**Gestión de Proyectos de Programación con Git y GitHub: Aplicación Colaborativa**

**Objetivo de la actividad:**

Aplicar de forma práctica los conocimientos adquiridos sobre Git y GitHub mediante la colaboración en un repositorio compartido y la documentación de los primeros pasos del proyecto grupal, incluyendo la formulación del problema utilizando técnicas de análisis.

**Qué es GitHub:**

GitHub es una plataforma en línea para alojar y gestionar proyectos de software usando el sistema de control de versiones. Permite a los desarrolladores colaborar en tiempo real mediante herramientas como ramas, pull requests y revisiones de código. Su entorno fomenta el trabajo en equipo, la transparencia y la organización del código.

**Configuración y trabajo con Git y GitHub**

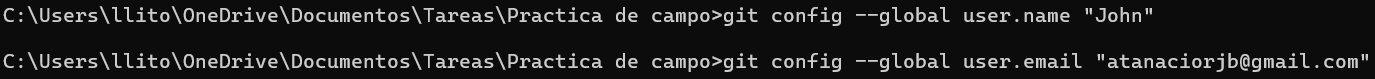
1. Creación de repositorio local y remoto:
2. Configuración inicial:

Para crear un repositorio local primero se tiene que verificar que Git se encuentre instalado, utilizando:  
 git --version

  
En caso de no tenerlo descargado, se puede descargar desde   
git-scm.com

Se configuran el nombre y correo (necesario para los commits):

git config --global user.name "Tu Nombre"  
 git config --global user.email [tu@email.com](mailto:tu@email.com)



1. Creación del repositorio local:

Para crear el repositorio local abre una terminal y navega a tu carpeta del proyecto utilizando.

cd ruta/de/tu/proyecto



Se inicializa un repositorio Git usando:

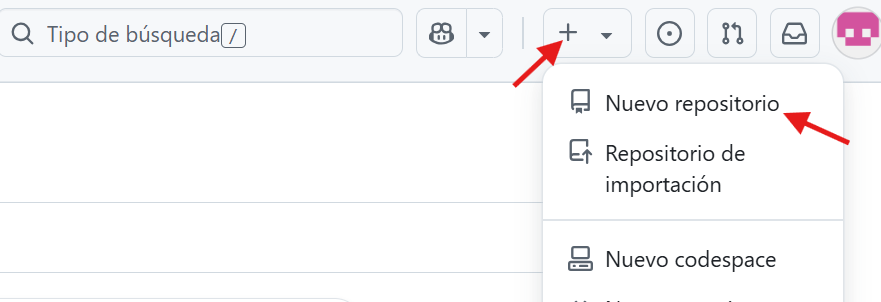
Git init

(Esto creará una carpeta .git que almacenará el historial de versiones)

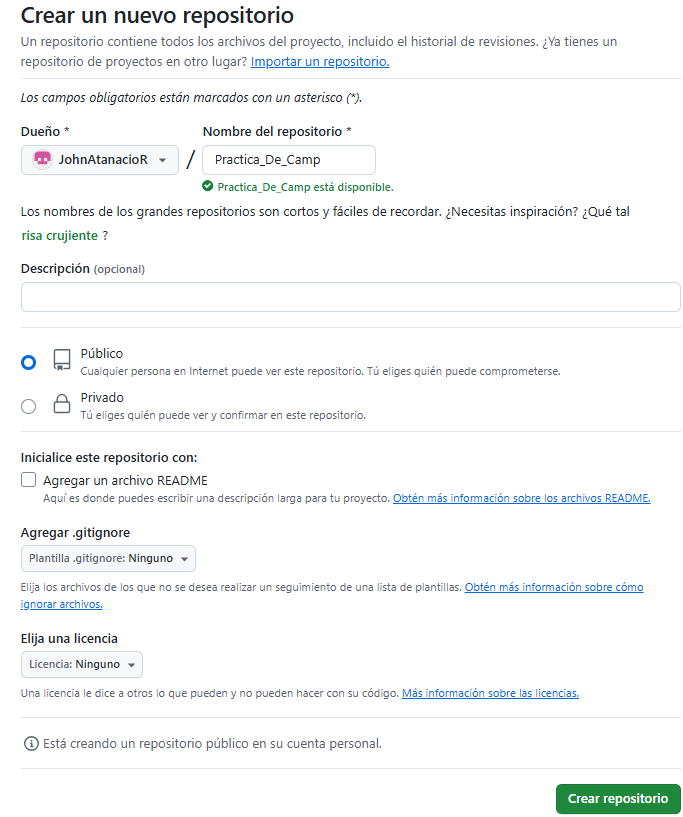


1. Creación del repositorio remoto.

Para crear un repositorio remoto en GitHub se ingresa a <https://github.com/> y se inicia sesión.



Se crea un nuevo repositorio, se inserta el nombre, una breve descripción del repositorio, se elige si el proyecto será público o privado y se crea.



1. Sincronizar repositorio local y remoto.

Empezamos creando un archivo de documentación, como ejemplo de primer commit.

echo "# Practica\_De\_Campo" >> README.md



Ya creado el archivo en la carpeta del repositorio local, ahora conectamos remotamente al repositorio de GitHub

git remote add origin https://github.com/link-del-proyecto

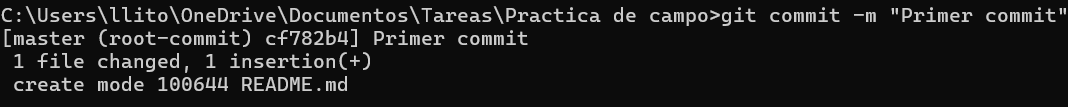


Luego realizaremos Nuestro primer commit utilizando los comandos:

git add README.md o git add .

git commit -m "first commit"

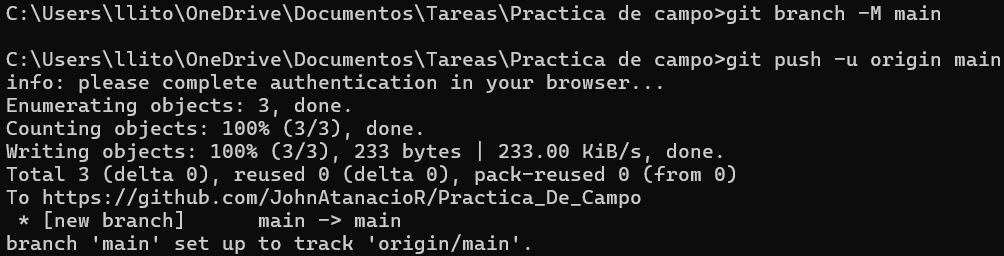




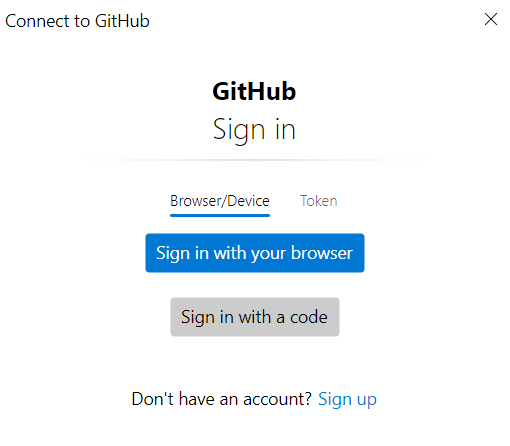
Y finalizaremos hacienda un push para subir lo del repositorio local al repositorio remoto de GitHub.

git branch -M main

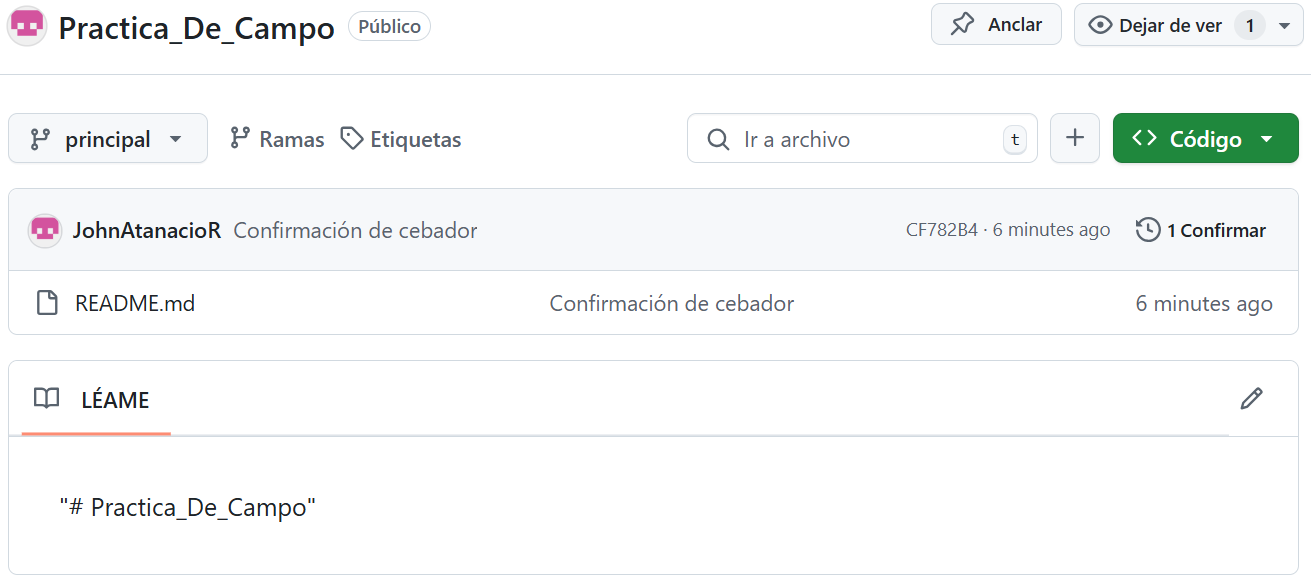
git push -u origin main



Si todo está bien se pedirá iniciar sesión en GitHub para tener la información de la persona en el historial de versiones.



Después de iniciar sesión ya se podrán ver los cambios reflejados en el repositorio remoto.



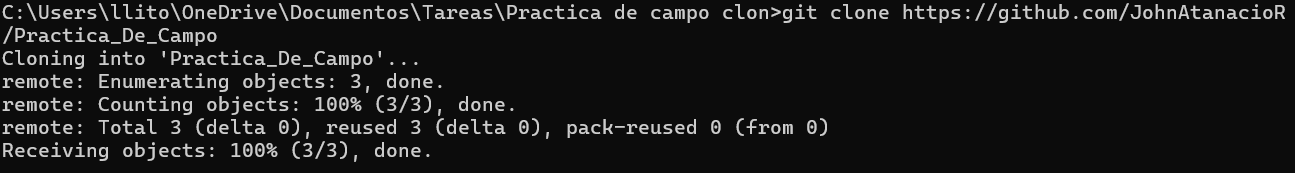
1. Clonar el repositorio en las máquinas locales de cada miembro.

Para clonar un repositorio de GitHub primero se escoge la carpeta donde se desea clonar y luego se clona el repositorio utilizando los siguientes comandos en cada pc:

cd ruta/de/tu/proyecto/donde/clonar

git clone <https://github.com/link_del_repositorio>





1. Crear y cambiar entre ramas utilizando git branch y git checkout.

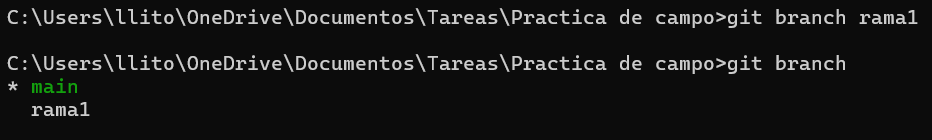
Las ramas de GitHub son una herramienta útil que sirve para aislar el trabajo del proyecto principal, facilitando el trabajo grupal, probar partes de código sin poner en riesgo el main. Cada rama tiene su propio historial de commits, y cuando código está listo, se pueden fusionar usando merge.

Para crear una rama nueva se utiliza el siguiente comando:

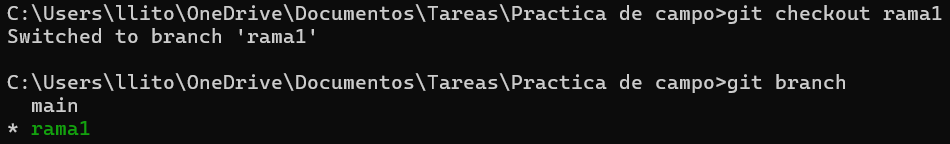
git branch (nombre de la nueva rama)

Y para ver todas las ramas y en cual nos encontramos se utiliza:

git Branch

Para moverse entre ramas se utiliza el comando:

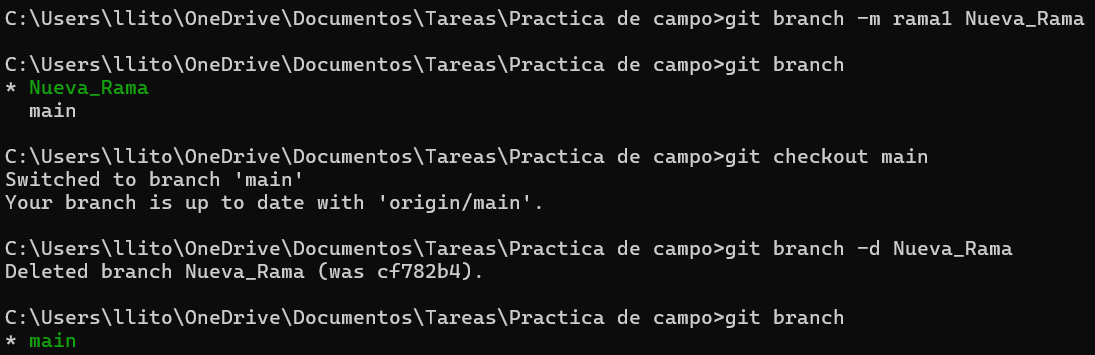
git checkout (nombre de la rama)



Otros comandos útiles:

git branch -d (nombre de la rama) – Para eliminar una rama

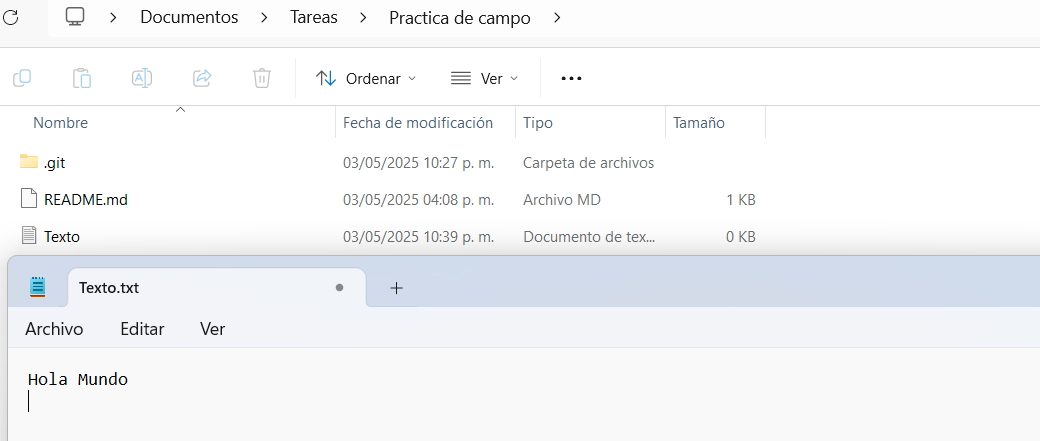
git branch -m (nombre de la rama) (nuevo nombre) – Para renombrar una rama



1. Realizar cambios en el código, hacer commits (git add, git commit) y visualizar historial (git log).

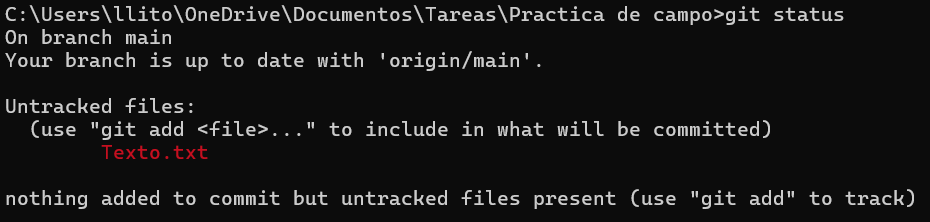
***I. Uso de add y commit***

Primero como ejemplo crearemos en la carpeta de nuestro repositorio local un txt llamado Texto con el texto “Hola mundo”.



Para ver el estado de nuestro repositorio se utiliza el comando:

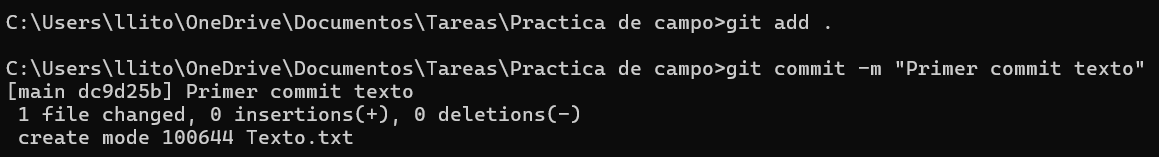
git status



Nos especifica que hay un archivo (Texto.txt) que no está commiteado, para commitearlo debemos utilizar add y seguido un commit con comentario “Primer commit texto”

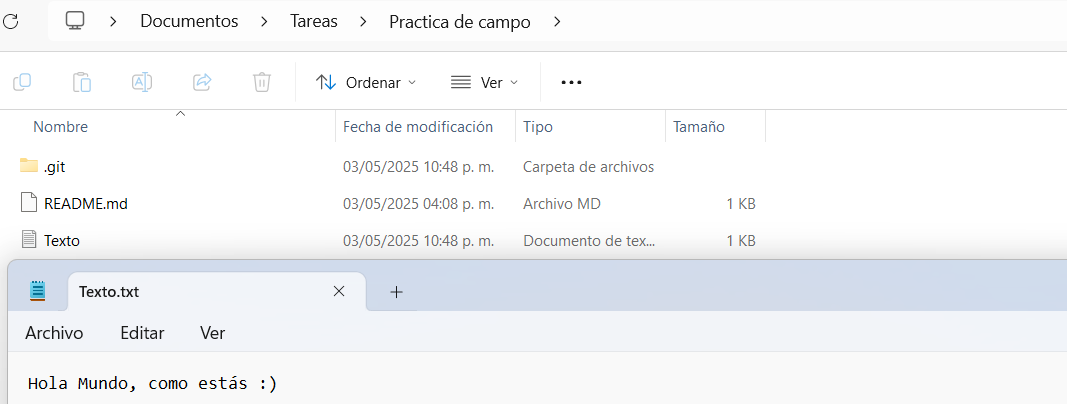
git add . (para todos los archivos) o git add (Nombre del archivo)

git commit -m “Primer commit texto”

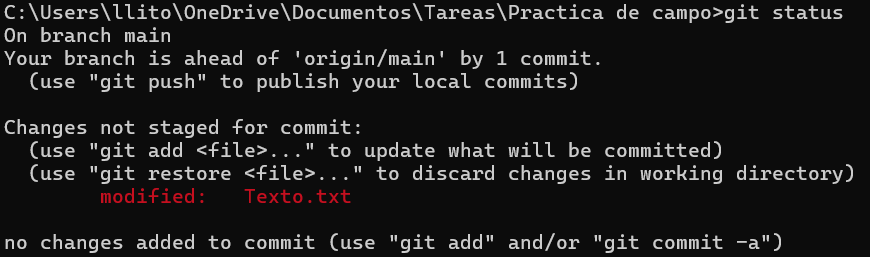


II. Cambios en Código y historial de cambios (log)

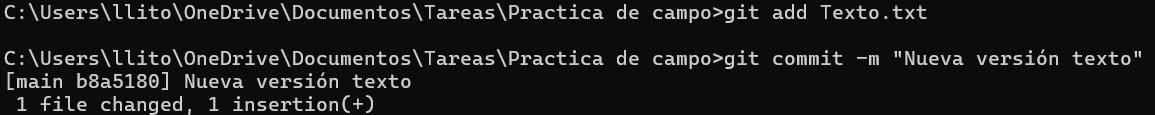
Al momento de realizar cualquier cambio en el código o en algún archivo del repositorio, este automáticamente cambiará su estado el cual podemos visualizar en git status.

*(Cambio del texto)*  


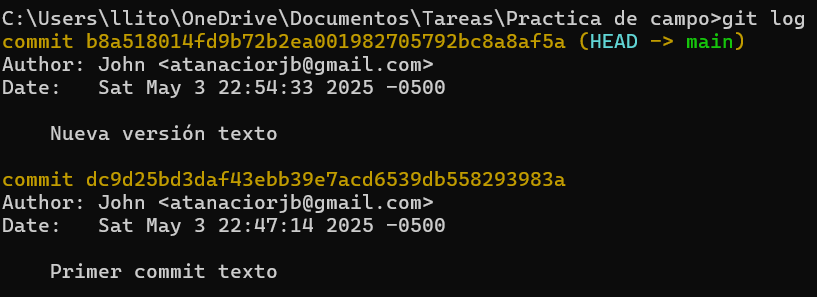
git status



Para actualizar el estado del archivo en nuestro repositorio tenemos que utilizar nuevamente add y commit para esta nueva versión.



Ahora con ambas versiones podemos revisar el historial de versiones utilizando git log.

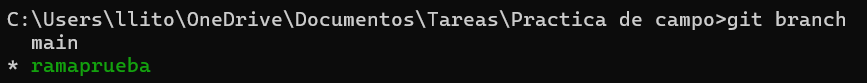


En el historial de versiones se puede revisar el comentario del commit, el autor y la fecha en la que se actualizó.

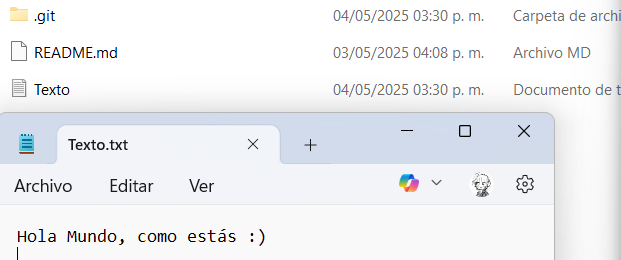
Finalmente podemos pasarlo a nuestro repositorio remoto utilizando el comando push

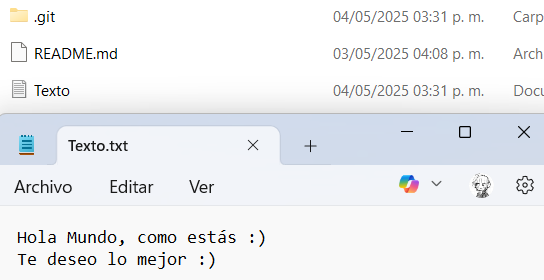
1. Fusionar ramas usando git merge.

Para fusionar ramas usando merge utilizaré la rama de ejemplo “ramaprueba”



En ramaprueba alteraré del código de Texto moviéndome con checkout:

*Código de main*  


*Código alterado de la rama ramaprueba  
*

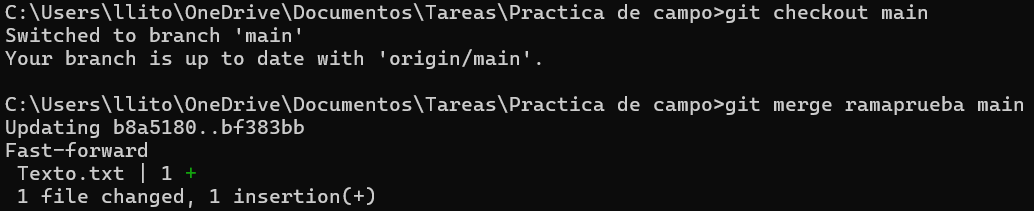
Para usar git merge se utiliza el siguiente orden:

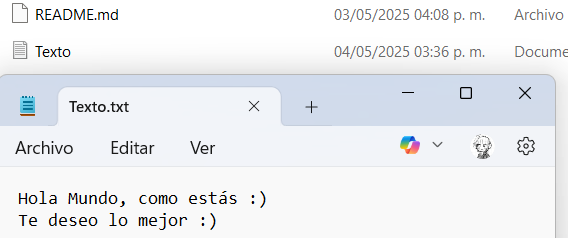
git merge (ramaOrigen) (ramaDestino)

Y además, se tiene que estár posicionado con checkout en la rama destino:

git checkout main

En este caso como fusionaré la rama ramaprueba con main:



*(Nuevo código en main)  
*

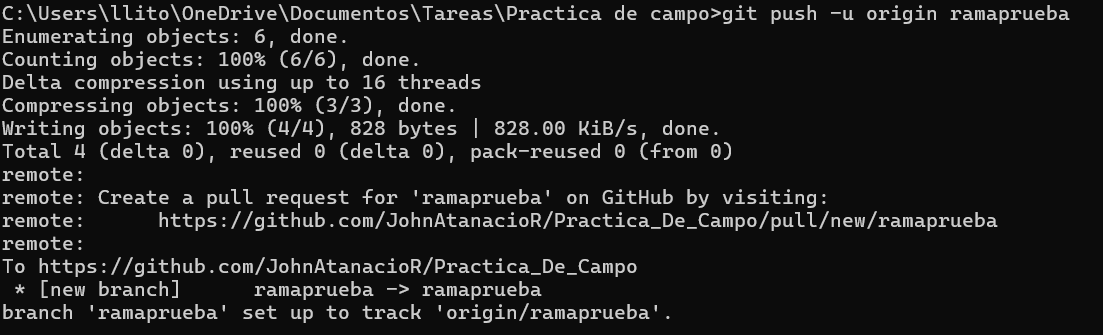
Ahora todos los cambios de la rama de origen se trasmiten a la rama destino, desde nuevos archivos hasta cambios en los existentes.

1. Colaborar a través de Pull Requests en GitHub y documentar la revisión de código (evidencias).

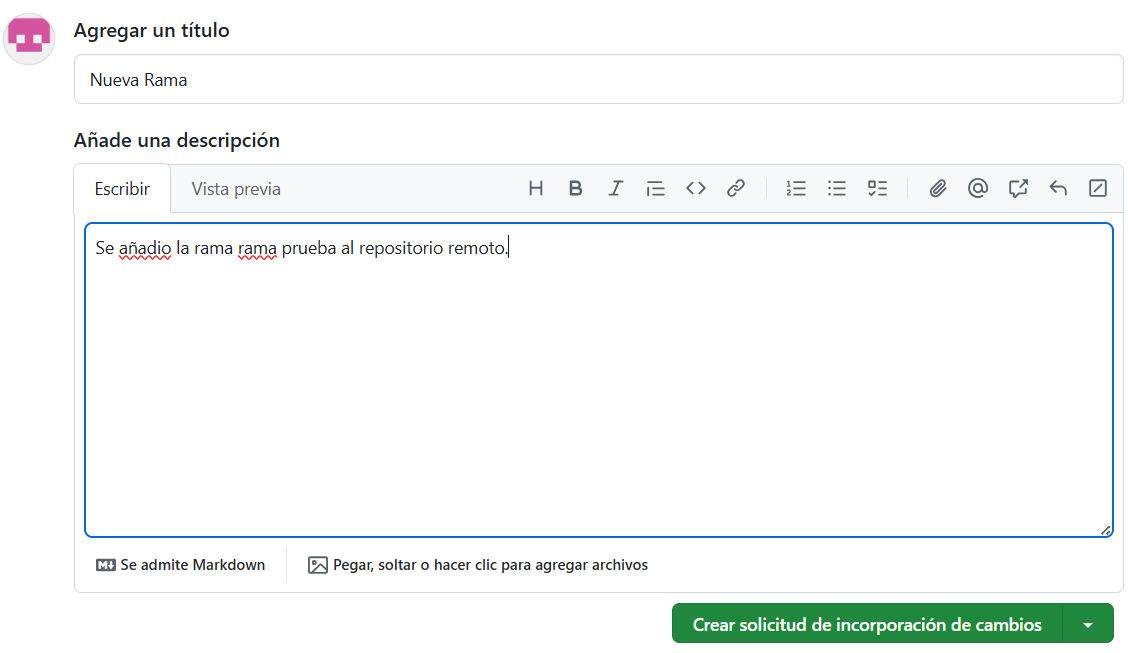
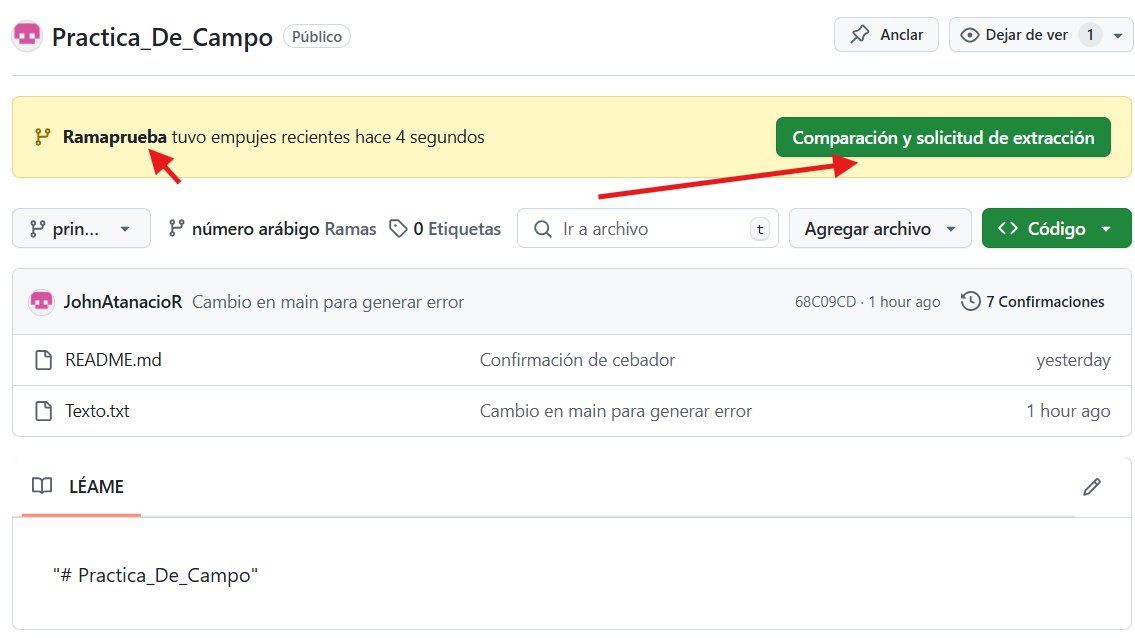
Un pPull Request es intentar hacer un push a un repositorio del cual no eres dueño, por ende se envia una solicitud que tiene que aprobada, se puede hacer siguiendo estos pasos:

Crear una rama para tus cambios (git checkout (nombre rama)).

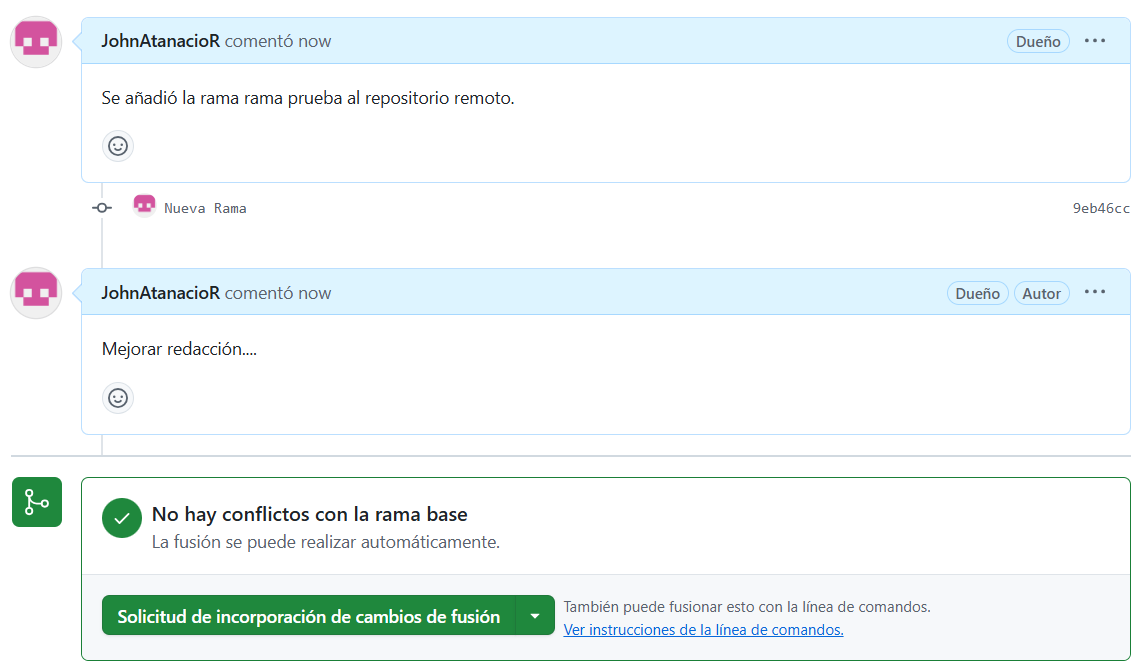
Subirla a GitHub (git push origin (nombre rama)).



Abrir un PR en GitHub para fusionar tu rama con main.



Revisar código: Los colaboradores comentan sugerencias en el PR.



Y si el código es aceptado por los colaboradores se realizó el PR con éxito.

1. Aplicar comandos avanzados como git stash, git revert, y git cherry-pick.

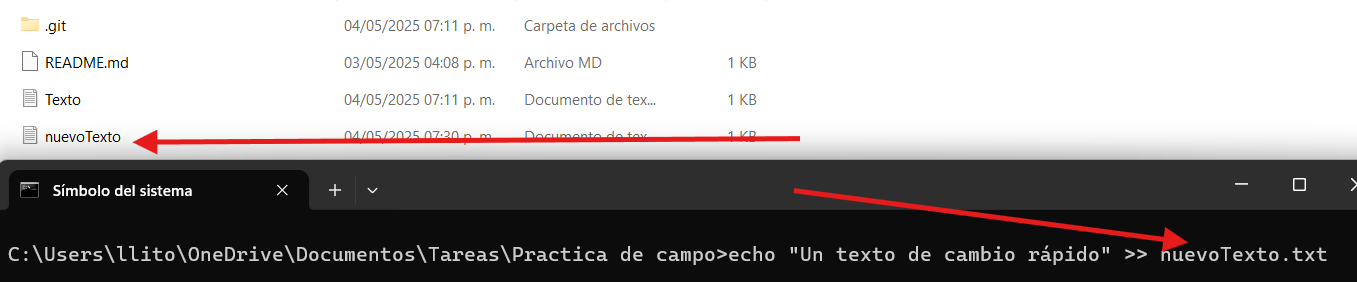
Funcionalidad de los comandos mencionados:

**git stash**

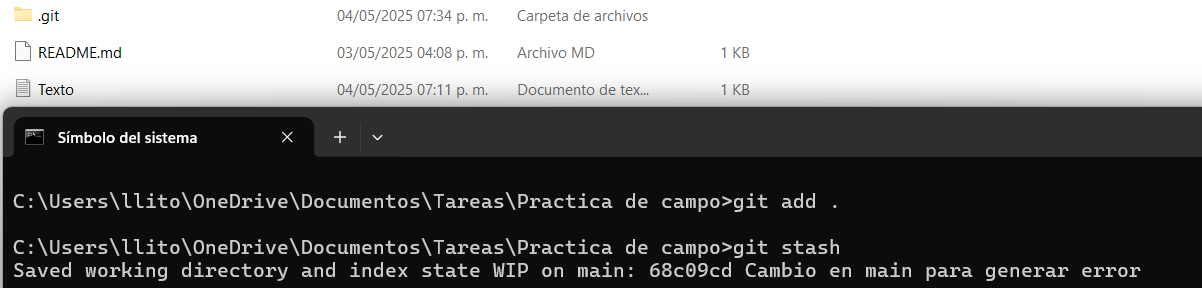
git stash – guarda cambios sin necesidad de hacer commit  
git stash pop– recupera los cambios

Ejemplo:

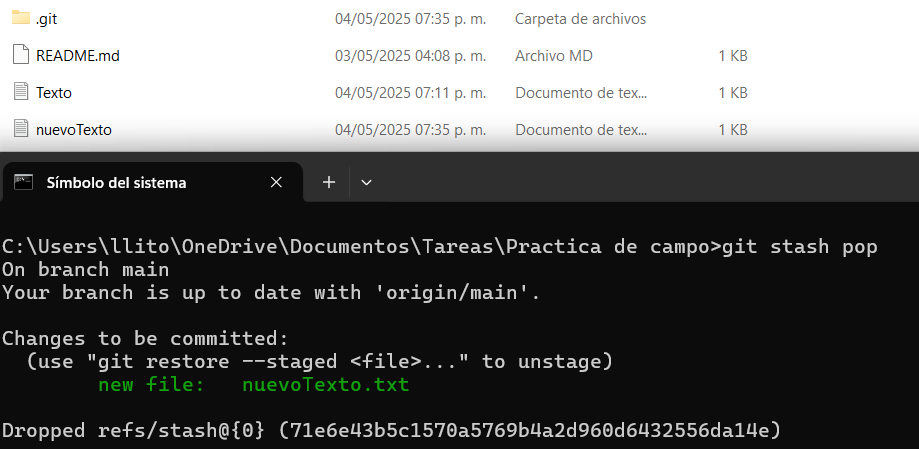
Creamos un nuevo archivo en la nuestra carpeta (nuevoTexto)



Sin usar commit vamos a guardar de forma temporal utilizando git add . y git stash.



Como se puede observar el archivo “nuevoTexto” se “borra” temporalmente de la carpeta, a este se puede volver a invocar utilizando git stash pop



Lo cual restaura todos los cambios que se añadieron al git stash anterior.

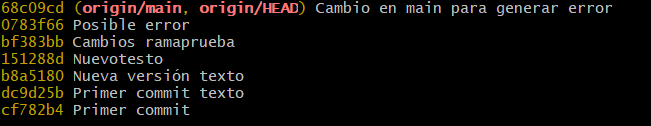
**git revert**

git revert se utiliza para borrar commits dejando un comentario, es decir,

borrar un commit, pero dejar un registro de que se corrigió.

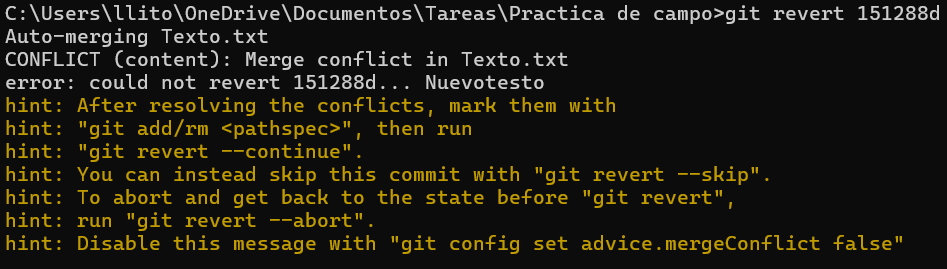
Para utilizar git revert se necesita ver el registro de commits para escoger el commit que borraremos.

Git log --one line



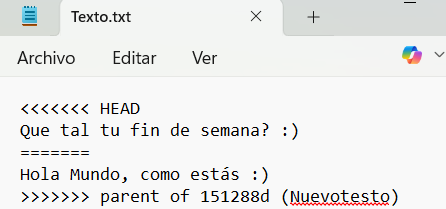
Escogemos el 0783g66 como ejemplo

git revert 0783g66



**Solución de errores**

Git encontró un conflicto esto pasa porque el commit que quieres revertir modificó partes del código que también fueron cambiadas después en otros commits.Esto cambiará el código para ser arreglado manualmente (borrando todo excepto la versión que necesitamos) y tendrá que volver a ser subido con git revert –continue



Despues de solucionar los conflictos se puede usar git revert --continue para que se haga el revert.

