

Catedra: TIO

Comisión N°1

**22 - 10 - 2020**

Profesores:

* Alonso Diego
* Varona Braian

Integrantes:

* Choque Eliana
* Jorge Eugenia
* Laborde Julieta
* Moreno Jonatan
* Nocetti Francisco

Informe

Trabajo Practico Especial

GIT

**Índice**

**Introducción**…………………………………………………………………………..2

**Desarrollo**……………………………………………………………………………...3

Git clone………………………………………………………………………...4

Git add- commit- push…………………………………………………………5

.gitignore………………………………………………………………………..5

Branch…………………………………………………………………………..6

Ramas Remotas……………………………………………………….6

Ramas Locales………………………………………………………...7

Eliminar – Renombrar archivos………………………………………………8

Cambios git …………………………………………………………………….8

**Conclusión**…………………………………………………………………………….9

**Introducción**

El presente informe tratara sobre el uso de git en diferentes proyectos como una herramienta de control de versiones, que sirve para el trabajo en equipo en entornos de desarrollo de software, y a su vez el uso de Trello para la administración de dichos proyectos.

Para realizar el proyecto, se trabajó con archivos de diferentes lenguajes de programación, como HTML, CSS, JS y JAVA, además se incorporó un archivo de Excel con la resolución de unos de los trabajos requeridos por la catedra de TMC.

El objetivo del trabajo es mostrar cómo se utiliza git y cuáles son las ventajas que proporciona, tanto para el trabajo en equipo, como para gestionar eficientemente grandes proyectos almacenando los archivos en los repositorios de git.

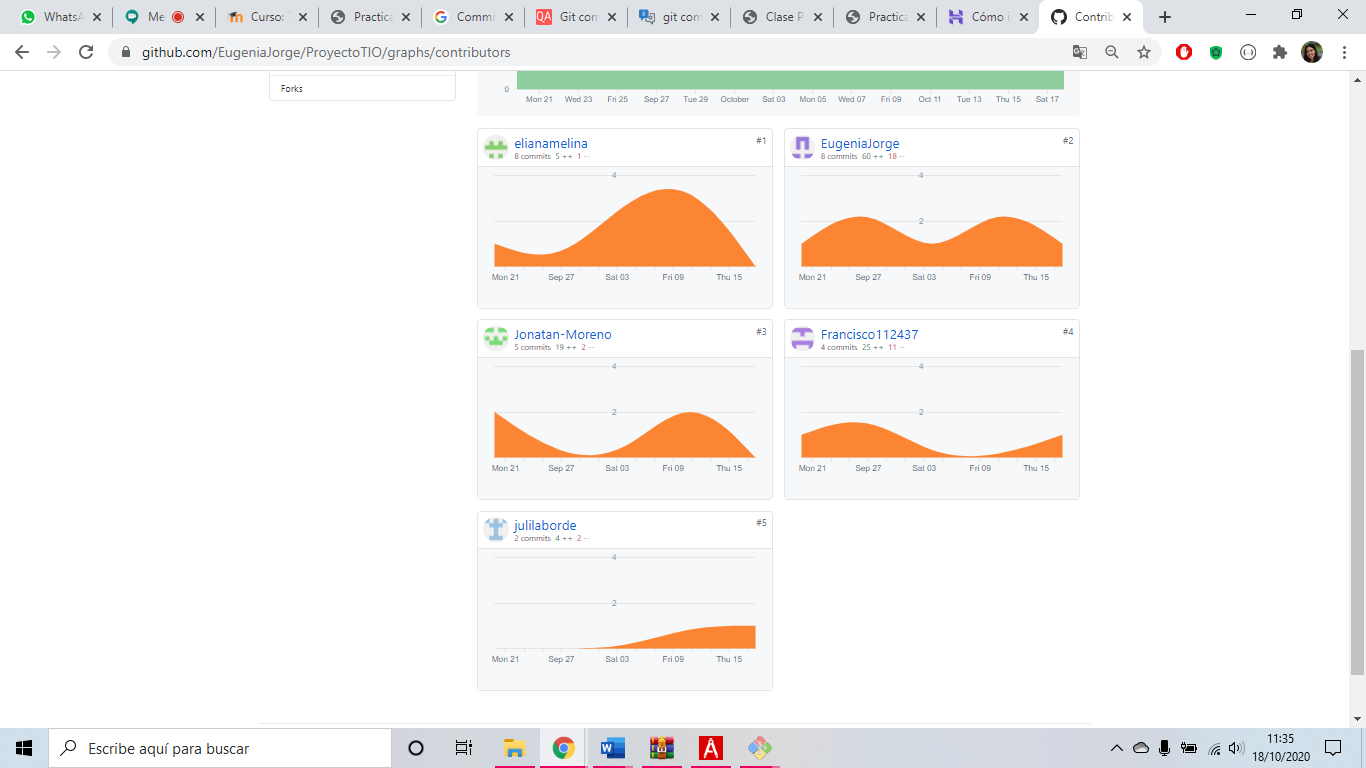
**Desarrollo**

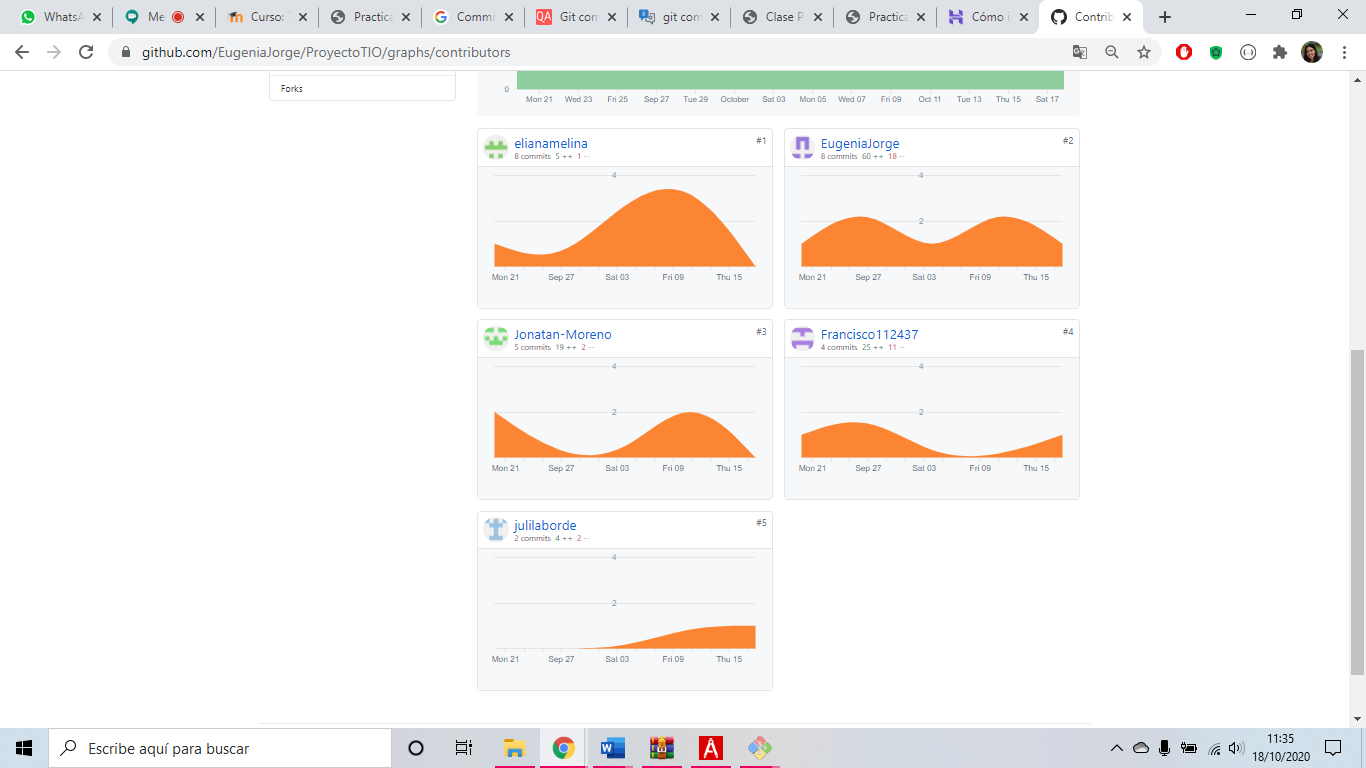
Como se mencionó anteriormente se decidió trabajar con diferentes archivos, cada uno asignado a los diferentes integrantes del grupo.

Como primer paso se creó un repositorio público en GitHub, con los integrantes como colaboradores:

<https://github.com/EugeniaJorge/ProyectoTIO>

La imagen a continuación nos muestra la cantidad total de integrantes con la actividad realizada por cada uno:





Como segundo paso se decidió crear un tablero de Trello para coordinar el proyecto y asignar las diferentes responsabilidades de cada uno de los miembros, con etiquetas de diferentes colores que marcan la prioridad de cada actividad a realizar:

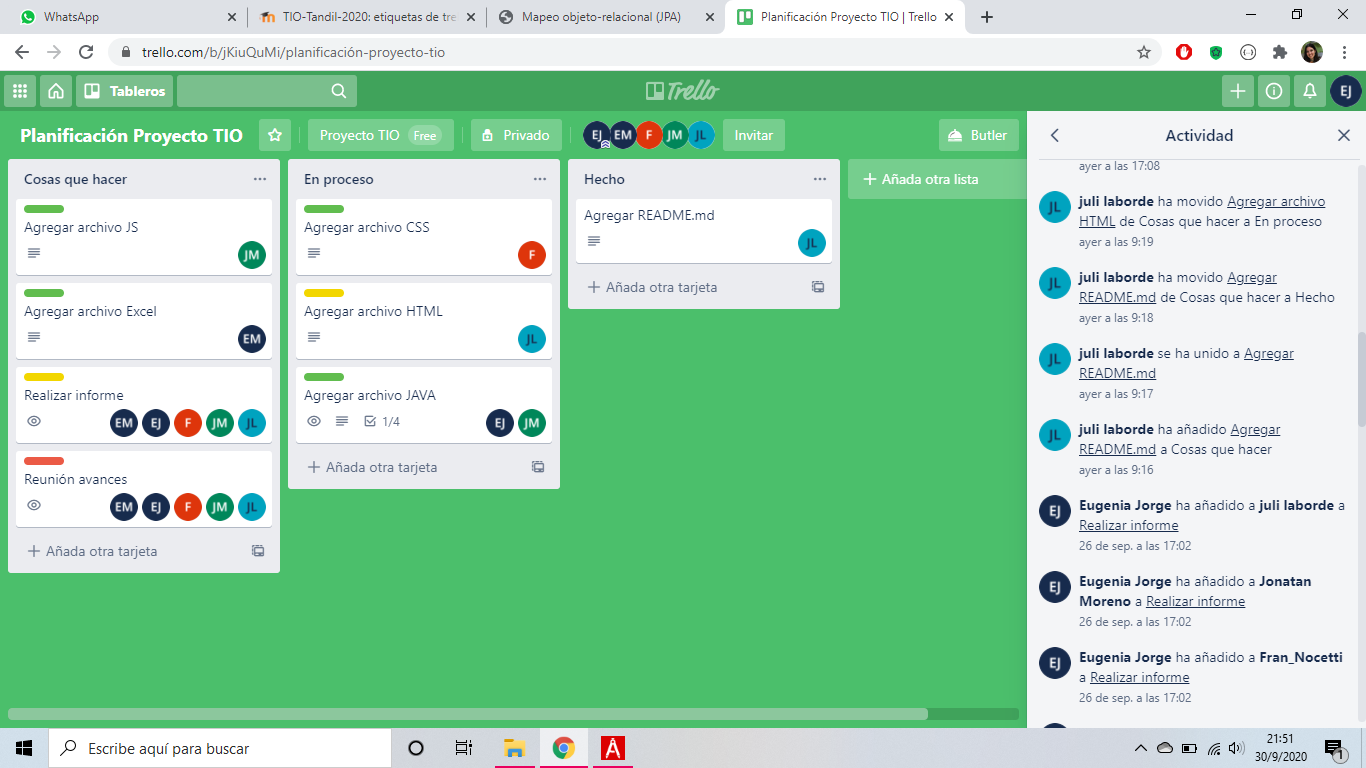
Verde 🡪 Prioridad baja

Amarillo🡪 Prioridad Intermedia

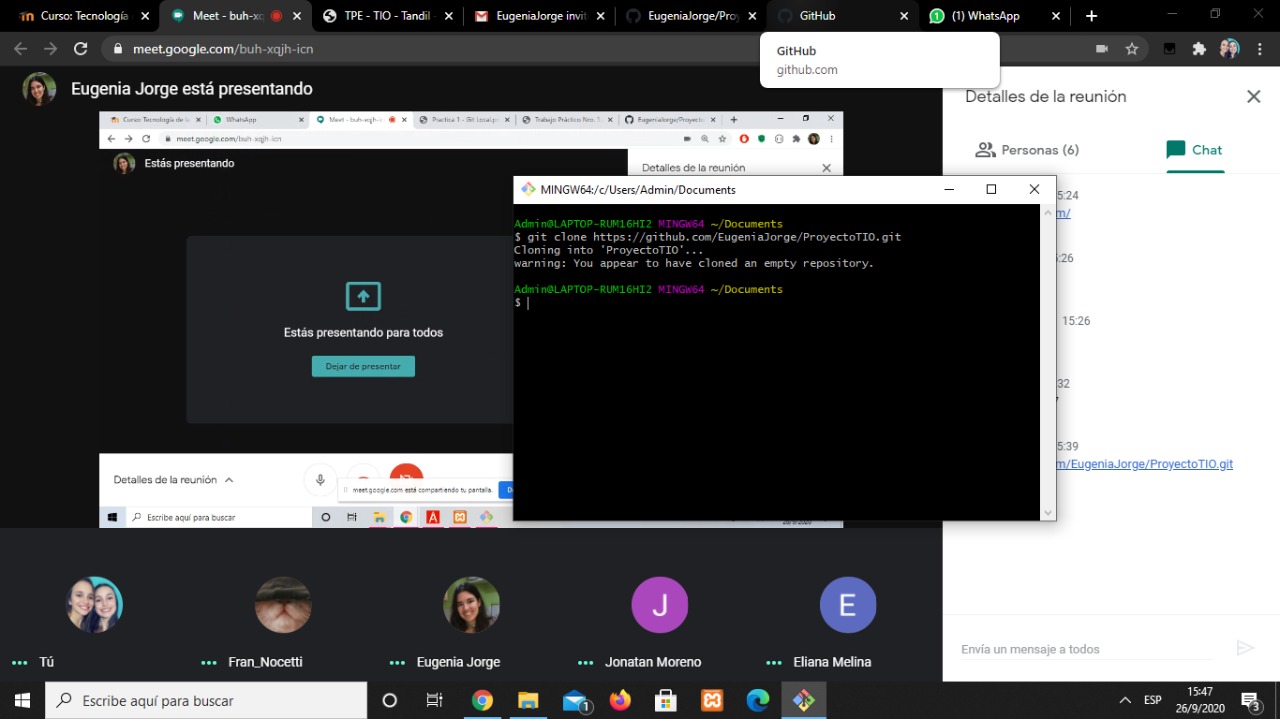
Rojo 🡪Prioridad Alta

<https://trello.com/b/jKiuQuMi/planificaci%C3%B3n-proyecto-tio>

En la siguiente imagen se muestra el tablero de Trello:



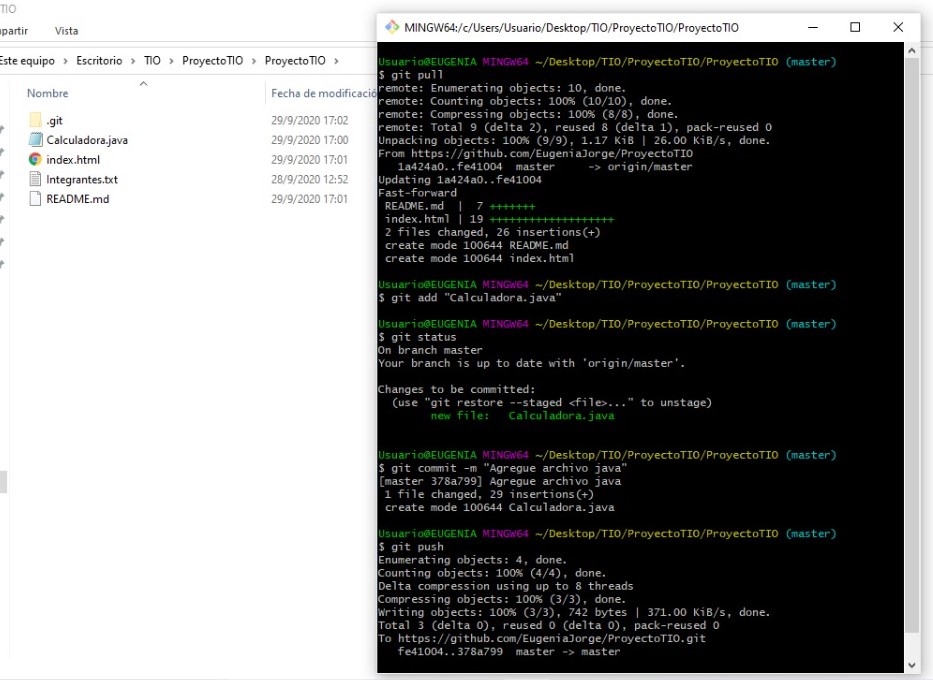
En el siguiente paso se clono el repositorio Remoto de GitHub a un repositorio local, utilizando el comando **git clone**, como se pude ver a continuación:



Una vez realizado esto, cada uno de los integrantes llevo a cabo sus actividades marcadas en Trello.

A la hora de subir archivos al repositorio, se utilizaron los comandos:

**Git add + git commit + git push**

1. Se agrega el archivo “Calculadora.java” a la carpeta donde fue clonado el repositorio.
2. Se realiza un git add, para agregar el archivo al área de preparación.
3. Se procede a realizar un commit con un mensaje representativo de los cambios que se van a confirmar.
4. Finalmente se realiza git push para subir esos archivos al master, en el repositorio remoto.

Siempre se utiliza git status para controlar el estado del repositorio, y antes de modificar cualquier archivo se utiliza git pull para traer los cambios del repositorio remoto al local.

**.gitignore**

Se utilizo el archivo *.gitignore* para indicarle a git que el contenido de determinadas carpetas u archivos no sea tenido en cuenta para el seguimiento.

Por ej.: para informar que a la carpeta “Principal” no debe hacerle un seguimiento, el archivo se escribió con la siguiente línea: */Principal/*

**Creación de Branch**

Rama Remota

Se decidió la creación de una rama remota para que el equipo trabaje en paralelo sobre los archivos específicos de Web (HTML, CSS y JS), sin afectar a la rama master.

Para la creación de esta, primero se creó una rama local con **git checkout -b Web** y posteriormente se utilizó el comando **git push -u origin Web** para subir la rama al repositorio remoto.

Luego de esto, para que los demás integrantes puedan trabajar localmente sobre dicha rama, es conveniente utilizar **git checkout -b Web origin/Web** con este comando se linkea la rama remota con el repositorio local.

El equipo utilizo otros comandos:

1. Se creo una rama local con el mismo nombre que la rama remota.
2. Se realizo Git pull para traer todos los cambios del repositorio remoto al local.
3. Posteriormente Git checkout Web para apuntar a la rama Web, que como se llamaba exactamente igual que la rama remota, git interpreta que son lo mismo y las linkea de forma automática.
4. Por último, confirmados los cambios realizados, se procedió a subirlos a la rama remota con el comando git push -u origin Web.

Estos comandos trajeron diversos problemas:

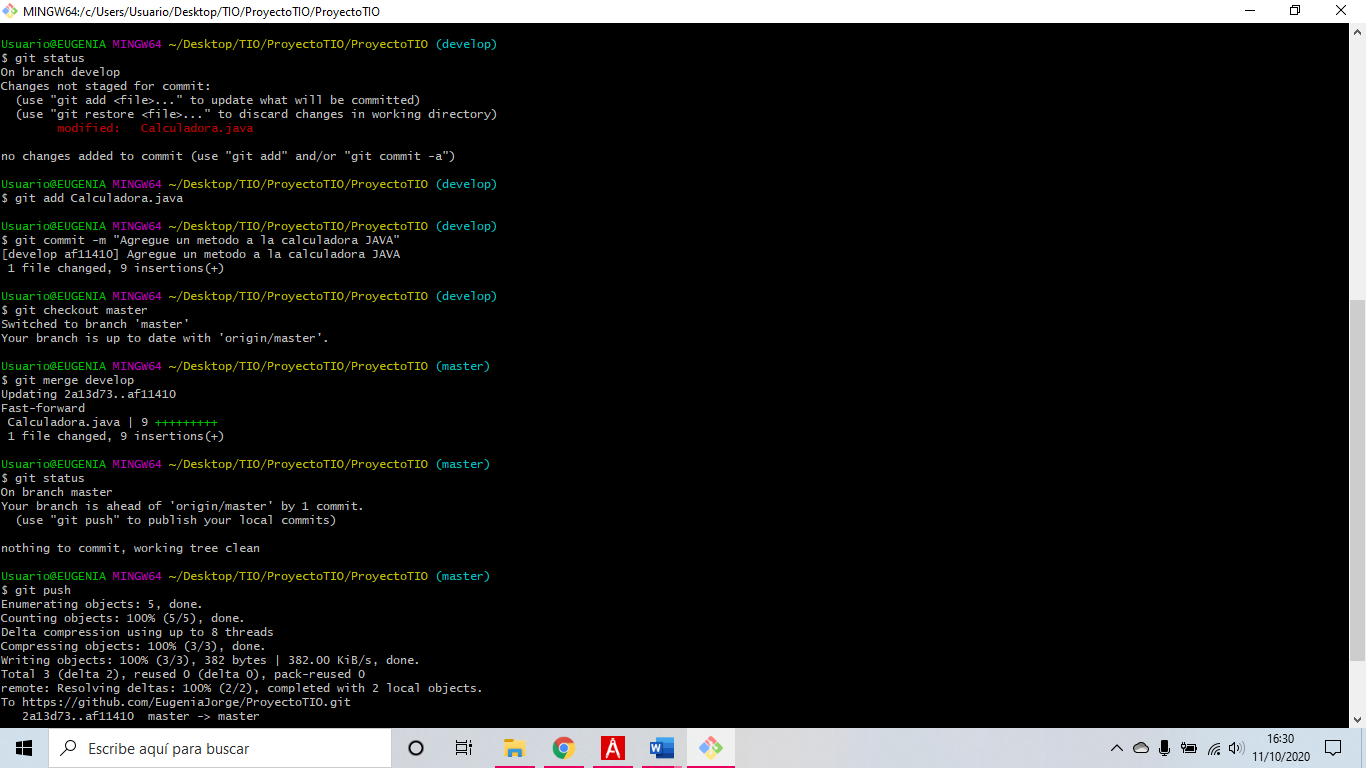
En primer lugar, por equivocación se utilizó otro nombre (“web”- con minúscula) distinto al de la rama remota “Web”, por lo que no se linkeo la rama, por ende, al realizar un push se creó otra rama Remota adicional a las ya creadas, esto produjo que no se trabaje sobre la rama que se quería.

En segundo lugar, otro de los problemas fue que como la rama master presentaba commits por delante de la rama secundaria creada localmente, se generaron conflictos. Para resolverlos se realizaron los commit correspondientes para fusionar las ramas. Como contra de esto, podemos mencionar que se tuvieron que subir confirmaciones de archivos que no eran necesarios para la rama Web.

El último paso que se realizó con la rama remota fue la fusión de la misma con la rama master, para ello, primero se apuntó a la rama master y luego se trabajó con el comando git merge Web.

Ramas locales

Se utilizaron ramas locales para agregar nuevas funcionalidades a determinados archivos del proyecto, a continuación, se muestra un ejemplo de creación de la rama local, fusión de la misma y su posterior eliminación.

1. Creación: git branch develop.
2. Apuntar a dicha rama: git checkout develop.
3. Confirmaciones: git status para comprobar, git add y git commit.
4. Fusionar con rama master: git checkout master para apuntar a dicha rama, git merge, para fusionar los cambios.
5. Eliminar: una vez absorbidas todas las confirmaciones, se procede a eliminarla con el comando git branch -d develop.

**Eliminar – Renombrar archivos**

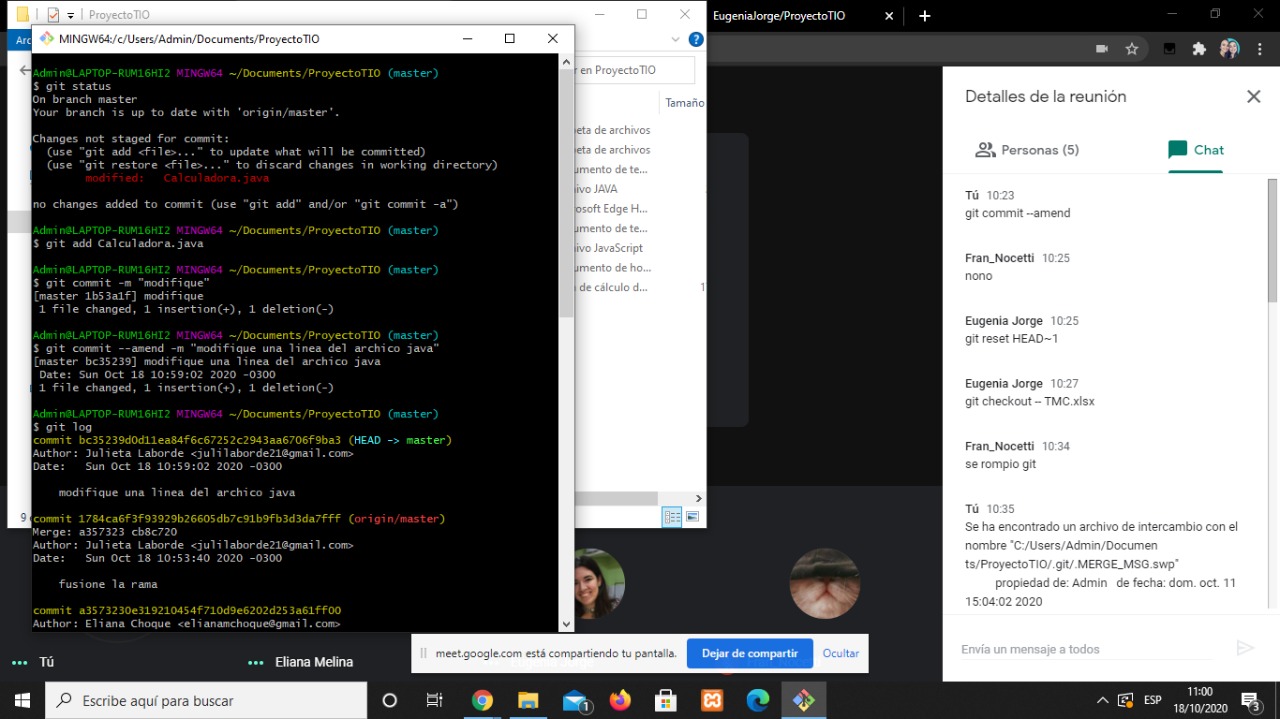
Se utilizo git rm “README.md” para informar a git que se eliminó el archivo, y posteriormente se llevó a cabo un commit para confirmar dicho cambio.

Para confirmar el cambio de nombre de un archivo se utilizó el comando git mv “git.css” “style.css”.

**Cambios en git**

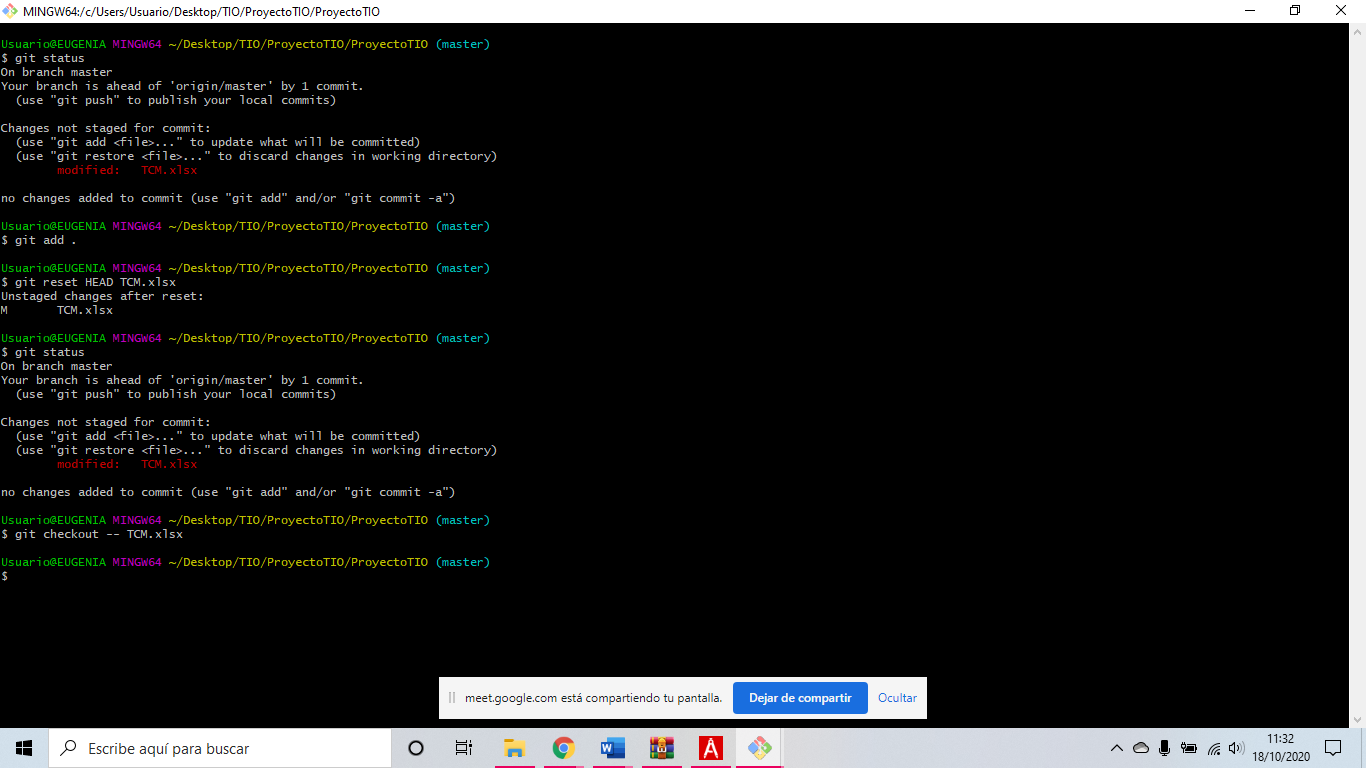
Modificar el mensaje del ultimo commit

Para modificar los mensajes de los commit, se utilizó el comando git commit –-amend -m “mensaje”, para que sobrescriba el mensaje del ultimo commit realizado.



Deshacer un archivo que se encuentre en el área de preparación.

Se utilizo el comando git reset HEAD <archivo>, para retirar la modificación del área de preparación.



Deshacer una confirmación

En caso de que se decida volver a la versión anterior de un archivo, se debe utilizar **git reset HEAD~1**, con esto se elimina la última confirmación, pero los cambios de este último commit no se eliminan, para eliminar las modificaciones en forma definitiva se utiliza además el comando **git checkout -- <archivo>.**

**Conclusión**

Para concluir podemos decir que fue una gran experiencia trabajar con git. Nos permitió facilitar el trabajo colaborativo, trabajar con distintas versiones de un proyecto, además de guardar todos los cambios realizados por cada integrante, lo que permite hacer un seguimiento individual y de los errores.

Podemos decir que en un principio nos costó trabajar con git porque era una herramienta nueva que no conocíamos del todo. Los distintos problemas que se plantearon nos sirvieron para entender aún más el funcionamiento de esta herramienta.

Este trabajo nos pareció super interesante ya que nos prepara para el ámbito laboral en el que git es muy utilizado para el desarrollo de software, por sus diferentes beneficios, en los que podemos mencionar la facilidad que brinda para trabajar en equipo, y compartir código, volver a versiones anteriores de un proyecto y generar distintas áreas de trabajo, para corregir errores o desarrollar nuevas características sin afectar el proyecto que se encuentre en master.