Github básico

Jorge Gael Lopez Figueras

April 3, 2025

Abstract

En este pequeño documento se vera como crear,subir, actualizar un repositorio desde git a github , para tener un entorno de trabajo bueno con compañeros.

1 Instrucciones en General

Lo primero que debemos hacer es crear nuestro repositorio en github y darle el nombre que deseemos.

Una vez hecho esto vamos a Git Bash, vamos a la carpeta en donde queramos clonar nuestro repositorio, cuando estemos en esa carpeta colocamos la siguiente instrucción:

$$\text{git clone} < URL >$$

el URL lo encontramos en github donde dice code y esta de verde.

Después si queremos subir algo lo añadimos a la carpeta y hacemos todos los cambio y modificaciones que queramos , ahora para subir estos cambios se hace con las siguientes instrucciones en orden.

git add $< nombrearchivo >$ "Ejemplo git add hola.pdf"
git commit -m "Aqui se escribe el mensaje por ejemplo: se añadio un ejemplo"

Ahora usamos el comando git status para ver si se subió el archivo , también podemos ocupar este comando haber como va nuestro repositorio. finalmente usamos el comando **git push origin master** y corroboramos en github.

Ahora si queremos cargar todo lo que subieron los demás(esto siempre se debe hacer) colocamos el comando **git pull origin main.**

Para subir una carpeta debemos crear la carpeta con el comando mkdir NombreDeLaCarpeta.

Despues debemos crear un archivo de texto para poder subir la carpeta.Con el sigueiente comando podemos crear un archivo de texto desde git Bash:

echo "Texto que va en el archivo" > ejemplo.txt

Después colocamos todas las instrucciones vistas anteriormente.Pero primero colocamos git status para ver los archivos que aun no se han subido.

2 Ramas

2.1 1. Verificar las ramas existentes

Antes de crear una nueva rama, es recomendable ver las ramas existentes en el repositorio con el siguiente comando:

git branch

Esto mostrará todas las ramas locales disponibles.

2.2 2. Crear una nueva rama

Para crear una nueva rama, usa el siguiente comando:

git branch nombre-de-la-rama

Donde `nombre-de-la-rama` es el nombre que deseas darle a la nueva rama.

2.3 3. Cambiar a la nueva rama

Después de crear la rama, debes cambiar a ella con:

git checkout nombre-de-la-rama

Desde Git 2.23, puedes usar un solo comando para crear y cambiar de rama al mismo tiempo:

git switch -c nombre-de-la-rama

2.4 4. Verificar la rama actual

Para asegurarte de que estás en la rama correcta, usa:

git branch

La rama actual estará marcada con un asterisco (`*`).

3 Cómo actualizar una rama en el repositorio

3.1 1. Obtener los últimos cambios del repositorio remoto

Antes de actualizar una rama, es recomendable obtener los cambios más recientes del repositorio remoto con:

git fetch

Esto actualizará la información de las ramas remotas sin modificar el código local.

3.2 2. Fusionar cambios de la rama principal

Si trabajas en una rama secundaria y quieres actualizarla con la última versión de la rama principal (por ejemplo, `main` o `develop`), usa:

git merge main

o si prefieres actualizar con un enfoque más lineal:

git rebase main

3.3 3. Subir los cambios al repositorio remoto

Después de actualizar y resolver posibles conflictos, los cambios se pueden subir con:

git push origin nombre-de-la-rama

Si la rama es nueva y no existe en el repositorio remoto, usa:

git push –set-upstream origin nombre-de-la-rama

4 ¿Cómo funcionan las ramas en Git?

Git maneja las ramas como referencias a commits específicos dentro del historial del proyecto. Cada vez que se crea una nueva rama, simplemente apunta a un commit existente y permite realizar cambios sin afectar otras ramas.

Algunos conceptos clave en el uso de ramas incluyen:

- **HEAD**: Indica la rama y el commit en el que estás trabajando actualmente.
- **Merge**: Combina los cambios de una rama en otra.
- **Rebase**: Reaplica los cambios de una rama sobre otra para mantener un historial más limpio.
- **Branch remoto**: Son las ramas almacenadas en un servidor remoto como GitHub, GitLab o Bitbucket.

5 Conclusión

Las ramas en Git son fundamentales para una gestión eficiente del código fuente. Permiten trabajar en paralelo, experimentar con nuevas características y colaborar sin interferir en el código principal. Mantenerlas actualizadas y gestionarlas correctamente mejora la eficiencia y organización en los proyectos.