos en Git

Git maneja **cuatro estados principales** en los que puede estar un archivo:

| Estado | Descripción |
|------------------|---|
| Untracked | El archivo existe en tu carpeta, pero Git no lo está controlando. |
| Tracked | El archivo ya fue añadido al control de versiones. |
| Staged | El archivo ha sido preparado para el siguiente commit (<code>git add</code>). |
| Committed | El archivo ya está guardado en la base de datos de Git (<code>git commit</code>). |

Fases principales

```
[ Working Directory ] → git add → [ Staging Area ] → git commit → [ Local Repository ] → git push → [ Remote Repository (GitHub) ]
```

- **Working Directory:** tus archivos reales.
 - **Staging Area (o index):** una "zona intermedia" donde eliges qué cambios se guardarán en el próximo commit.
 - **Local Repository:** historial de commits en tu máquina.
 - **Remote Repository:** copia del repo en un servidor (GitHub, GitLab, etc.).
-

Configuración inicial

```
git config --global user.name "Tu Nombre"
git config --global user.email "tuemail@example.com"
git config --list
git config --global --edit # Abrir configuración global con editor de texto.
git config --global core.editor "nano" # Configurar editor por defecto (cambia nano por vscode o el editor de tu elección)
```

Crear y gestionar repositorios

```
git init                # Inicia un nuevo repositorio en la carpeta
actual
git clone <URL>         # Clona un repositorio existente (de GitHub u
otro servidor)
```

Flujo de trabajo completo

1 Working Directory → Staging Area

Añadimos archivos a la zona de preparación.

```
git add <archivo>      # Añadir un archivo
git add .               # Añadir todos los cambios
git status              # Ver el estado actual de los archivos en el
repositorio
```

2 Staging Area → Repository (commit)

Guardamos los cambios en la base de datos local de Git.

```
git commit -m "Mensaje del commit" # Confirmar los cambios preparados.
También podemos hacer git commit sin el parámetro -m, pero en lugar de
hacer el commit con el mensaje directamente nos abrirá el editor por
defecto que hayamos configurado.
```

3 Repository → Remote Repository

Subimos nuestros commits al servidor remoto (GitHub).

```
git push origin main    # Subir cambios a la rama main
git push --set-upstream origin main # Primera vez, vincula la rama local
con la remota
```

4 Remote Repository → Repository

Descargamos o actualizamos los cambios remotos.

```
git pull origin main    # Descargar y fusionar cambios
git fetch               # Descargar sin fusionar (solo
actualiza la info remota)
```

✂ Modificar y deshacer cambios

```
git restore --staged <archivo> # Quitar un archivo del área de stage
git restore <archivo>          # Deshacer cambios en el working directory
git revert <id_commit>         # Crear un commit que revierta otro
git reset --hard <id_commit>   # Volver el repo a un commit anterior
(borra cambios)
```

🌿 Ramas (branches)

```
git branch                    # Ver ramas

git branch <nombre>          # Crear una rama

git switch <nombre>          # Cambiar de rama
git checkout <nombre>

git switch -c <nombre>       # Crear y cambiar a la vez
git checkout -b <nombre>

git merge <nombre>           # Fusionar una rama con la actual. No
genera commit de merge, solo mueve el puntero de la rama hacia delante
git merge <nombre> --no-ff    # Fusiona una rama con la actual pero
forzando dejar un commit de merge. --no-ff significa "No fast forward".
git merge <nombre> --allow-unrelated-histories # Fusionar una rama sin
cambios relacionados con la actual
git branch -d <nombre>       # Borrar una rama
```

☁ Remotos (GitHub y otros)

```
git remote add origin <URL HTTP> / <URL SSH> # Conectar el repo local a uno
remoto
git remote -v                                # Ver los remotos configurados
git push                                     # Subir cambios
git pull                                    # Bajar y fusionar cambios
git fetch                                   # Descargar sin fusionar
```

🔍 Comparar y revisar cambios

```
git diff                                # Ver diferencias entre archivos
git diff <commit1> <commit2>          # Comparar commits
git log --oneline --graph --all        # Ver el historial con ramas
git blame <archivo>                   # Ver quién cambió cada línea
```

Ignorar archivos

Archivo `.gitignore`:

```
# Carpetas
node_modules/
dist/

# Archivos temporales
*.log
.env
```

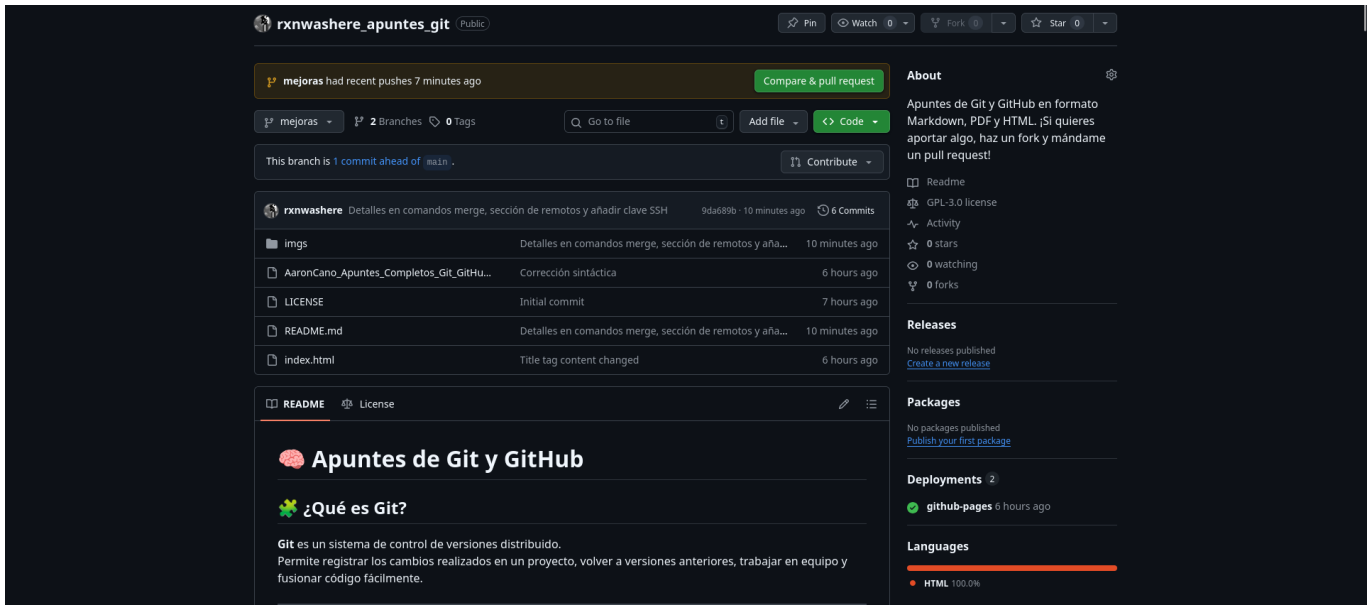
Flujo de trabajo típico

```
git clone <url>
git checkout -b feature/nueva-funcionalidad
# (editar archivos)
git add .
git commit -m "Añadida nueva funcionalidad"
git push origin feature/nueva-funcionalidad
# Crear Pull Request en GitHub
```

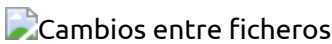
EJEMPLO DE PULL REQUEST:

Cuando entremos a nuestro repositorio desde GitHub a la nueva rama veremos un aviso que nos informará sobre los commits por delante o por detrás que está respecto a `main`, para este ejemplo estará un commit por delante.

También veremos una notificación avisando de las ramas que han tenido pushes recientemente:

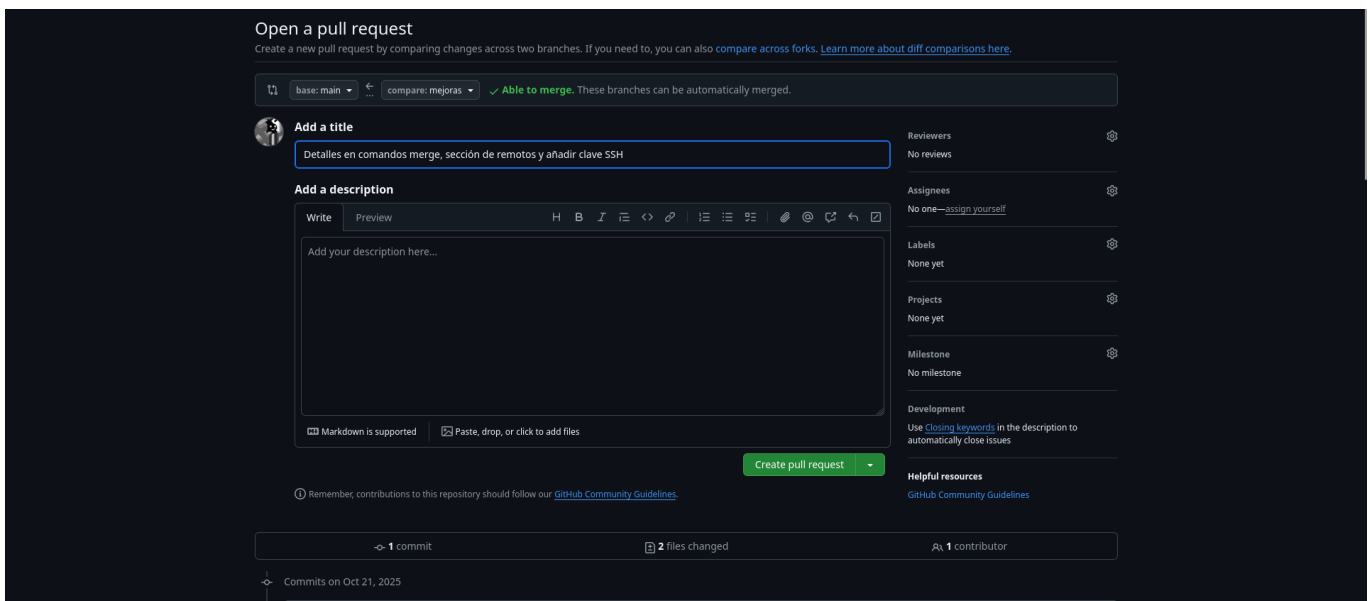


Presionaremos **Compare & Pull Request** para comprobar los cambios y fusionar ambas ramas:



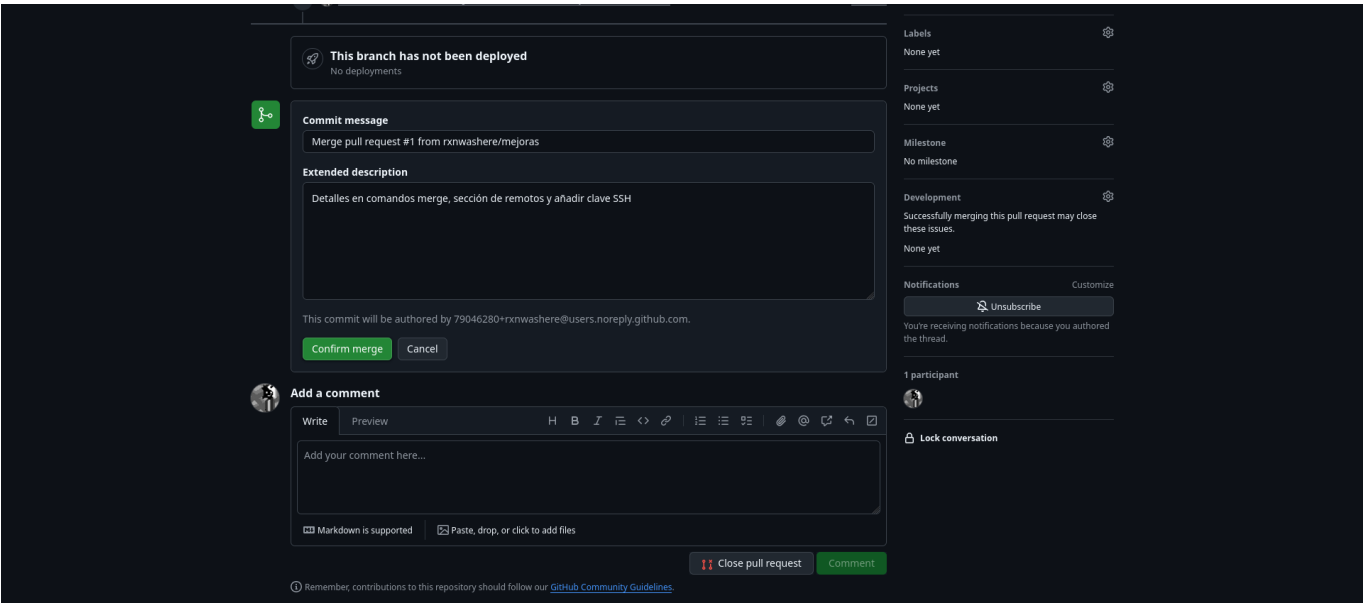
Cambios entre ficheros

Añadimos un nombre a nuestra request (puede coger el nombre de nuestro último commit en la rama a fusionar):

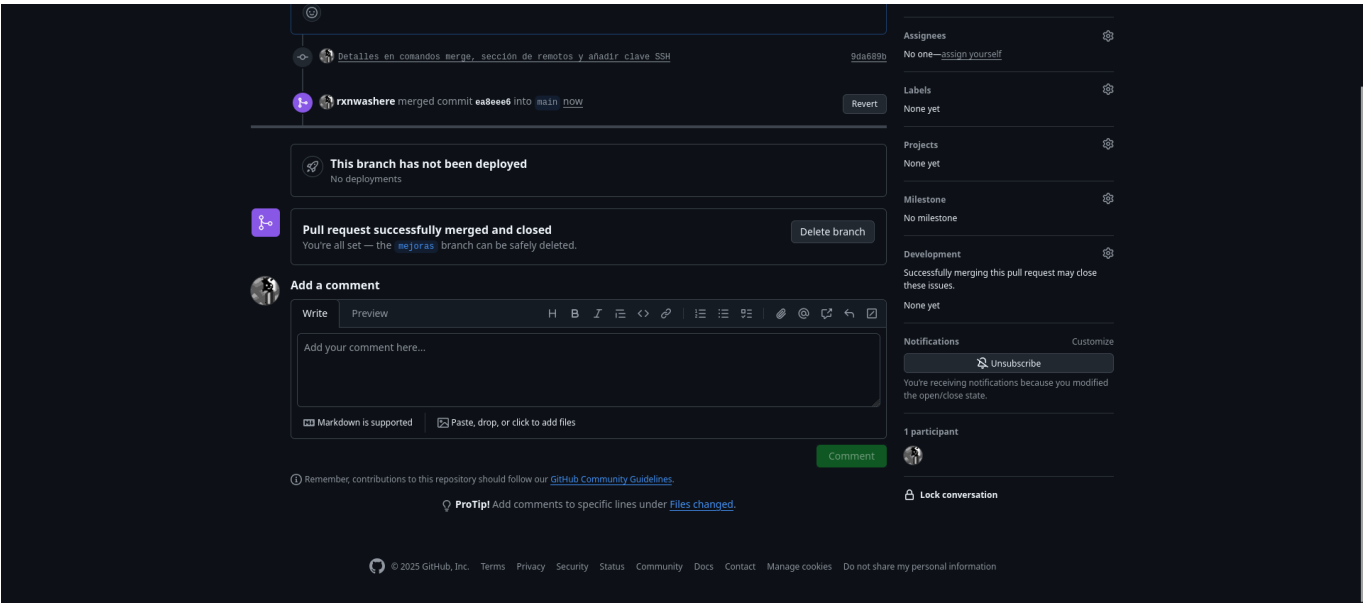


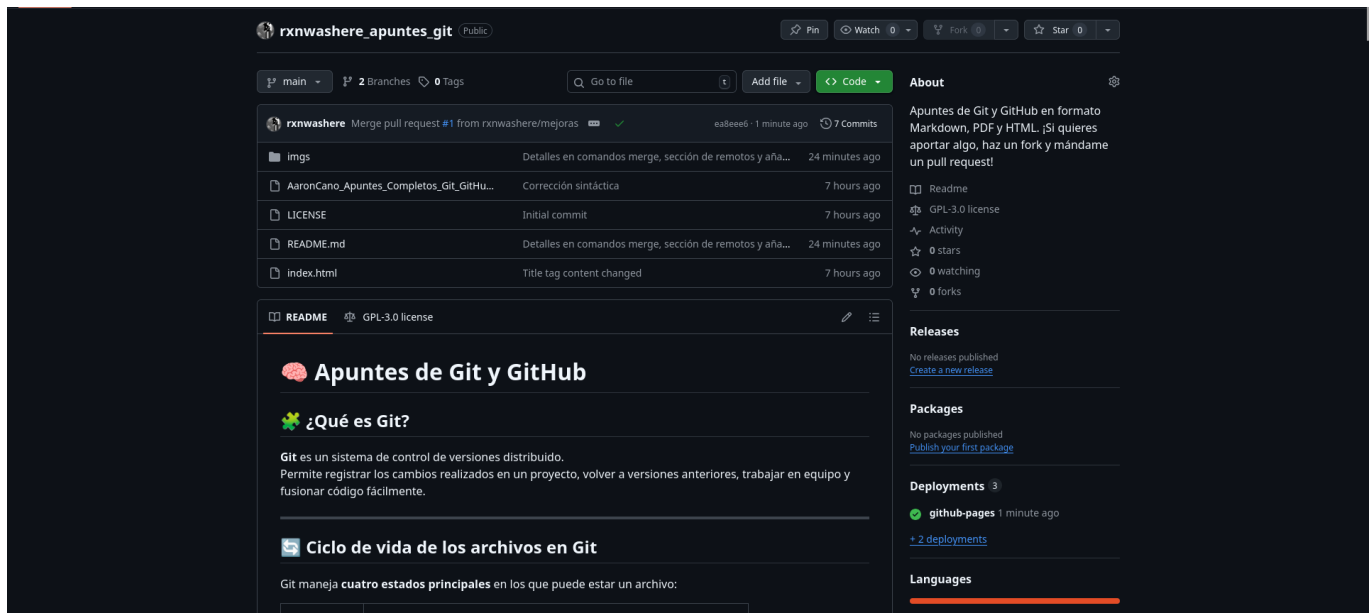
Si nos fijamos arriba podremos observar que nos informa que está listo para fusionar automáticamente debido a que no hay conflictos (**Able to merge**)

Al abrir la pull request si no hay conflictos como en este caso podremos hacer un **merge** con main:



Cuando se fusione, la pull request será cerrada y podremos eliminar la rama si lo deseamos:





Comandos útiles

```
git log --oneline          # Ver historial corto
git reflog                 # Ver todos los movimientos de HEAD
git clean -f               # Borrar archivos no rastreados
```



SSH con GitHub

```
ssh-keygen -t ed25519 -C "tuemail@example.com"
cat ~/.ssh/id_ed25519.pub
```

Luego añades el contenido de la clave a una nueva desde GitHub, podrás encontrar el menú en:




Settings (configuración de tu cuenta de GitHub) --> SSH and GPG Keys

SSH keys

New SSH key

This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.

Authentication keys




RXN SSH Key

SHA256 : a102hsIy6dqMouXmFhnwAD43UVQAEapJaAjKVLe5Q94

Added on Oct 1, 2025

Last used within the last week — Read/write

Delete



RXN Laptop

SHA256 : uA2JW5pw19obLVJkYnmhmtIDLZqFtkg23IXPXhgG7Ts

Added on Oct 14, 2025

Last used within the last 2 weeks — Read/write

Delete

Check out our guide to [connecting to GitHub using SSH keys](#) or troubleshoot [common SSH problems](#).

GPG keys

New GPG key

There are no GPG keys associated with your account.

Learn how to [generate a GPG key and add it to your account](#).


Vigilant mode

☐ Flag unsigned commits as unverified

This will include any commit attributed to your account but not signed with your GPG or S/MIME key.

Note that this will include your existing unsigned commits.

[Learn about vigilant mode.](#)

 Recursos recomendados

- [Pro Git Book \(en español\)](#)
- [Git Cheatsheet \(GitHub\)](#)
- [Learn Git Branching \(interactivo\)](#)
- [Documentación oficial de Git](#)

Hecho por [Aarón Cano \(rxnwashere\)](#) y revisado con ChatGPT

8 / 8