**Escuela Colombiana de Ingeniería Julio Garavito**

**Carrera:** INGENIERIA DE SISTEMAS.

**Materia:** CVDS

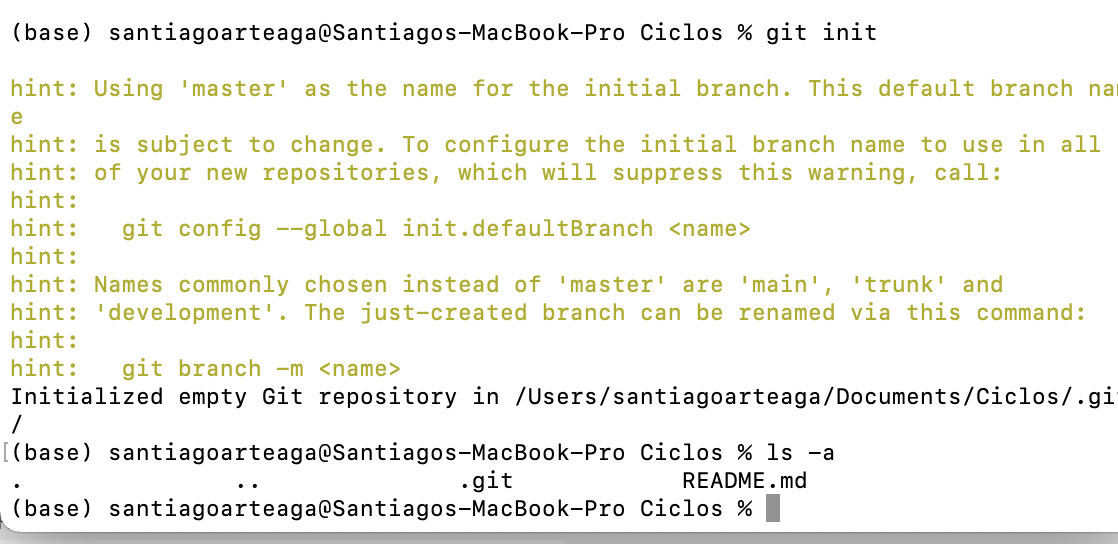
**Título:** LABORATORIO 1- INTRODUCCIÓN GIT

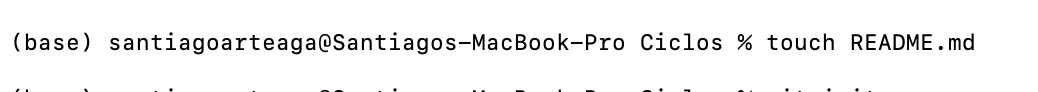
**Estudiante:** Santiago Andrés Arteaga Gutiérrez

**Manejos básicos de GitHub:**

En el presente laboratorio vamos a aprender los manejos básicos de GitHub, esto con el propósito de que entiendas y comiences a trabajar con esta herramienta. Para este laboratorio se trabajará de manera individual la primera parte y con dos integrantes en la segunda parte.

**PARTE I (Trabajo Individual).**

1. Crea un repositorio localmente.
2. Agrega un archivo de ejemplo al repositorio, el README.md puede ser una gran opción.



1. Averigua para qué sirve y como se usan estos comandos git add y git commit -m “mensaje”

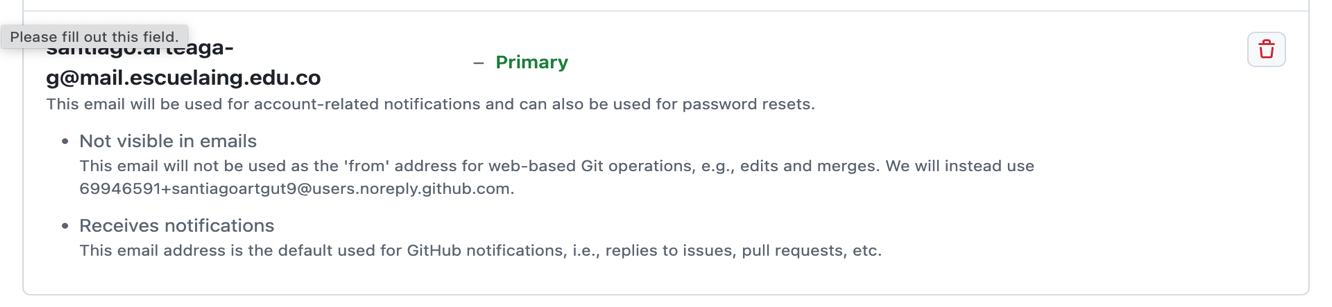
* Gitadd: El comando git add se usa para mover los cambios que has realizado en tus archivos al **área de preparación (staging area)**. El área de preparación es un lugar intermedio donde puedes revisar y decidir qué cambios deseas incluir en el siguiente commit.

**Importancia**: git add no guarda los cambios en el historial del repositorio, solo los prepara. Esto te da control sobre qué archivos y cambios quieres confirmar en tu próximo commit.

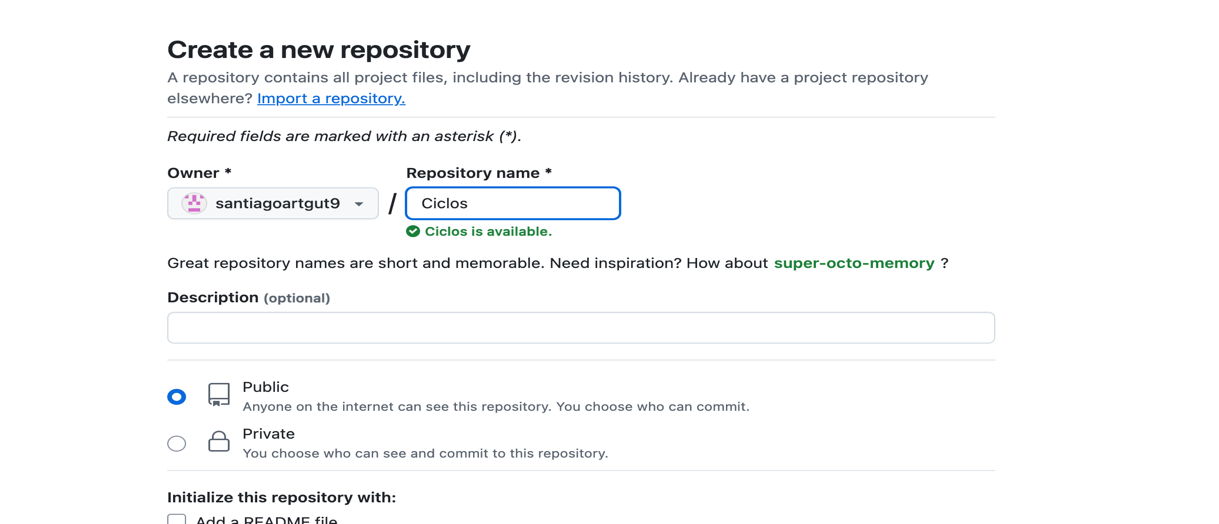
* git commit -m "mensaje": El comando git commit guarda los cambios que has agregado al área de preparación en el **repositorio local**. Cada vez que ejecutas un commit, Git guarda una instantánea de esos cambios y les asigna un identificador único, creando un "punto de control" en la historia del proyecto.

**Importancia**: Hacer commits regulares y con mensajes claros es crucial para llevar un buen historial del proyecto. Los mensajes ayudan a ti y a otros colaboradores a entender el propósito de cada cambio y facilita la colaboración y la revisión del código.

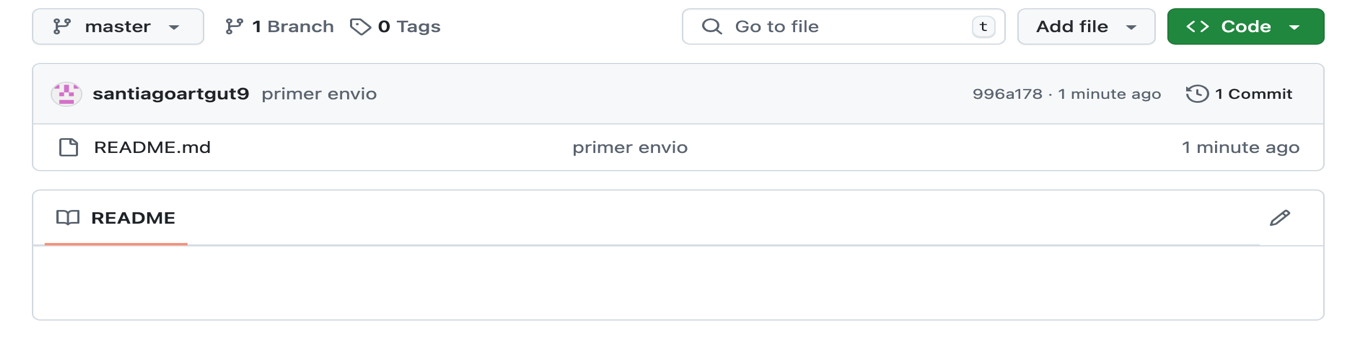
1. Abre una cuenta de github, si ya la tienes, enlazala con el correo institucional.

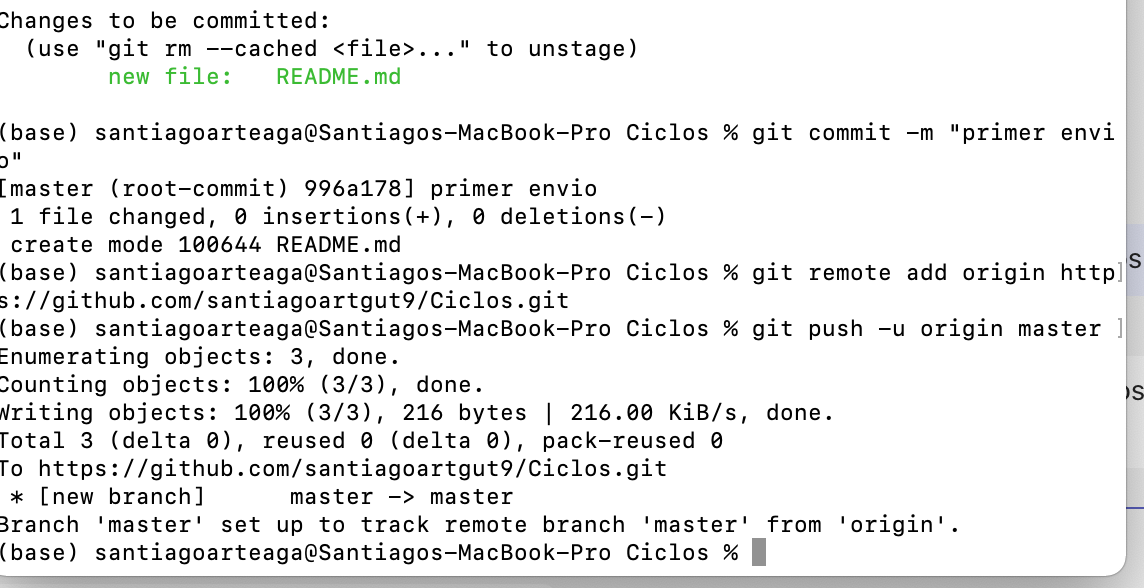


1. Crea un repositorio en blanco (vacío) e GitHub.



1. Configura el repositorio local con el repositorio remoto.



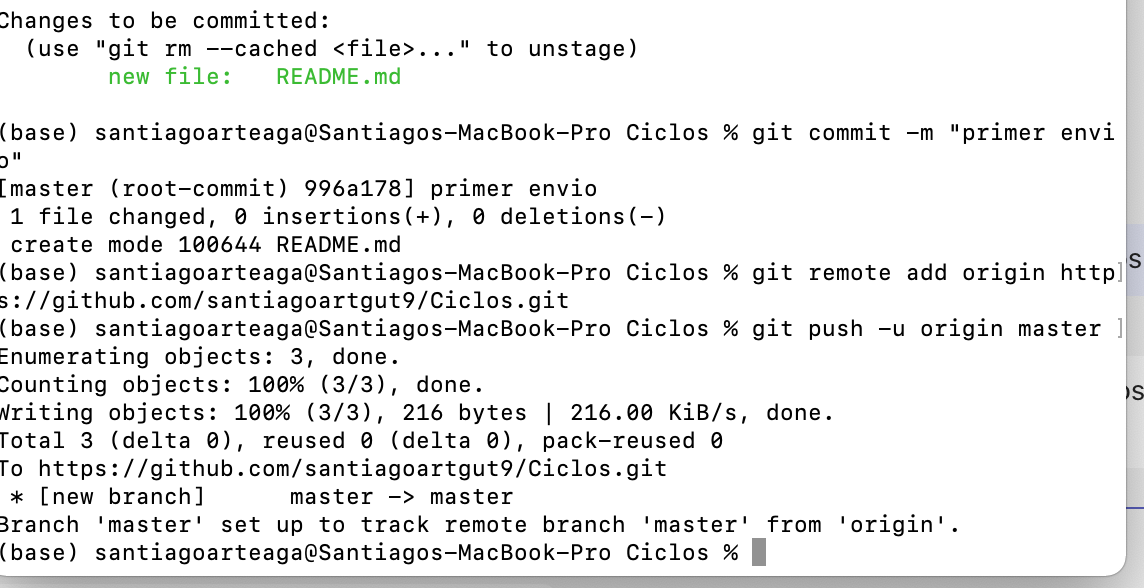


1. Sube los cambios, teniendo en cuenta lo que averiguaste en el punto 3 Utiliza los siguientes comando en el directorio donde tienes tu proyecto, en este orden:

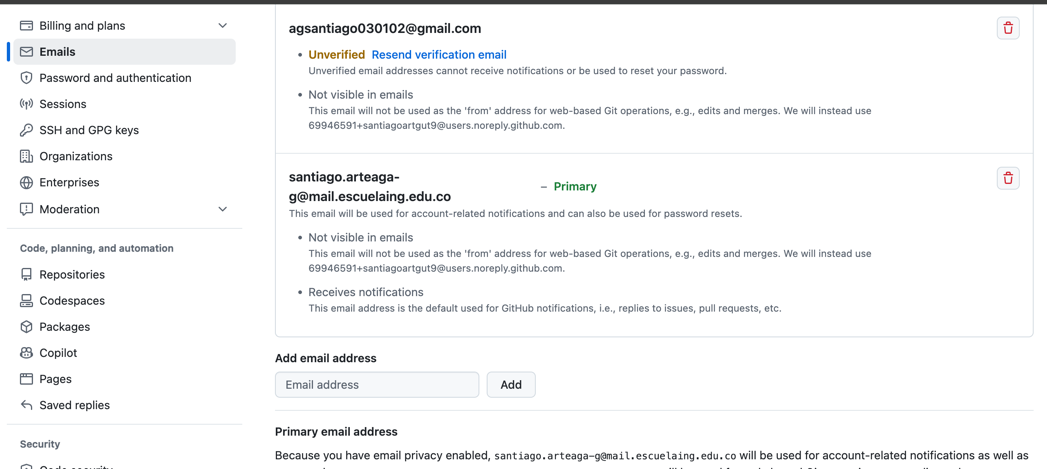
git add .

git commit -m "mensaje, lo que hiciste con el archivo"

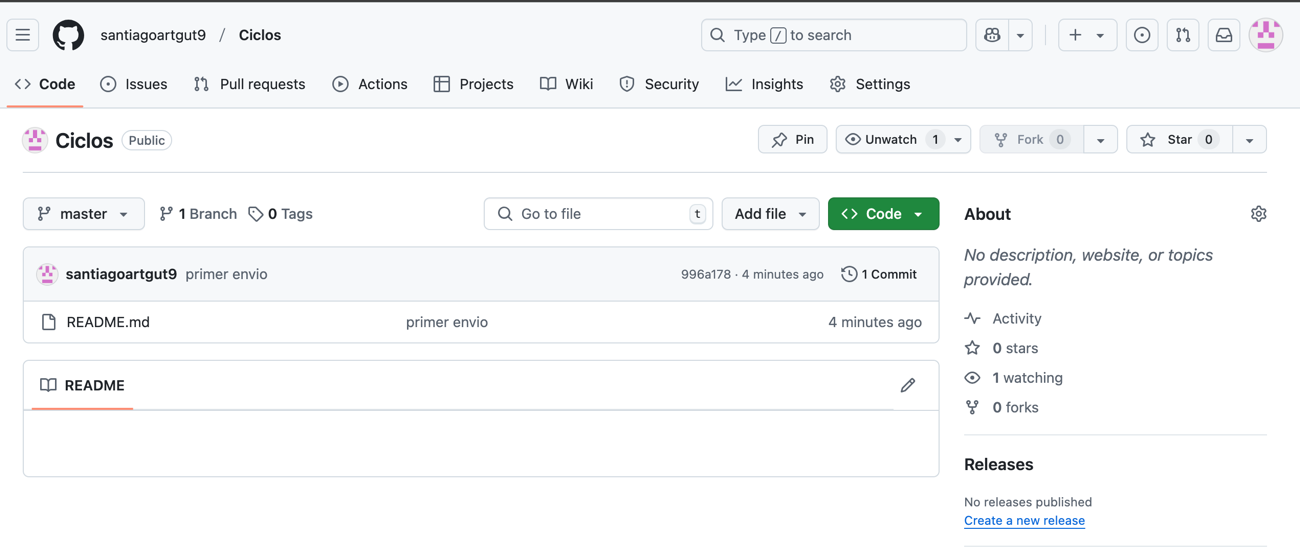
git push "url repositorio"



1. Configura el correo en git local de manera correcta Configurar correo electrónico en GitHub



1. Vuelve a subir los cambios y observa que todo esté bien en el repositorio remoto (en GitHub).

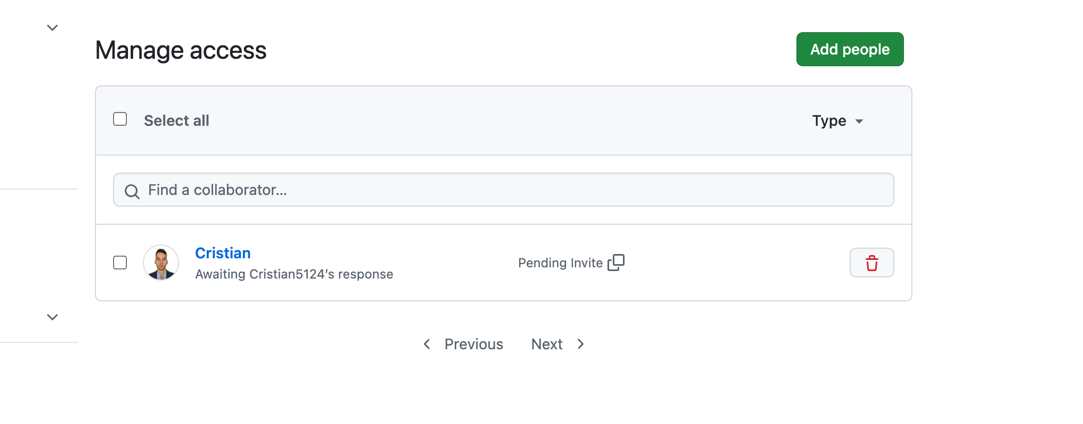
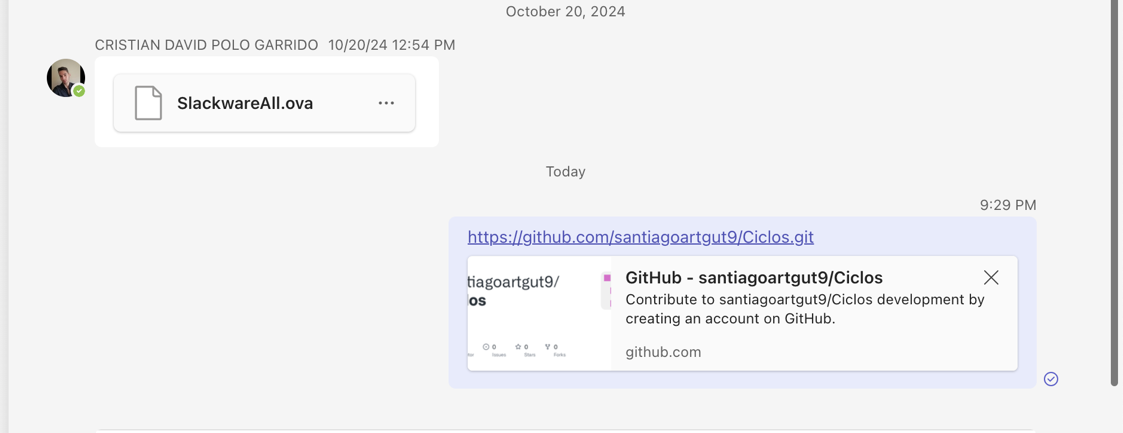


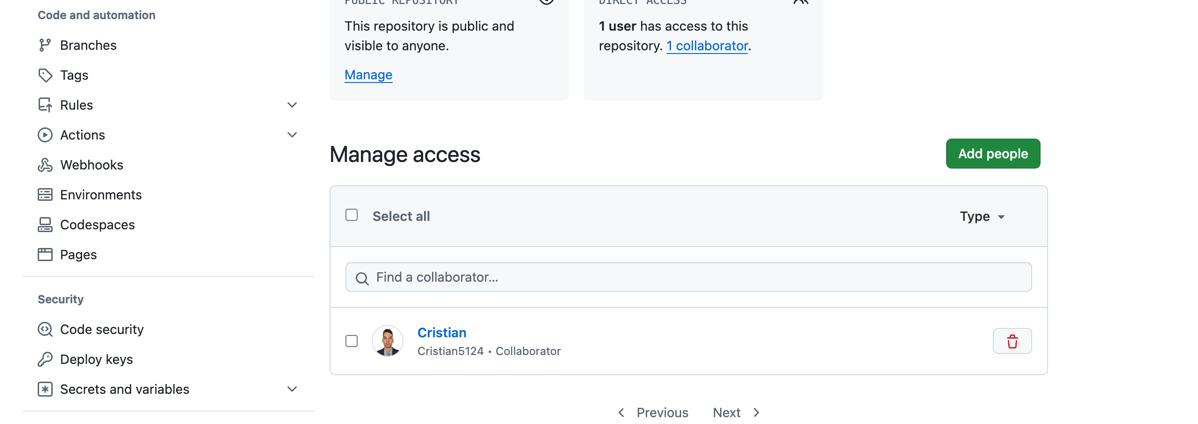
**PARTE II (Trabajo en parejas)**

1. Se escogen los roles para trabajar en equipo, una persona debe escoger ser "Owner" o Propietario del repositorio y la otra "Collaborator" o Colaborador en el repositorio.

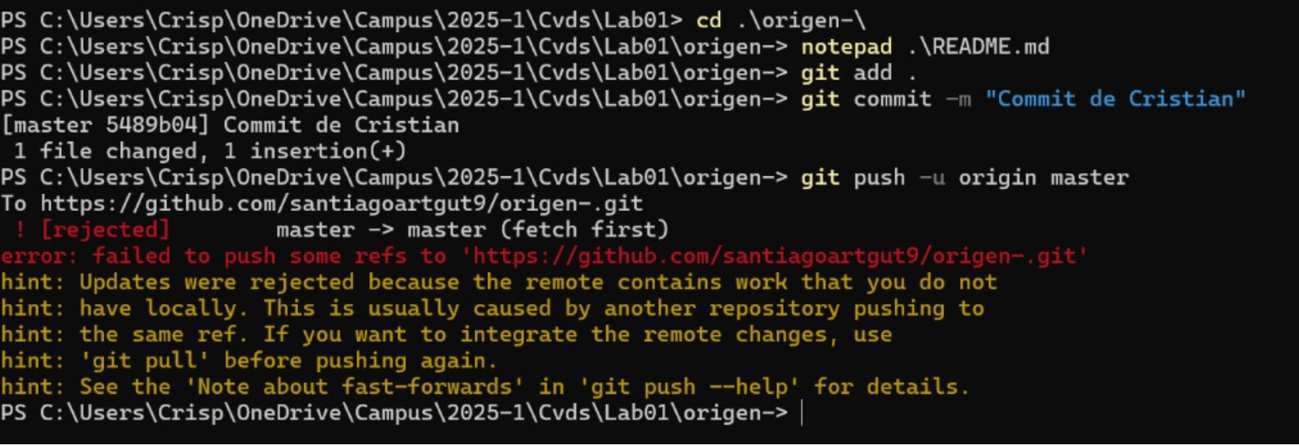
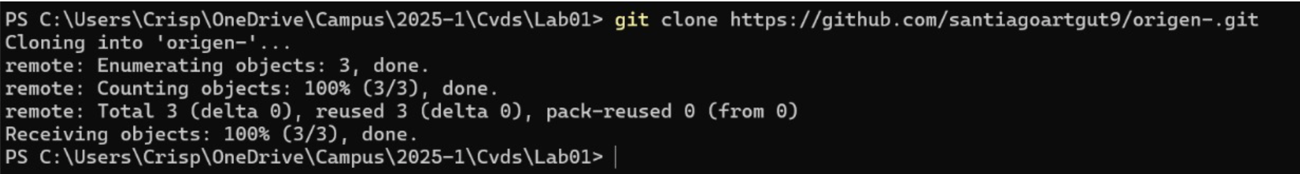
Collaborator: Cristian Polo

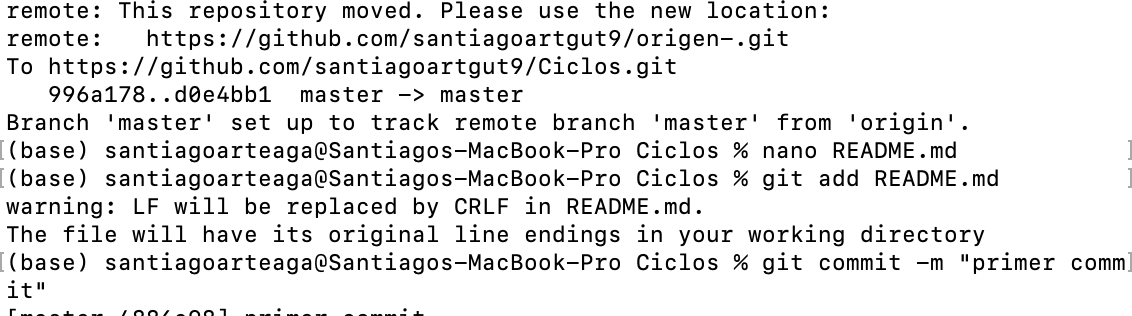
Owner: Santiago Arteaga.

1. El owner agrega al colaborador con permisos de escritura en el repositorio que creó en la parte 1
2. El owner le comparte la url via Teams al colaborador
3. El colaborador acepta la invitación al repositorio



1. Owner y Colaborador editan el archivo README.md al mismo tiempo e intentan subir los cambios al mismo tiempo.





1. ¿Que sucedió?

* El error ocurre porque el repositorio remoto en GitHub tiene cambios que no están en mi copia local. Esto sucede cuando yo y mi colaborador estamos trabajando en la misma rama (master) y él hizo un git push antes que yo.
* Para evitar este problema en el futuro, podemos considerar trabajar en ramas separadas y fusionarlas mediante pull requests.

1. La persona que perdió la competencia de subir los cambios, tiene que resolver los conflictos, cúando haces pull de los cambios, los archivos tienen los símbolos <<< === y >>> (son normales en la resolución de conflictos), estos conflictos debes resolverlos manualmente. Como resolver Conflictos GitHub
2. Volver a repetir un cambio sobre el README.md ambas personas al tiempo para volver a tener conflictos.
3. Resuelvan el conflicto con IntelliJ si es posible, Resolver conflictos en IntelliJ

De esta forma ya sabes resolver conflictos directamente sobre los archivos y usando un IDE como IntelliJ, esto te será muy útil en los futuros trabajos en equipo con Git.

**PARTE III (Trabajo de a parejas)**

1. ¿Hay una mejor forma de trabajar con git para no tener conflictos?
2. ¿Qué es y como funciona el Pull Request?
3. Creen una rama cada uno y suban sus cambios
4. Tanto owner como colaborador hacen un cambio en el README.md y hacen un Pull Request (PR) a la rama main/master

(Recomendación : deben trabajar en equipo y distribuirse de mejor manera quien va a modificar qué, para evitar modificar los mismos archivos al mismo tiempo, si no se editan los mismos archivos la resolución de conflictos es automática)

1. Teniendo en cuenta la recomendación, mezclen los cambios a la rama main a través de PR con el check/review/approval del otro compañero (Cuando se hace merge se deberían borrar las ramas en github)]

Como Borrar Ramas después de un Pull Request

Se dirigen a la configuración de su repositorio:

Y en general Se dirigen al final en el área del pull requests y seleccionan “Automatically delete head branches”

**Aprobación Pull Request**

Nos dirigimos (todavía en configuraciones) a Branches, en esta visualizarán donde daremos protección de nuestras ramas, seleccionamos Add rule

Aquí damos el nombre de nuestra rama (Verificar el nombre tal cual lo tenemos en nuestro repositorio) y seleccionamos la primera opción como se muestra, así estamos requiriendo que cuando se haga ese pull request en nuestra rama se necesita aprobación de otro compañero, Vamos al final y damos clic en Create

Y así protegimos nuestra rama principal, esto se vuelve muy relevante cuando trabajamos en parejas o en equipos, deberían tener un mensaje final que se vea así