Práctica GIT-GITHUB

**2024**

Ángel Roldán Rabanal

Entorno de Desarrollo 1º DAM

28/10/2024



Índice

[1. Explicación de Git 2](#_Toc181959153)

[2. Características de Git 2](#_Toc181959154)

[3. Diagrama de las áreas de Git 2](#_Toc181959155)

[4. Explicación de GitHub 2](#_Toc181959156)

[5. Instalación de Git y creación de cuenta en GitHub 3](#_Toc181959157)

[Instalación de Git 3](#_Toc181959158)

[Creación de una cuenta en GitHub 3](#_Toc181959159)

[6. Repositorio Local 4](#_Toc181959160)

[1. Configuración inicial 4](#_Toc181959161)

[2. Crear un nuevo repositorio local 4](#_Toc181959162)

[3. Añadimos los archivos y hacemos pruebas 4](#_Toc181959163)

[6. Repositorio remoto 14](#_Toc181959164)

# 1. Explicación de Git

**Git** es un sistema de control de versiones distribuido, que permite gestionar los cambios en el código de un proyecto. Esto es útil para proyectos de programación o de colaboración, ya que ayuda a mantener un historial de cambios, permite trabajar en equipo sin sobrescribir el trabajo de otros y facilita la restauración de versiones anteriores.

# 2. Características de Git

* **Control de versiones distribuido**: Cada copia del repositorio contiene todo el historial de cambios, lo que permite trabajar sin conexión y distribuir el trabajo entre varios desarrolladores.
* **Velocidad y eficiencia**: Git está optimizado para realizar operaciones de manera rápida.
* **Integridad**: Git usa algoritmos de hashing (SHA-1) para garantizar que los cambios y versiones sean seguros.
* **Soporte para ramas y merges**: Facilita la creación de ramas, permitiendo a los desarrolladores trabajar en diferentes características sin afectar la rama principal.
* **Manejo de cambios no lineales**: Git permite gestionar múltiples líneas de desarrollo de manera efectiva.

# 3. Diagrama de las áreas de Git

El flujo de trabajo en Git se organiza en tres áreas principales:

1. **Área de trabajo (Working Directory)**: Es la copia de los archivos en la que estás trabajando en tu computadora.
2. **Área de preparación (Staging Area)**: Es donde se guardan los cambios que quieres incluir en el próximo commit.
3. **Repositorio (Repository)**: Es la base de datos donde se guardan de manera definitiva los cambios confirmados.

# 4. Explicación de GitHub

**GitHub** es una plataforma de desarrollo colaborativo que permite alojar repositorios de Git en la nube. Fue creada para facilitar la colaboración entre desarrolladores y para gestionar proyectos de código fuente compartidos. Al ser una herramienta en línea, GitHub facilita la integración de equipos que trabajan en un mismo proyecto, permitiendo que cada desarrollador trabaje en una copia del repositorio y que se combinen los cambios en una ubicación central.

**Funciones principales de GitHub**:

* **Almacenamiento de repositorios en la nube**: Puedes guardar proyectos de Git en la plataforma, permitiendo su acceso desde cualquier lugar con conexión a internet.
* **Colaboración**: GitHub permite a varios usuarios trabajar en un mismo repositorio, realizar sugerencias de cambio mediante pull requests y discutir modificaciones en el código.
* **Control de versiones y ramas**: Integra las funciones de Git para gestionar diferentes versiones del código y mantener un registro de los cambios realizados por cada usuario.
* **GitHub Pages**: Permite alojar páginas web estáticas directamente desde un repositorio.
* **Integración con herramientas de CI/CD**: GitHub permite configurar flujos de trabajo de integración y despliegue continuos, facilitando la entrega de software de manera rápida y segura.

# 5. Instalación de Git y creación de cuenta en GitHub

## Instalación de Git

1. **Windows**: Descarga el instalador desde <https://git-scm.com/> y sigue los pasos de instalación.
2. **Mac**: Usa el comando brew install git si tienes Homebrew instalado, o descarga desde el sitio oficial.
3. **Linux**: Usa el comando sudo apt-get install git en distribuciones basadas en Debian o sudo yum install git en distribuciones basadas en Red Hat.

Para verificar la instalación, ejecuta:

git –version

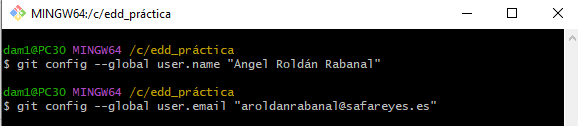
## Creación de una cuenta en GitHub

1. Ve a <https://github.com/> y selecciona "Sign up" (Registrarse).
2. Completa el formulario de registro con tus datos y verifica tu cuenta.

# 6. Repositorio Local

## 1. Configuración inicial

Antes de nada deberemos hacer una configuración inicial del nombre de usuario y el correo electrónico, estos se utilizarán para guardar las configuraciones.



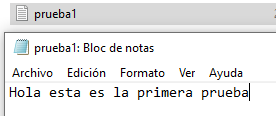
## 2. Crear un nuevo repositorio local

Primero debemos tener nuestra carpeta creada y situarnos en la misma, una vez allí ya podremos iniciar nuestro repositorio de Git.

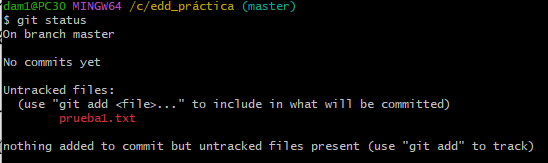


## 3. Añadimos los archivos y hacemos pruebas

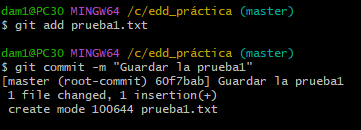
Creamos nuestro archivo y escribimos algo dentro.



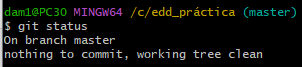
Si hacemos un git status podemos ver lo siguiente.



Lo que nos indica es que se ha detectado un nuevo archivo pero no está añadido para que pueda ser guardado.

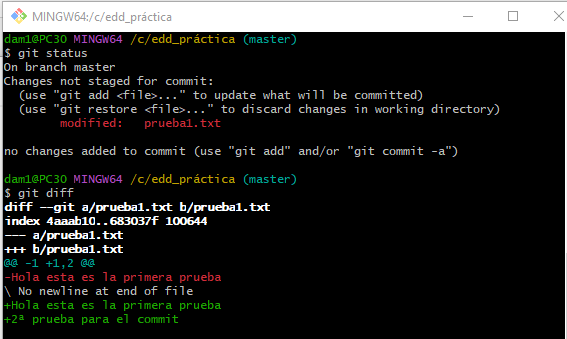


Ahora habiendo hecho el git add para añadir el archivo y el git commit –m “El mensaje que nosotros queramos”, ya tendremos hecho nuestro primer commit en Git.

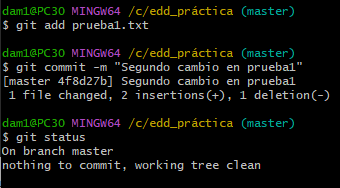


Si hacemos otro git status podemos ver que ya no nos salen cambios.

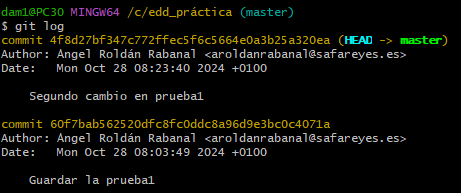
Ahora modificamos nuestro documento y lo guardamos, si hacemos el git status podemos ver que nos dice que se ha modificado nuestro archivo, y con el git diff podemos ver los cambios realizados.



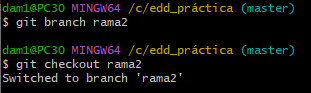
Y ahora si volvemos a añadir el archivo y hacerle un commit se nos habrá guardado de nuevo nuestros cambios.



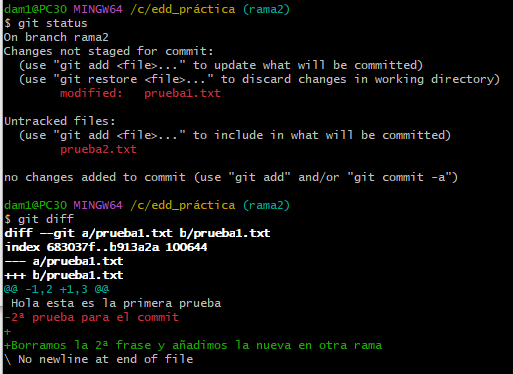
Si hacemos un git log podremos ver los commit realizados con sus etiquetas, su ID y la rama a la que pertenece.



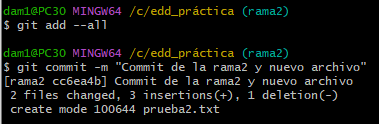
Ahora vamos a crear otra rama con el git branch y con el git checkout podemos cambiar de rama.



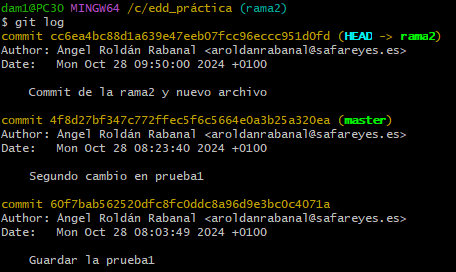
En nuestra nueva rama volvemos a hacer cambios en nuestro archivo y si queremos podemos crear otro nuevo y vemos lo que pasa, con el git status vamos a ver que se ha modificado el archivo prueba1 y que el prueba2 todavía no ha sido añadido, y el git diff vemos los cambios en el archivo prueba1.



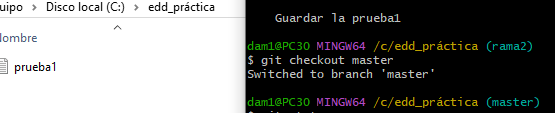
Y ahora hacemos que añada todos los archivos con el git add –all y el git commit –m “Nuestro mensaje” para guardar los archivos



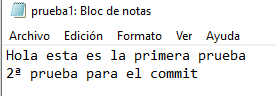
Si hacemos el git log, podemos ver que se ha añadido el nuevo commit en la rama2



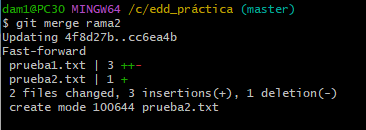
Ahora si cambiamos de rama podemos ver que solo tenemos un archivo



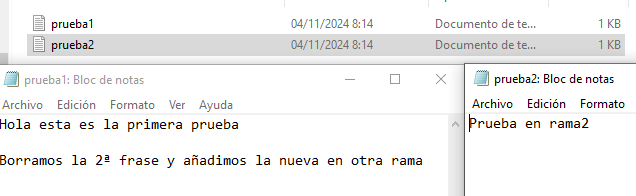
Y además con el texto antiguo



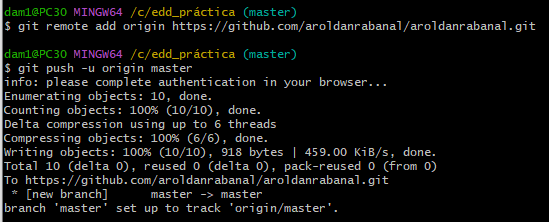
Ahora si hacemos un git merge podemos unir las 2 ramas que tenemos



Y nuestro directorio de la rama principal, quedaría tal que así

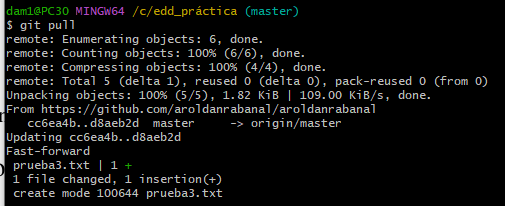


Ahora vamos a conectar nuestro repositorio local con nuestra cuenta de GitHub para tener nuestros repositorios en la nube

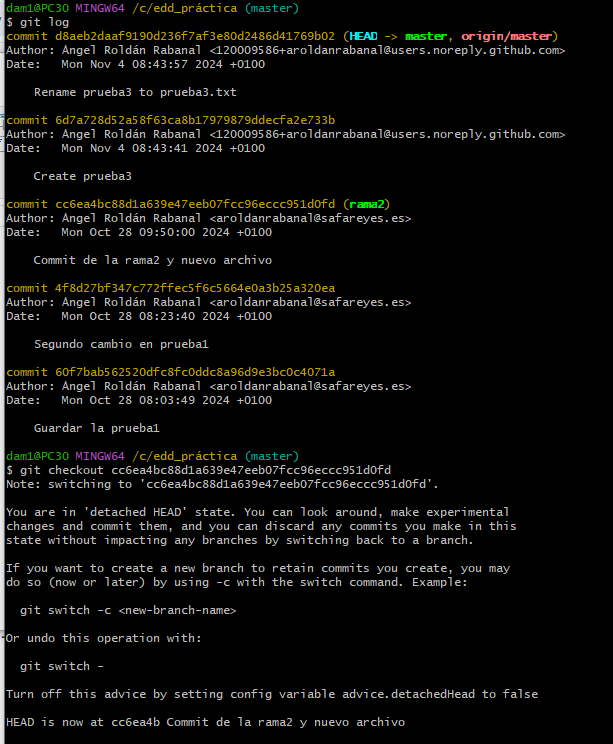


Vamos a crear un archivo desde el mismo GitHub y luego hacemos un git pull para recibir la rama con los cambios

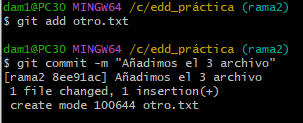




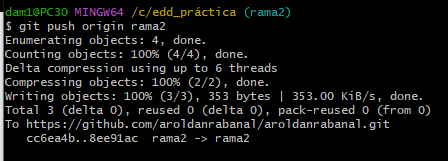
Si queremos volver atrás, vamos a hacer un checkout con el SHA en vez de con el nombre de la rama.



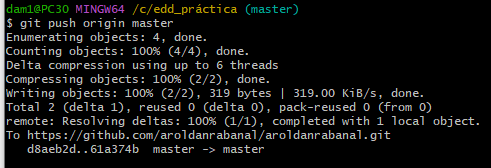
Ahora que estamos en la rama2 vamos a crear otro archivo



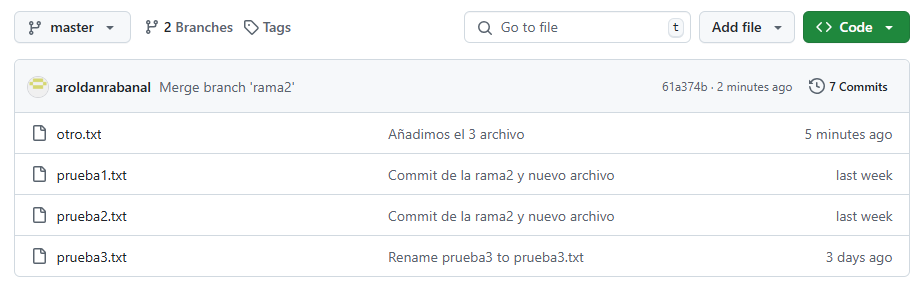
Y lo subimos al GitHub con la rama2



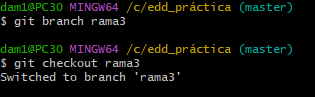
Hacemos un merge para juntar las dos ramas y lo subimos a GitHub



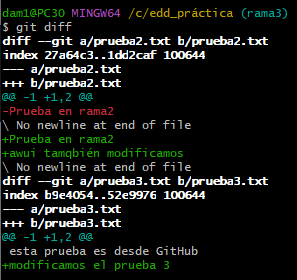
Como podemos ver están nuestros archivos



Vamos a crear otra rama



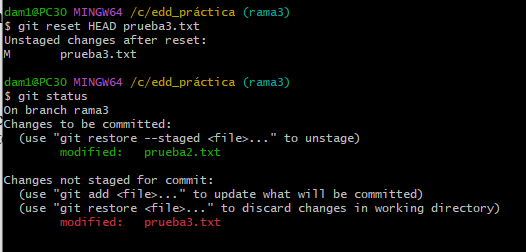
Y vamos a modificar varios archivos



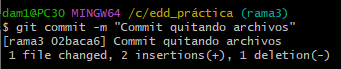
Y vamos a añadir todo



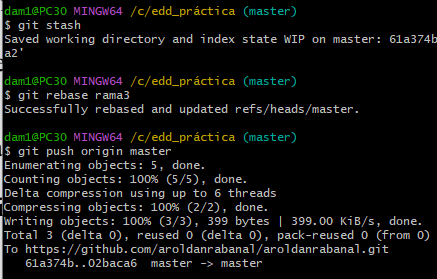
Sin embargo, luego nos hemos dado cuenta de que no queremos guardar los cambios del prueba3, por lo tanto vamos a quitarlo del staging area y luego vemos el status



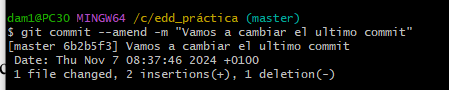
Como podemos ver ya no está en verde si no que está en rojo, ahora podremos hacer nuestro commit bien



Ahora vamos a hacer que la rama master se sobreescriba de la rama3, primero hacemos un stash para hacer un guardado temporal ya que tenemos un archivo que ha sido modificado pero no queremos guardarlo definitivamente y luego hacemos el rebase y lo subimos a GitHub



Y si no confundimos poniendo el mensaje del commit podemos arreglar el ultimo realizado con el siguiente comando

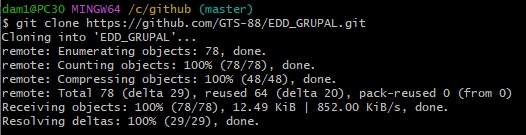


Y por último borramos la rama2 ya que no la vamos a usar más

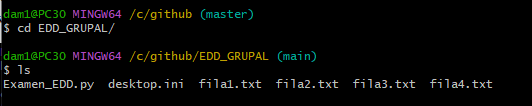


# 6. Repositorio remoto

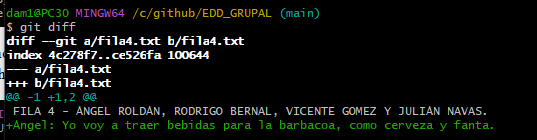
Primero vamos a iniciar nuestro repositorio y configurarlo como hemos hecho otras veces, luego vamos a clonar nuestro repositorio remoto.



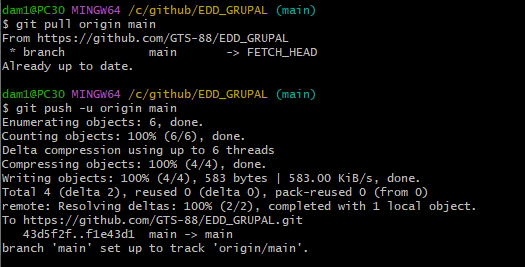
Luego nos movemos a la carpeta contenedora de todo y si hacemos un ls, podremos ver todos los archivos que tenemos



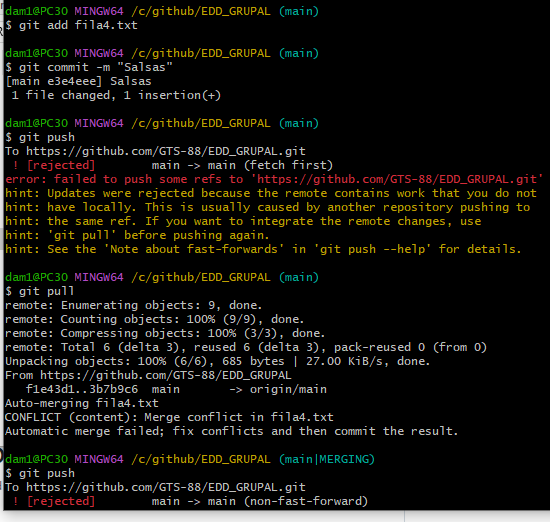
Vamos a modificarlo, para luego añadirlo, hacer el commit y por ultimo subirlo al GitHub



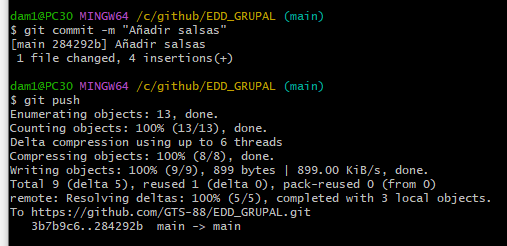
Antes de hacer el push tenemos que hacer un pull, ya que si no nos puede dar algunos fallos, nos debería salir algo así:



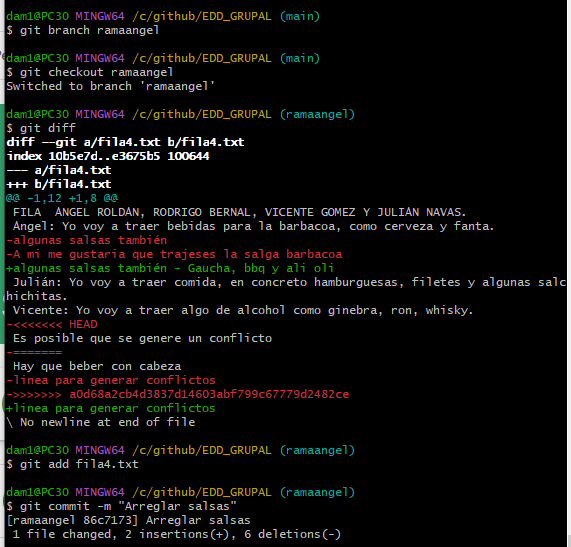
Como podemos ver a la hora de modificar un archivo que no está actualizado, a la hora de hacer el push nos va a dar conflictos, además también hemos hecho que de conflictos pisando la línea que ya estaba escrita, para eso hacemos antes un pull, una vez eso, hacemos el add y el commit, y podremos hacer luego el pull sin problema



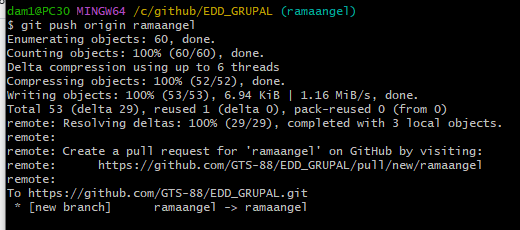
Aquí podemos ver cómo se puede subir nuestro archivo cambiado



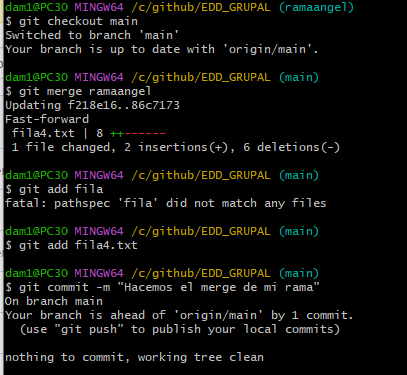
Ahora vamos a crear nuestra rama, hacemos unos cambios en el archivo y hacemos el commit



Luego subimos nuestra rama al repositorio de GitHub

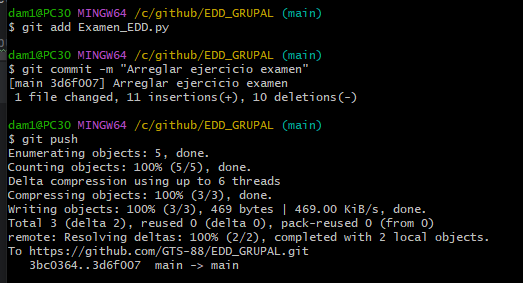


Cambiamos a nuestra rama principal y hacemos el merge y nuestro commit



He tocado también el archivo de Python para hacer algunos cambios y lo subimos, con la siguiente línea podemos abrir directamente nuestro archivo en el IDE de programación.





Por ultimo vamos a eliminar una rama que ya se ha quedado inutilizada, ya que hemos terminado con ella.

Así la borraríamos del repositorio local.



Y de esta forma podremos borrarla del repositorio remoto.



# 7. Conclusión