

# Guía GIT

October 28, 2024

## 1 ¿Qué es Git?

Git es un sistema de control de versiones distribuido, usado principalmente para rastrear cambios en archivos y colaborar en proyectos de software. Permite que múltiples desarrolladores trabajen en el mismo proyecto sin sobrescribir el trabajo de los demás. Al ser distribuido, cada copia de un proyecto tiene todo el historial de versiones, lo que lo hace muy robusto.



Figure 1: Git

## 2 Herramientas comunes para GIT

Existen varios servicios que proporcionan repositorios Git para alojar proyectos, colaborar y realizar control de versiones de código. A continuación, se detallan algunos de los repositorios Git más conocidos y utilizados:

- **GitHub**

Es la plataforma más popular para alojar repositorios Git. Ofrece características colaborativas como pull requests, revisiones de código, problemas (issues), y GitHub Actions para CI/CD.

- Repositorios públicos y privados.
- Integraciones con CI/CD a través de GitHub Actions.
- GitHub Pages para alojar sitios estáticos.
- Amplia comunidad de código abierto.
- Soporte para wikis y seguimiento de problemas.

- **GitLab**

GitLab es una plataforma de DevOps que incluye Git para control de versiones, junto con muchas otras funcionalidades integradas como CI/CD, gestión de proyectos y seguridad.

- Repositorios públicos y privados.

- CI/CD integrado mediante GitLab CI.
- Seguridad avanzada con escaneo de vulnerabilidades y análisis estático de código.
- Gestión completa de ciclo de vida de software (planificación, desarrollo, revisión y despliegue).
- Disponible en una versión autohospedada (GitLab CE) y en la nube (GitLab.com).

- **Bitbucket**

Bitbucket es una plataforma de repositorios Git que se integra estrechamente con Jira y otros productos de Atlassian. Está diseñada para equipos profesionales y empresariales.

- Repositorios privados gratuitos para equipos pequeños.
- Integración con Jira para seguimiento de proyectos.
- Soporte para Git y Mercurial (hasta junio de 2020, Mercurial ya no es soportado).
- Bitbucket Pipelines para CI/CD integrado.
- Funcionalidades de seguridad como control de acceso y revisiones de código.

- **AWS CodeCommit**

AWS CodeCommit es un servicio de repositorios Git totalmente gestionado en la nube de Amazon Web Services (AWS). Está diseñado para integrarse con otros servicios de AWS, proporcionando una solución escalable para equipos que ya utilizan el ecosistema de AWS.

- Repositorios Git privados gestionados en AWS.
- Alta disponibilidad y escalabilidad.
- Integración con otros servicios de AWS como CodeBuild y CodeDeploy.
- Sin límite de tamaño para los repositorios.
- Control detallado de acceso mediante AWS Identity and Access Management (IAM).

- **Azure Repos**

Azure Repos es parte de Azure DevOps de Microsoft, que ofrece repositorios Git privados con integraciones profundas en el flujo de trabajo de DevOps.

- Repositorios Git privados y gratuitos con soporte ilimitado.
- Integración con Azure Pipelines para CI/CD.
- Revisiones de código colaborativas mediante pull requests.
- Soporte para grandes equipos y proyectos con escalabilidad.
- Integración con otras herramientas de Azure DevOps, como Boards y Pipelines.

### 3 Instalación de Git

Puedes descargar Git desde el siguiente **enlace** para cualquier dispositivo. Y para su instalación usa los siguientes comandos

- En Linux: `sudo apt install git`
- En MacOS: `brew install git`
- En Windows: Usa el instalador desde la página oficial.

### 4 Conceptos clave de Git

Antes de entrar en los comandos básicos, hay algunos conceptos que debes conocer.

#### Repositorio

Un repositorio (repo) es donde se almacena tu proyecto y su historial de cambios. Hay dos tipos de repositorios en Git:

- Local: El repositorio que tienes en tu máquina.
- Remoto: Un repositorio que está en un servidor, como GitHub, GitLab, etc.

## **Commit**

Un commit es un registro de los cambios realizados en los archivos del proyecto. Cada commit es como un “punto de control” al que puedes volver si es necesario.

## **Branch (Rama)**

Una rama es una versión del proyecto que evoluciona independientemente de las demás ramas. La rama principal por defecto se llama ‘main’ o ‘master’.

## **Staging Area**

El área de preparación (staging area) es donde se colocan los archivos que quieres incluir en tu próximo commit.

## **5 Creación del repositorio el servicio Git**


Antes de agregar los elementos al servicio git, deberá crear el repositorio. Algunos servicios de git permiten repositorios privados gratuitos. En la Figura 2 se ve la información que se solicita para crear el repositorio en GitHub

# Create a new repository

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

*Required fields are marked with an asterisk (\*).*

Owner \*

 jprestrepou ▾

Repository name \*

/

Great repository names are short and memorable. Need inspiration? How about **animated-palm-tree** ?

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:



**Add a README file**

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

Add .gitignore

.gitignore template: **None** ▾

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

Choose a license

License: **None** ▾

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



You are creating a public repository in your personal account.

Create repository

Figure 2: GitHub

Esto creará un repositorio, en este caso en GitHub, el cual te proporcionará una dirección HTTPS la cual permite que el repositorio sea asequible. Esta dirección en el ejemplo de la figura 3

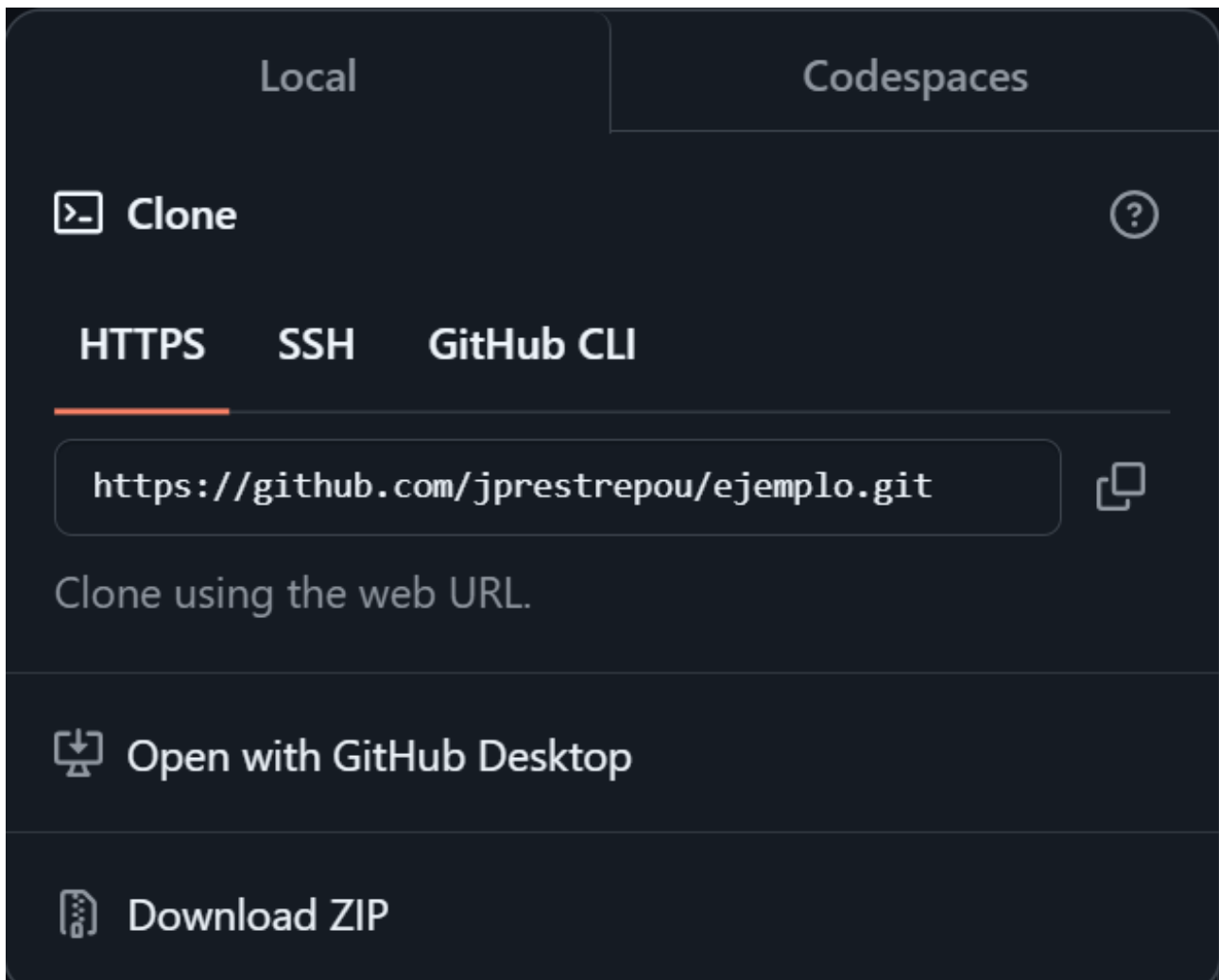


Figure 3: Dirección HTTPS

## 6 Configuración inicial

Después de instalar Git, es importante configurarlo con tu nombre y correo electrónico, que se asociarán a los commits que realices.

```
1 git config --global user.name "TuNombre"
```

```
1 git config --global user.email "tuemail@dominio.com"
```

Verifica la configuración con:

```
1 git config --list
```

Obtendrás algo similar a lo de la 4

```
jupa_@JPRU MINGW64 ~/Desktop
$ 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/etc/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
user.name=jprestrepou
user.email=jprestrepou@gmail.com
```

Figure 4: Resultado de aplicar el comando `git config --list`

## 7 Comandos básicos de Git

### Crear un repositorio

Si ya tienes un proyecto existente y quieres comenzar a rastrear sus cambios con Git, puedes inicializar un nuevo repositorio.

```
1 git init
```

Esto crea un repositorio Git en tu carpeta actual.

### Clonar un repositorio

Para descargar un repositorio remoto en tu máquina local:

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

Ejemplo:

```
1 git clone <https://github.com/jprestrepou/inteligenciaArtificial2.git>
```

### Ver el estado del repositorio

Para ver el estado de tus archivos y saber si hay cambios no rastreados o que aún no han sido confirmados:

```
1 git status
```

Se debería ver información del repositorio y si hay cambio en el mismo, como se ve en la Figura 5

```
jupa_@JPRU MINGW64 ~/Desktop/EjemploGit (master)
$ git status
On branch master
nothing to commit, working tree clean
```

(a) Uso del comando `git status` sin cambios en el repositorio

```
jupa_@JPRU MINGW64 ~/Desktop/EjemploGit (master)
$ git status
On branch master
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:   ejemplo.txt

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

(b) Uso del comando `git status` con cambios en el repositorio no sincronizados

Figure 5: Subfiguras usando el paquete subcaption

## Añadir archivos al área de preparación

Para mover archivos modificados al área de preparación y prepararlos para un commit:

```
1 git add <nombre-del-archivo>
```

Para añadir todos los archivos modificados:

```
1 git add .
```

## Hacer un commit

Una vez que los archivos están en el área de preparación, puedes hacer un commit con un mensaje descriptivo.

```
1 git commit -m "mensajeDescriptivoDeLosCambios"
```

## Ver el historial de commits

Para ver un historial de todos los commits realizados en el proyecto:

```
1 git log
```

y te mostrará algo como lo que se ve en la Figura 6

```
jupa_@JPRU MINGW64 ~/Desktop/EjemploGit (master)
$ git log
commit 42d4ef04bf09b819e50f71afe7509813ffd70473 (HEAD -> master, origin/master)
Author: jprestrepou <jprestrepou@gmail.com>
Date:   Mon Oct 21 14:03:24 2024 -0500

    ejemplo
```

Figure 6: Resultado de aplicar el comando `git config --list`

También puedes ver un historial simplificado:

```
1 git log --oneline
```

## 7.1 Sincronizar el repositorio remoto y local

Para poder sincronizar los cambios del Git local y el Git remoto debemos sincronizar ambos repositorios, para ello utilizamos la dirección HTML que me proporciona el Git creado en GitHub. Y utilizamos el siguiente comando.

```
1 git remote add origin https://github.com/usuario/proyecto.git
```

## Subir cambios a un repositorio remoto

Para enviar tus cambios locales a un repositorio remoto, como GitHub o GitLab, usa:

```
1 git push origin <rama>
```

Si estás en la rama “main” o “master”, sería:

```
1 git push origin main
```

## Obtener cambios de un repositorio remoto

Si trabajas con un equipo, es importante obtener los cambios más recientes del repositorio remoto:

```
1 git pull origin <rama>
```

Esto actualiza tu rama local con los cambios del repositorio remoto.

## Crear una nueva rama

Para crear una rama nueva y trabajar en ella sin afectar la rama principal:

```
1 git pull origin <rama>
```

“bash git branch ¡nombre-de-la-rama! “  
Cambiar a la nueva rama:

```
1 git pull origin <rama>
```

“bash git checkout ¡nombre-de-la-rama! “  
También puedes crear y cambiar a la nueva rama en un solo paso:

```
1 git pull origin <rama>
```

“bash git checkout -b ¡nombre-de-la-rama! “

## Combinar ramas (Merge)

Para unir los cambios de una rama a otra, primero cámbiate a la rama donde quieres que se fusionen los cambios (generalmente ‘main’), luego usa:

```
1 git pull origin <rama>
```

“bash git merge ¡nombre-de-la-rama-a-unir! “

## Eliminar una rama

Si ya no necesitas una rama, puedes eliminarla:

```
1 git pull origin <rama>
```

“bash git branch -d ¡nombre-de-la-rama! “

## 8 Manejo de conflictos

Los conflictos ocurren cuando dos personas modifican el mismo archivo en áreas superpuestas. Git te avisará cuando hay un conflicto. Para resolverlo:

1. Abre el archivo con el conflicto. 2. Verás algo como esto:

```
1 git pull origin <rama>
```

“bash ¡¡¡¡¡¡ HEAD Tu versión del archivo. ===== La otra versión del archivo. ¡¡¡¡¡¡ otra-rama “

3. Decide qué cambios conservar y edita el archivo. 4. Después de resolver el conflicto, añade el archivo y haz un commit.

```
1 git add <archivo-conflicto>
2 git commit -m "ResueltoConflictoEnArchivo"
```



## 9 Etiquetas (Tags)

Las etiquetas son útiles para marcar versiones específicas del proyecto, como una versión final.

Crear una etiqueta:

```
1 git tag <nombre-de-la-etiqueta>
```

Subir la etiqueta al repositorio remoto:

```
1 git push origin <nombre-de-la-etiqueta>
```

## 10 Colaboración con GitHub

### Creación de un repositorio en GitHub

1. Crea una cuenta en [GitHub](https://github.com/). 2. Crea un nuevo repositorio en GitHub. 3. Luego sigue las instrucciones que GitHub te da para enlazar tu repositorio local con GitHub. Generalmente, se verá algo así:

```
1 git remote add origin https://github.com/usuario/repo.git
2 git push -u origin main
```

### Fork y Pull Requests

Si deseas colaborar en un proyecto de otra persona en GitHub, puedes hacer un **fork** (copia del repositorio en tu cuenta), realizar tus cambios en tu copia, y luego enviar un **pull request** (solicitud de incorporación) para que tus cambios sean revisados y posiblemente integrados en el proyecto original.

## 11 Comandos avanzados

### Revertir un commit

Si cometiste un error y deseas revertir los cambios de un commit:

```
1 git revert <ID-del-commit>
```

### Resetear el repositorio a un estado anterior

Este comando es más drástico, ya que deshace commits de manera más permanente:

```
1 git reset --hard <ID-del-commit>
```