Trabajo Práctico Nº 3 Sistema de Control de Versiones

Objetivos:

Que el alumno pueda:

* Identificar los beneficios de un sistema de control de versiones.
* Relacionar Git con plataformas web para administrar proyectos. Tarea previa:
* Lectura del material distribuido por la cátedra sobre el tema.

Crea una cuenta en un sistema de control de versiones web como github, gitlab u otros, desarrolla los incisos siguiendo la propuesta de cada video, sube la práctica desarrollada en tu cuenta e invitanos a colaborar.

Nota: Los textos subrayados en cada inciso son link a videos explicativos.

1. [¿Que es Git?](https://www.youtube.com/watch?v=jSJ8xhKtfP4)
2. [Instalar Git en tu Sistema Operativo.](https://www.youtube.com/watch?reload=9&v=1PiYqxog8mc)
3. Iniciar un proyecto. Uso de init
4. Modificar tu código. [Uso de init y commit](https://www.youtube.com/watch?v=UJGKWMHX038) [Uso del stage](https://www.youtube.com/watch?v=6CzY3WQVYUo)
5. Volver atrás tus cambios. [Cambios antes del commit](https://www.youtube.com/watch?v=9AkcsMpKYhI)

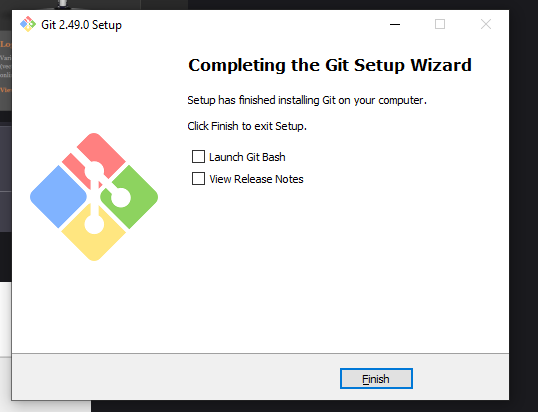
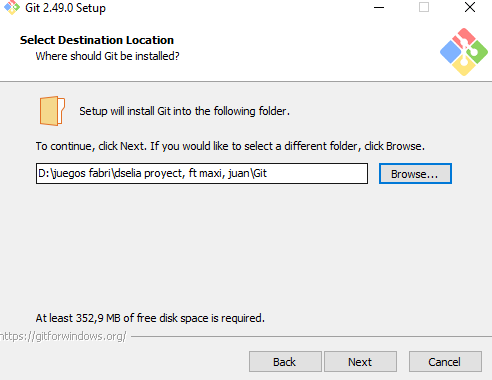
[Cambios luego del commit con reset](https://www.youtube.com/watch?v=h-xN14zPgsQ) [Cambios luego del commit con revert](https://www.youtube.com/watch?v=4SwaV29SpIc)

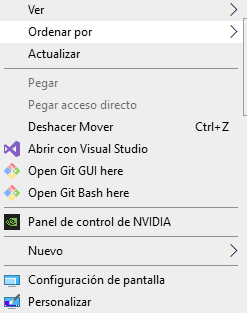
1. Gestionar ramas.

[¿Porque usar ramas?](https://www.youtube.com/watch?v=rmO7t35l1XI) [Uso de branch/checkout](https://www.youtube.com/watch?v=j0U9jBmP3LM) [Uso de commit con ramas](https://www.youtube.com/watch?v=Pw4TOTIkhfk)

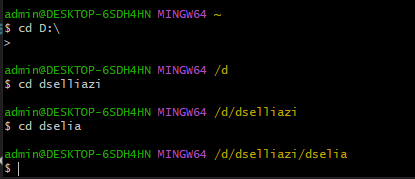
1. Fusionar ramas. [Uso de merge](https://www.youtube.com/watch?v=vu4Rv1SmzwM) [Conflictos con merge](https://www.youtube.com/watch?v=3-rCELg8feQ)
2. [Cambios con stash](https://www.youtube.com/watch?v=fD7ZOK5kZMs&list=PLTd5ehIj0goMCnj6V5NdzSIHBgrIXckGU&index=17)
3. [Gestionar repositorios remotos](https://www.youtube.com/watch?v=tenOpi8KPrc&list=PLTd5ehIj0goMCnj6V5NdzSIHBgrIXckGU&index=18)
4. [Uso de Push](https://www.youtube.com/watch?v=jgeLj45G0tk&list=PLTd5ehIj0goMCnj6V5NdzSIHBgrIXckGU&index=19)
5. [Clonar repositorios](https://www.youtube.com/watch?v=t8O7dMDD5nw&list=PLTd5ehIj0goMCnj6V5NdzSIHBgrIXckGU&index=20)
6. Mantener actualizados los cambios. [Uso de Pull/fecth](https://www.youtube.com/watch?v=r-PTFYVmGNo&list=PLTd5ehIj0goMCnj6V5NdzSIHBgrIXckGU&index=21)
7. [Uso de Rebase](https://www.youtube.com/watch?v=_kheHCD5m7Q&list=PLTd5ehIj0goMCnj6V5NdzSIHBgrIXckGU&index=22)
8. [Uso de Rebase iteractivo](https://www.youtube.com/watch?v=0_QwiNnj_dA&list=PLTd5ehIj0goMCnj6V5NdzSIHBgrIXckGU&index=23)

**Desarrollo**

1. Git es un sistema de control de versiones que realiza un seguimiento de los cambios en los archivos. Git es especialmente útil cuando un grupo de personas y tú estáis haciendo cambios en los mismos archivos al mismo tiempo.
2. 



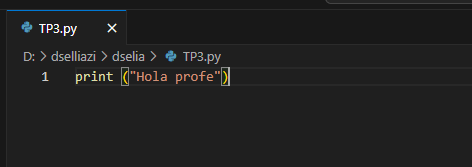
1. Cd para poner ubicación.

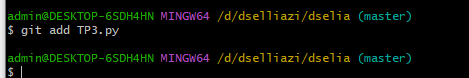


Git Init

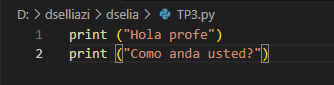


1. Para empezar

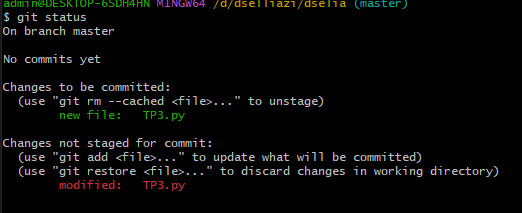


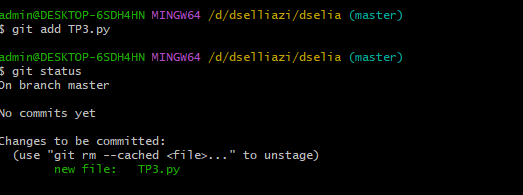


Modificar:

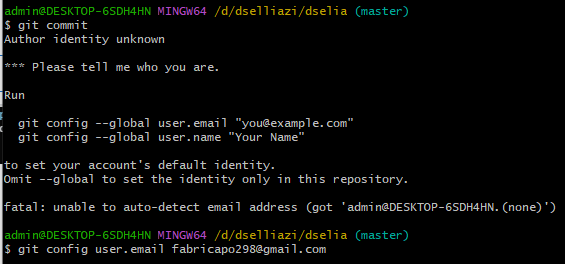


Git status después de guardar.



Git Status después de agregar 

Git Commit (antes de poner email)

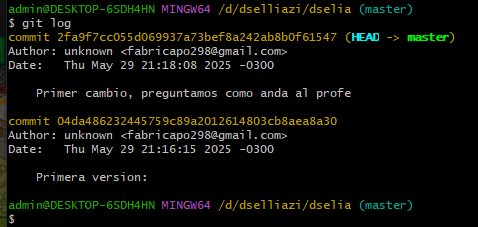


Una vez puesto el email.

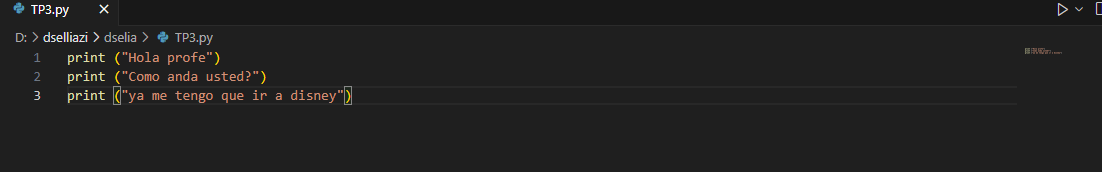


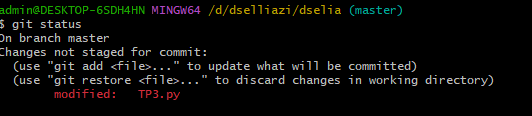


Git Log

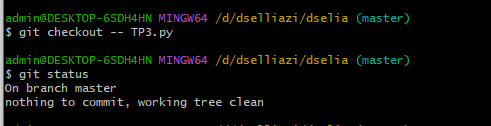


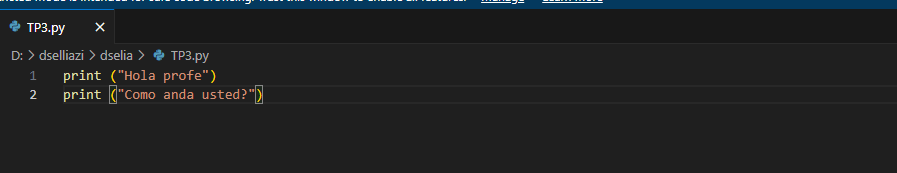
Modificamos TP3 sin hacer commit:



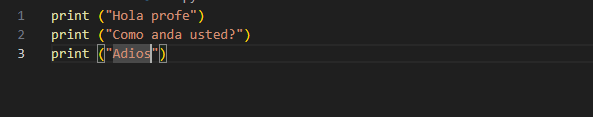
Como no lo agregamos queda: 

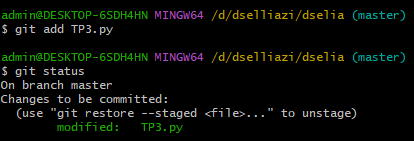
Usamos checkout para deshacer los cambios antes del commit:



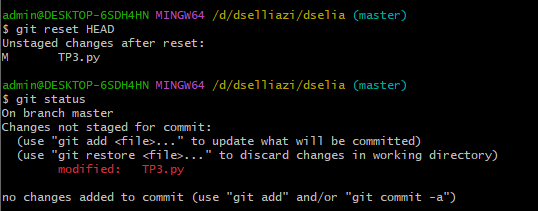


Para reset, Añadimos el cambio .

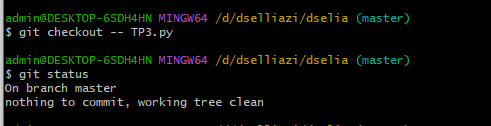




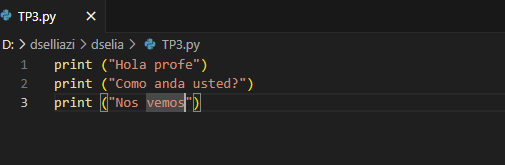
Usamos reset para volver antes de añadirlo.

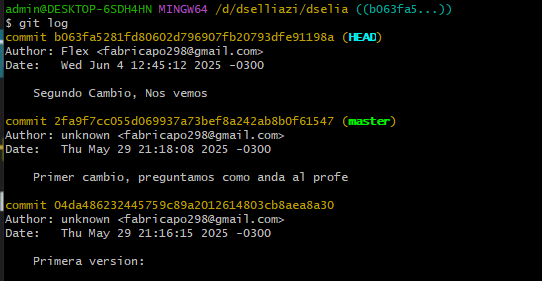


Mismo paso de antes.

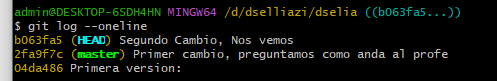


Agregamos un cambio y usamos commit:

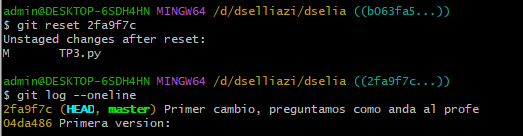




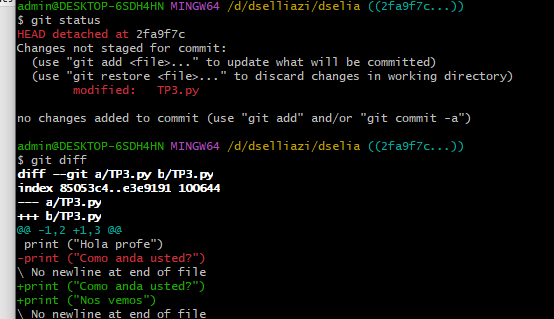
Git log oneline, tiene los commits resumidos.



Git reset y volvemos al primer cambio



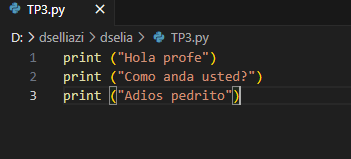
El reset lo saca del stage y git diff nos da los cambios:

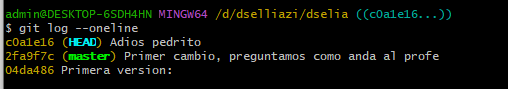


Y con esto ya volvimos al primer cambio

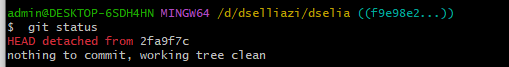


Para el revert volvemos a añadir un commit:

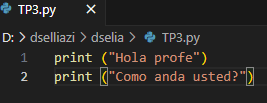




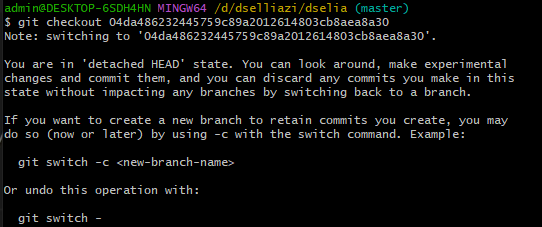
Si usamos git revert HEAD deshacemos los cambios hechos en el ultimo revert

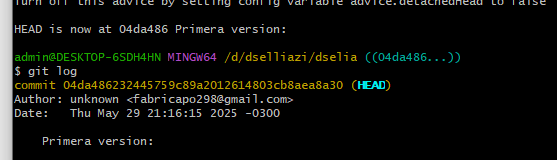


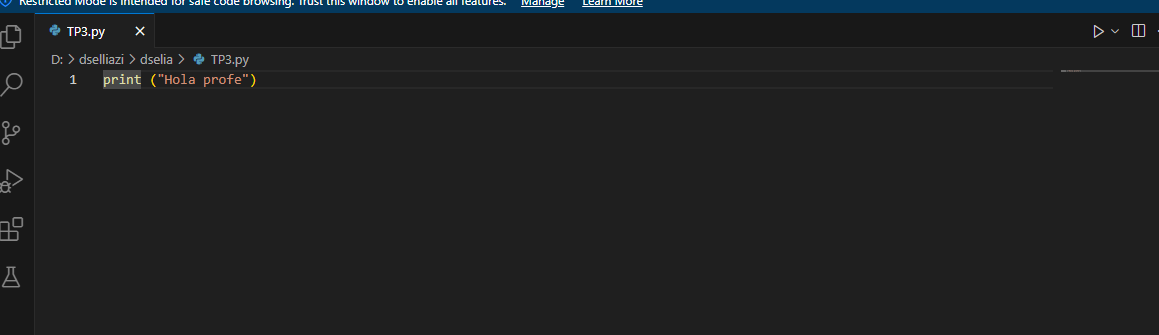




1. Volver al primer commit, ósea Primera versión, usando checkout:



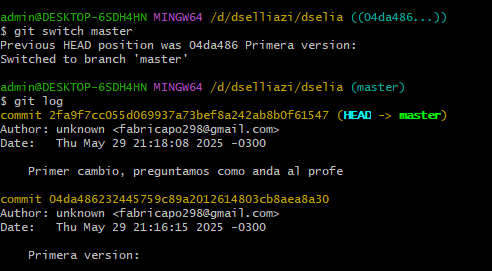


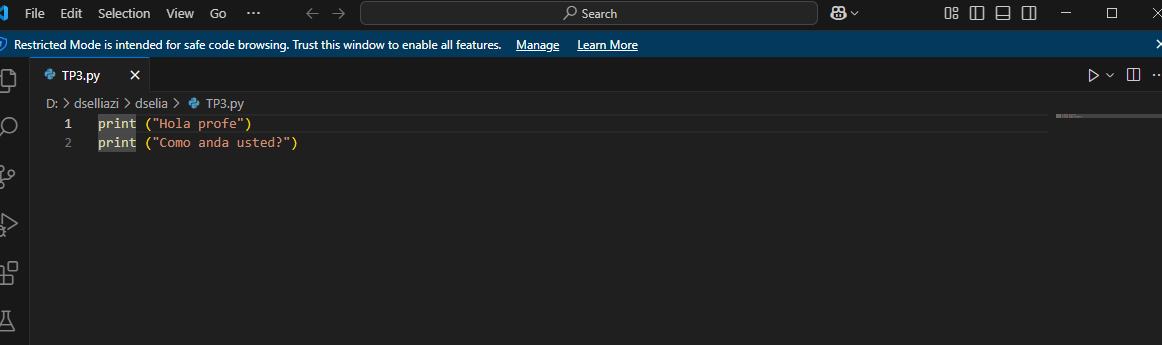


Gestionamos Branch (la rama)



Volvemos a master, es decir, al primer cambio:

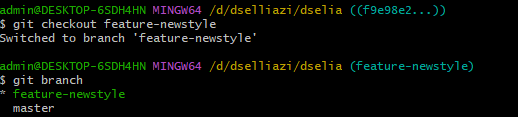




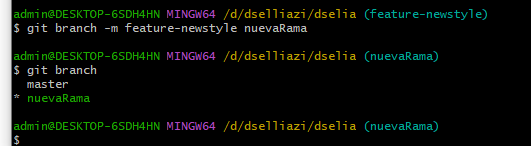
Creamos una nueva rama feature newstyle



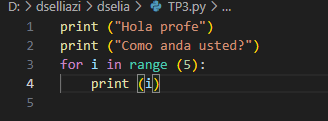
Y cambiamos de rama



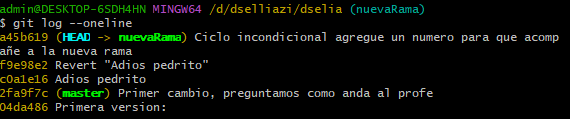
Podemos cambiar el nombre de la rama



Ahora trabajando en la nueva rama agregamos algo



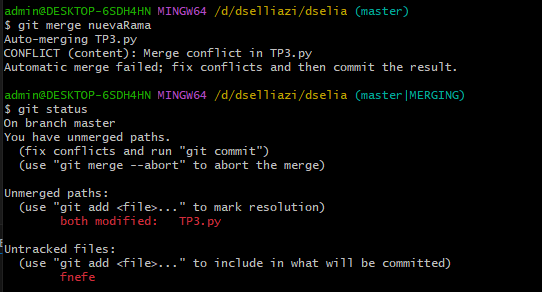
Y le hacemos commit:



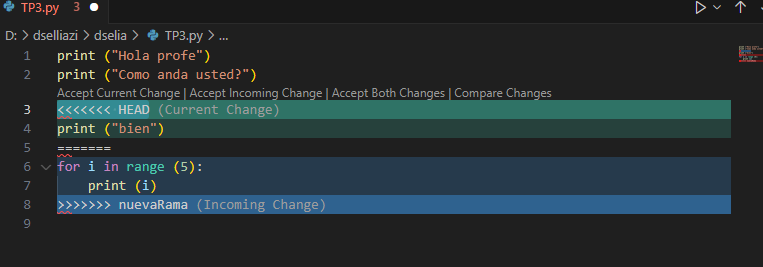
Ahora podemos cambiar de rama y hacer commits sin que se involucren con las otras.

1. Para eso nos posicionamos en master.

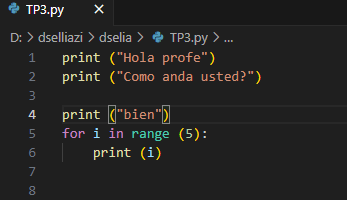


Creamos un commit que diga bien en la tercera línea (línea que usamos para el for), y surgio un conflicto

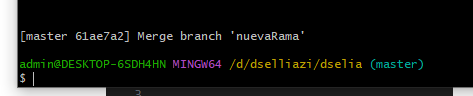
Entonces se modificó el código de ambos marcando a que rama pertenecen(a partir de la línea que coinciden) .



Eliminamos los marcadores y hacemos un comit y se juntaron ambos códigos

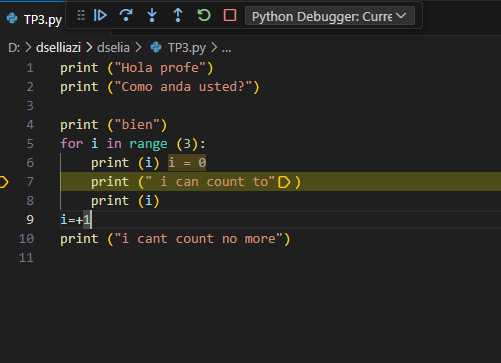


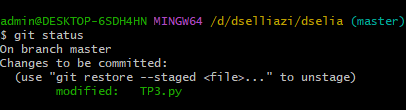
Y ahora ya podemos hacer commit



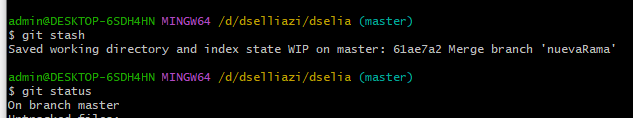
No habrá conflicto siempre y cuando no se reescriban en las mismas líneas de código. Aunque también hay conflictos mas grandes.

1. Para el stash vamos a modificar el código. Lo agregaremos, pero veremos como evitar perderlo sin necesidad de un commit.





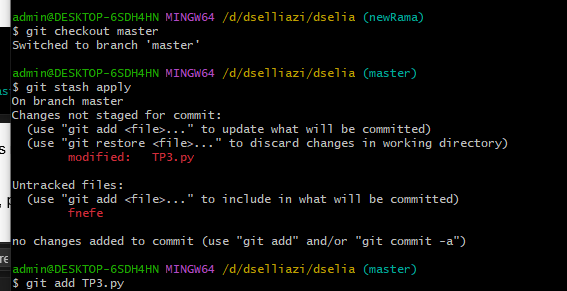
Y se guarda en el stash



Creamos una nueva rama aparte



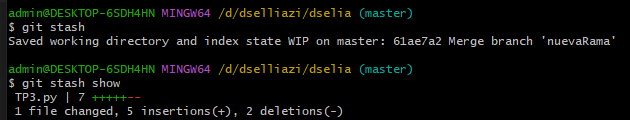
Y después de hacer otros cambios volvemos a master, y para poner lo del stash tenemos que poner git stash apply



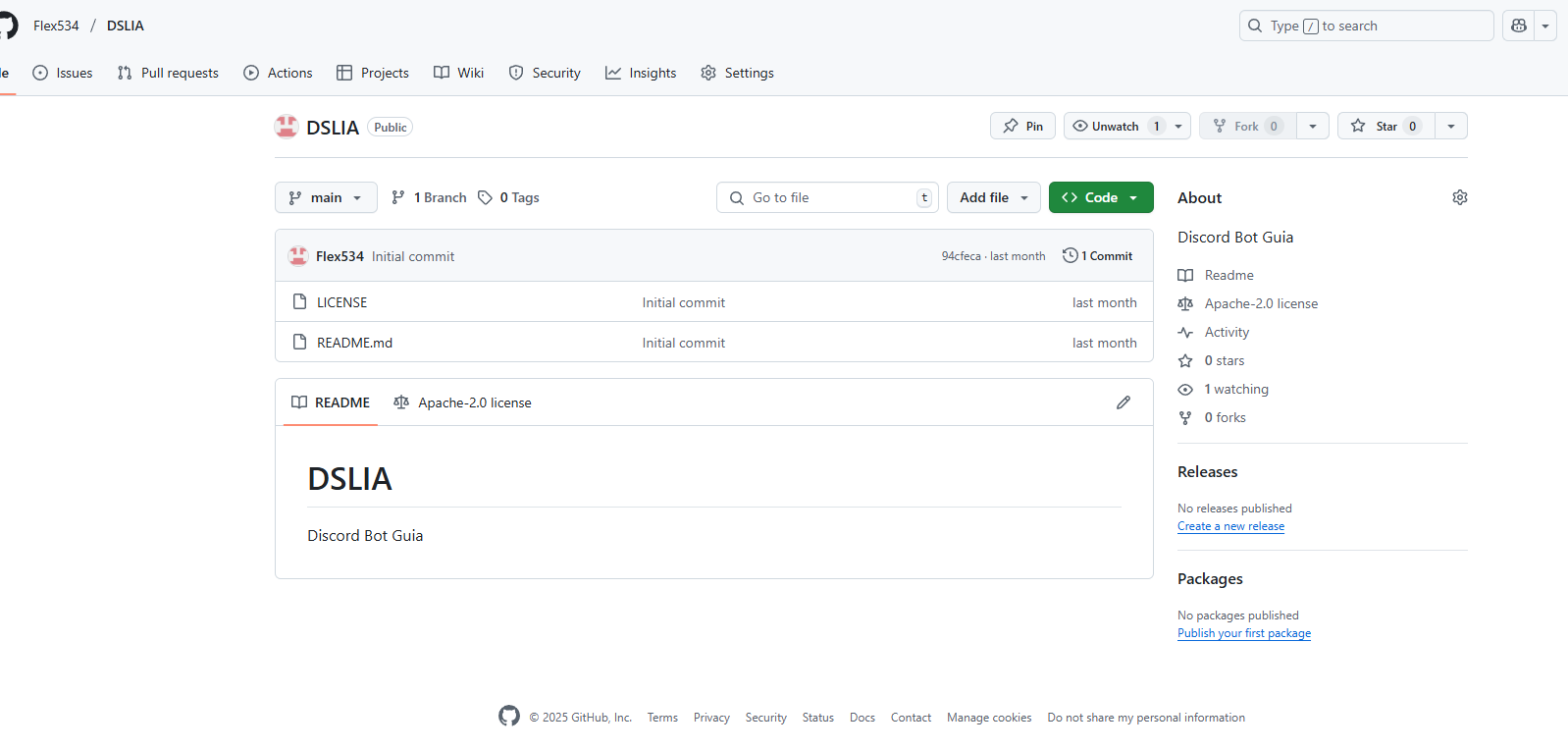
De esa manera recuperamos los cambios que tuvimos en el stash. Y si queremos borrar lo del stash, ponemos git stash drop



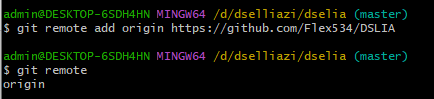
Podemos guardar muchos stash, podemos mostrarlos con git stash show o git stash list





1. Para gestionar un repositorio remoto, tenemos que poner git remote add (nombre del remote) y el link del repositorio en la nube. Usaremos el de dselia, que es el proyecto que estamos desarrollando. 

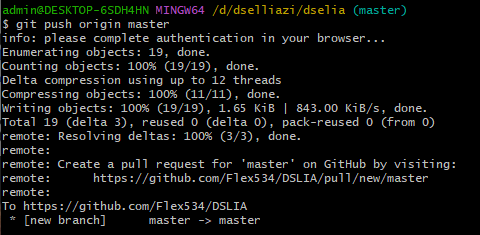
Usamos git remote



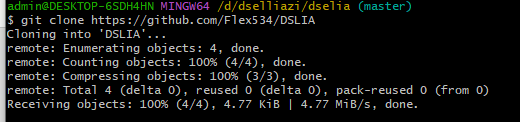
Y listo ya lo tenemos.

1. Usaremos el comando git push para subir el repositorio en cuestión de la rama principal, en este caso, master.

También hay que autenticarse para ver si tiene permiso de poder subir repositorios a este proyecto.

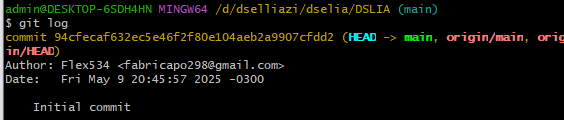


1. Podemos clonar el repositorio que estuvimos trabajando en github para poder tenerlo en nuestra computadora local. Solo hay que poner git clone y el link del repositorio.

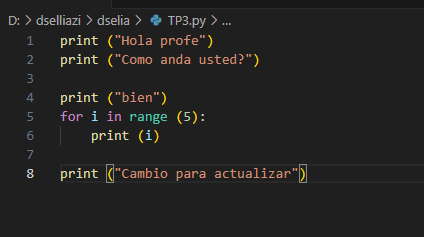


De manera que ya podemos acceder al mismo con todos los commits que se hayan hecho alguna vez

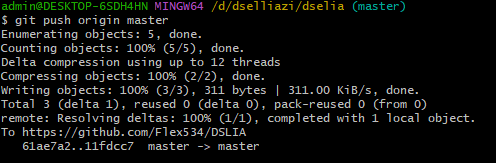


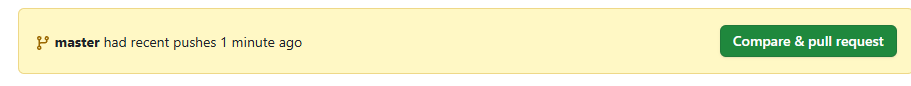


1. El pull es para actualizar directamente del github, asi que lo que haremos será mandar una actualización con push y traer la versión actualizada. Para ello agregamos al código:



La mandamos después del commit.





Nos posicionamos en la carpeta donde sacamos el repositorio remoto.

Y hacemos git pull de manera que tenemos el ultimo commit subido:

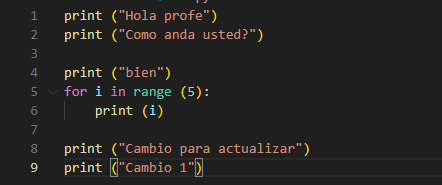


Si hacemos un pull a pesar de que no hubo cambios, se podrá ver que dice “ya está actualizado”

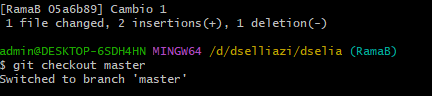


El git fetch funciona para fijarse si hay actualizaciones y si las tuviera, las descarga automáticamente.

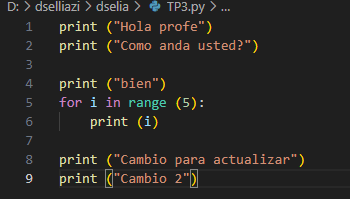
1. Rebase tiene casi la misma función que un merge, solo que mas destructivo, ya que la rama en la q se trabaja desaparece. Para probarlo haremos cambios en el código y crearemos una rama aparte



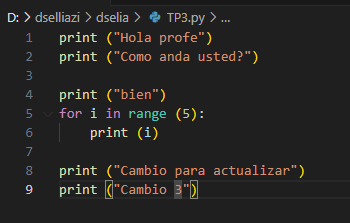
Despues del commit volvemos al master

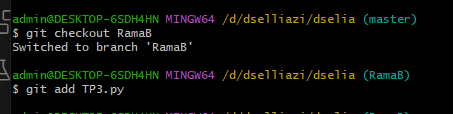


Hacemos un commit de otro cambio

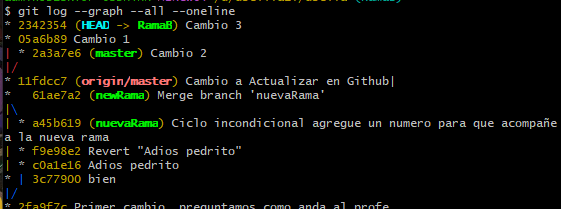


Volvemos a la rama B y hacemos otro cambio

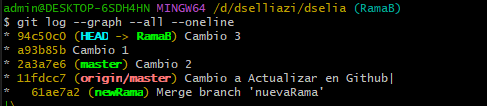




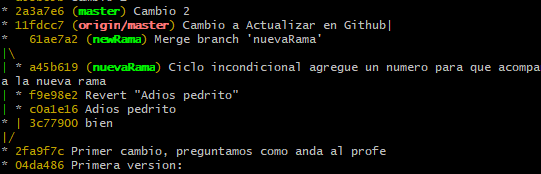
Nos quedo algo asi una vez haya hecho los comit



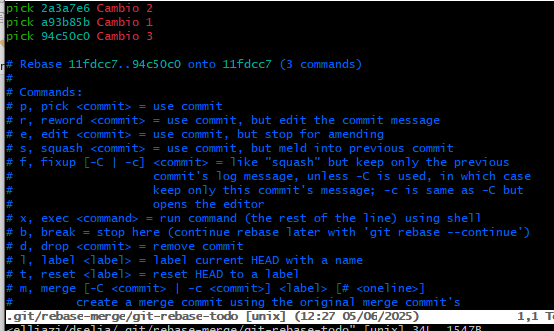
Ahora si aplicamos rebase master , todos los commits formaran parte de RamaB



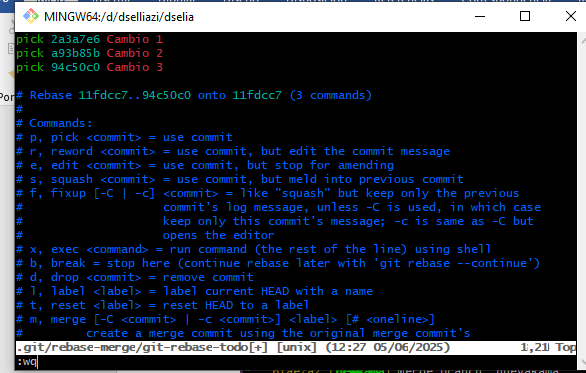
Y asi quedaba con merge



1. Ahora los últimos 3 commits que hicimos vamos a hacerlos un commit, para ello usaremos



Primero cambiamos el lugar del commit asi esta ordenado



Y al hacerlo reemplazamos por squash para unirlos todos. De manera que solo quede

