

# Taller de GIT

Pablo Martínez López

Full On Net

19 de septiembre de 2025



1 ¿Qué es GIT?

2 Básicos de GIT

3 Ramas

4 Metodología de trabajo GitFlow

5 Herramientas gráficas de apoyo

6 Fin del taller

# ¿Qué es Git?

- Git es una herramienta de control de versiones.
- Nos permite un seguimiento del trabajo y sus versiones.
- Combina trabajo local y trabajo remoto.
- Utilizado generalmente para trabajo en equipo, aunque puede ser utilizado de forma individual.



# ¿Qué NO es Git?

- Es común confundir Git con GitHub, GitLab...etc
- Git es el software que nos permite trabajar con los remotos.
- GitHub u otras herramientas similares son interfaces web que permiten visualizar los repositorios remotos.



1 ¿Qué es GIT?

2 Básicos de GIT

3 Ramas

4 Metodología de trabajo GitFlow

5 Herramientas gráficas de apoyo

6 Fin del taller

# Descargando Git

- Para trabajar con Git en Windows, utilizaremos **VSCode**. También se puede trabajar con otro modelo de terminal: CMD o PowerShell.
- Descargaremos git de: **<https://git-scm.com/downloads/win>**

# Iniciando el repositorio

Vamos a crearnos un repositorio remoto y trasladarlo a local.

Para el local simplemente crearemos un nuevo directorio.

Para el remoto utilizaremos **GitHub**.

- **git clone** (HTTPS vs SSH)

- **origin**

## .git y .gitignore

- Al realizar git init o git clone se genera (oculta) una carpeta llamada **.git**.
- Esta carpeta contiene todo lo referente a la configuración del repositorio, tal como la url de origin o la posición del HEAD.
- El archivo **.gitignore** es un archivo que bien viene creado o bien lo creamos nosotros.
- En él escribimos nombres de ficheros o directorios que queremos que git ignore a la hora de realizar merges, commits...etc.
- Este archivo respeta la sintaxis de Unix (\*.java).



# Esquema general

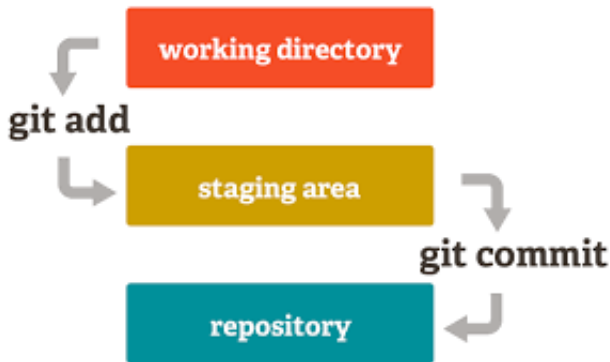


Figura: Esquema general

# Add

- Con **git add** añadimos archivos a nuestro espacio de trabajo.
- Con **git add -remove** eliminamos archivos añadidos.
- Podemos utilizar espacios de nombres u opciones de comando:  
git add \*.json ó git add --all.
- Ver: git - -help.

# Commit

- Los commits añaden los archivos añadidos (add) al *stage area*.
- Llevan un mensaje que explican los cambios que se han hecho desde el último commit.
- Con **git commit -m "mensaje"** introducimos un pequeño mensaje.
- Idealmente, **los commits deberían de ser atómicos y concisos**.

# Status y log

- Con **git status** podemos ver el estado del repositorio actual.
- Nos proporciona información sobre la rama, el área de trabajo, el *stage area*...etc.
- Con **git log** podemos ver el historial de *commits* en la rama actual.
- En breves veremos qué son las **ramas**.

# Revert y Reset

- Con **git revert** realizamos un *commit* que revierte el anterior *commit* realizado.
- Con **git reset** reseteamos o deshacemos el punto donde nos encontremos actualmente. Esto lo hace de varias formas dependiendo el argumento del comando. Hay que tener cuidado.
  - `git reset -soft` deshace el último commit.
  - `git reset -mixed` (por defecto) deshace el último commit **y archivos añadidos**.
  - `git reset -hard` deshace commit y add, y **MODIFICA EL ESPACIO DE TRABAJO**.
  - Si fuese su primer commit, **BORRARÍA EL ARCHIVO**.

# Push

- Con **git push**, "empujamos" los archivos del *stage area* del repositorio local al remoto.
- La sintaxis del comando es: `git push 'origin' <rama>`.
- Un push puede llevar varios commits.

# Pull y Fetch

- Con **git pull**, "nos traemos" los cambios del servidor remoto a nuestro espacio de trabajo local.
- La sintaxis del comando es: `git pull origin <rama>`.
- **Git fetch** es una operación intermedia incluida en el pull que sirve para traer los cambios del remoto al *staging area*.

- 1 ¿Qué es GIT?
- 2 Básicos de GIT
- 3 **Ramas**

- 4 Metodología de trabajo GitFlow
- 5 Herramientas gráficas de apoyo
- 6 Fin del taller



## ¿Qué son las ramas?

- Las ramas son **BIFURCACIONES** de nuestro proyecto.
- Siguen un esquema de árbol. No son líneas paralelas, contienen el código de la rama padre.
- Nos sirven para dividir el trabajo, tener una visualización clara del proyecto y para trabajar *de forma segura*.
- La rama principal se denomina **main**.



# Branch y checkout

- Son las operaciones principales de las ramas.
- Con **git branch** creamos una rama (seguido del nombre de la rama) o vemos en que rama estamos (sin argumentos).
- Con **git checkout** nos movemos de rama. La opción **git checkout -b 'rama'** nos permite crear una rama e instantáneamente movernos a ella.

# Merge teórico

- ¡CUESTIÓN MÁS DELICADA DE GIT!
- Consiste en juntar una rama bifurcada con un padre.
- ¿Problema? → Un padre puede tener muchos hijos (ramas) y al volver a juntarse cada uno puede haber tratado las cosas de forma distinta.
- Se genera lo que se denomina **conflicto**.
- A pesar de que puede hacerse por comandos, las interfaces web como **GitHub** o **GitLab** permiten hacerlo también de forma más visual.
- El proceso de unir ramas de forma controlada se conoce como **Pull Request**.

# Pull Request I

Cuando queramos incorporar cambios de nuestra rama a una rama padre, crearemos una Pull Request.

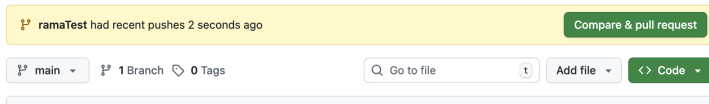


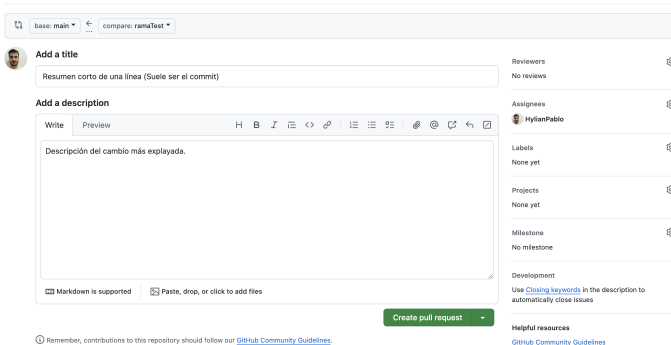
Figura: Notificación tras subir el cambio a nuestra rama.

## Pull Request II

Escribiremos un **mensaje descriptivo**, nos asignaremos la PR y de ser necesario, completaremos la descripción.

### Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#). [Learn more about diff comparisons here](#).



base: main ← compare: ramaTest

**Add a title**

Resumen corto de una línea (Suele ser el commit)

**Add a description**

Write Preview H B I

Descripción del cambio más explayada.

Markdown is supported ☒ Paste, drop, or click to add files

**Reviewers**  
No reviews

**Assignees**  
HylianPablo

**Labels**  
None yet

**Projects**  
None yet

**Milestone**  
No milestone

**Development**  
Use [Closing keywords](#) in the description to automatically close issues

**Helpful resources**  
[GitHub Community Guidelines](#)

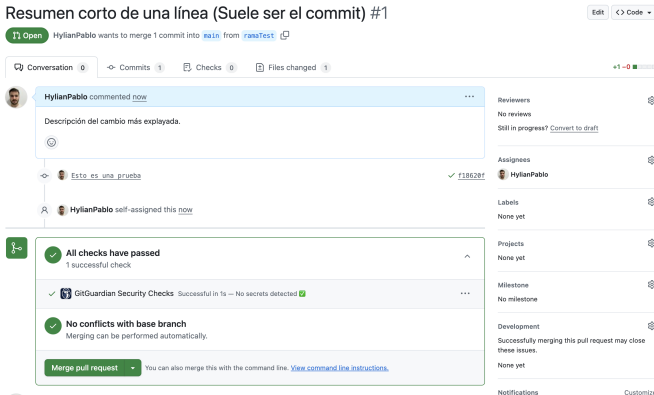
Create pull request

Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).

Figura: Creando la Pull Request.

# Pull Request III

Cuando la Pull Request pase la integración continua (*GitHub Actions*), se podrá mergear.



The screenshot displays a GitHub Pull Request (PR) titled "Resumen corto de una línea (Suele ser el commit) #1". The PR is created by HylianPablo and targets the 'main' branch from the 'ramaTest' branch. The interface shows a green "Open" button and a "Code" dropdown. Below the PR title, there are tabs for "Conversation" (0), "Commits" (1), "Checks" (0), and "Files changed" (1). A comment by HylianPablo states "Descripción del cambio más explorada." and "Esto es una prueba." A self-assignment comment is also visible. On the right side, the "Reviewers" section shows "No reviews" and a "Convert to draft" link. The "Assignees" section lists HylianPablo. The "Labels" section is empty. The "Projects" section is empty. The "Milestone" section is empty. The "Development" section shows a message: "Successfully merging this pull request may close these issues." The "Notifications" section is empty. At the bottom, a green box indicates "All checks have passed" with "1 successful check". Below this, a green checkmark and the text "GitGuardian Security Checks: Successful in 1s - No secrets detected" are shown. Another green checkmark and the text "No conflicts with base branch: Merging can be performed automatically." are shown. A green button labeled "Merge pull request" is visible, along with a link to "View command line instructions".

Figura: Pull Request lista para merge.



1 ¿Qué es GIT?

2 Básicos de GIT

3 Ramas

4 Metodología de trabajo GitFlow

5 Herramientas gráficas de apoyo

6 Fin del taller

# Metodología básica

De nuevo recalcamos que una de las características que hacen a Git tan potente es su facilidad para organizar el trabajo. Pero tenemos que poner de nuestra parte.

Vamos a diferenciar varias ramas "*clásicas*" que nos permiten trabajar de forma eficiente:

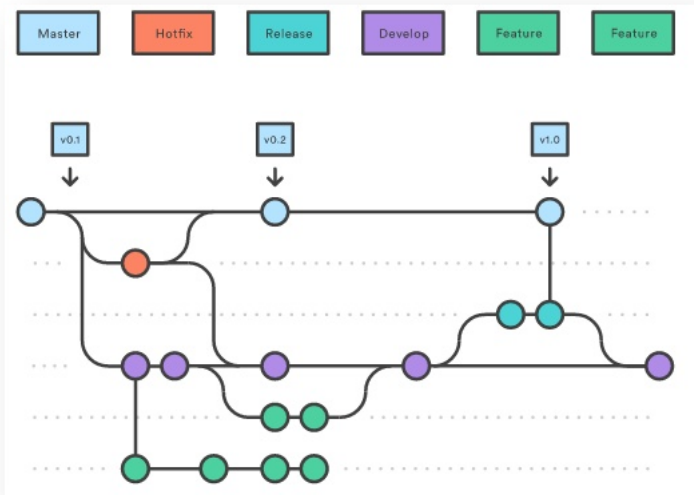
- **main** - Rama principal donde está todo en orden y estable.
- **develop** - Rama principal del desarrollo.
- **rama por miembro / cambio** - Subdivisiones de develop.
- **release** - Estado pasado congelado de main para entregas.



## Otras buenas prácticas

- Crear una rama por característica o issue en desarrollo a partir de develop o sub-ramas de esta. Se denominan **feature branches**.
- Utilizar commits significativos y autoexplicativos.
- Utilizar etiquetas, descripciones cuando sea necesario... Hacer que a futuro todo sea más sencillo.

## Ejemplo visual





1 ¿Qué es GIT?

2 Básicos de GIT

3 Ramas

4 Metodología de trabajo GitFlow

5 Herramientas gráficas de apoyo

6 Fin del taller



# SourceTree

- Existen herramientas de terceros gratuitas que ayudan a realizar todas estas tareas a través de una interfaz gráfica en vez de a través de línea de comandos.
- Pueden bastante útiles para ayudarnos a ver los cambios si hemos trabajado sobre muchos ficheros.
- Existen muchas más: GitHub Desktop, GitKraken...etc



1 ¿Qué es GIT?

2 Básicos de GIT

3 Ramas

4 Metodología de trabajo GitFlow

5 Herramientas gráficas de apoyo

6 Fin del taller



# Dudas y preguntas

Las diapositivas, como complemento teórico las encontraréis aquí:  
**<https://github.com/HylianPablo/TallerGitFONTelefonica>**