

**TUDAI**

**Cátedra:** ​**Tecnología de la Información en las Organizaciones**

**Curso:**​ **1° año**

**Ciclo lectivo: 2020**​

**Trabajo Práctico Especial - Git**

**Integrantes:**

* **Facundo De Luca**
* **Luis Rosiello,**
* **Maximiliano Otharan,**
* **Cristian Amici.**
* **Alias: (**Los soldados del General)

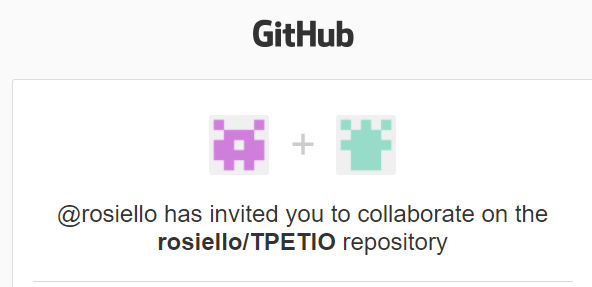
***INTRODUCCIÓN:***

En el siguiente informe, se explicará cómo se llevó a cabo un trabajo colaborativo sobre una página web utilizando GIT, TRELLO y DISCORD como herramientas informáticas de comunicación e intercambio de información entre los colaboradores.

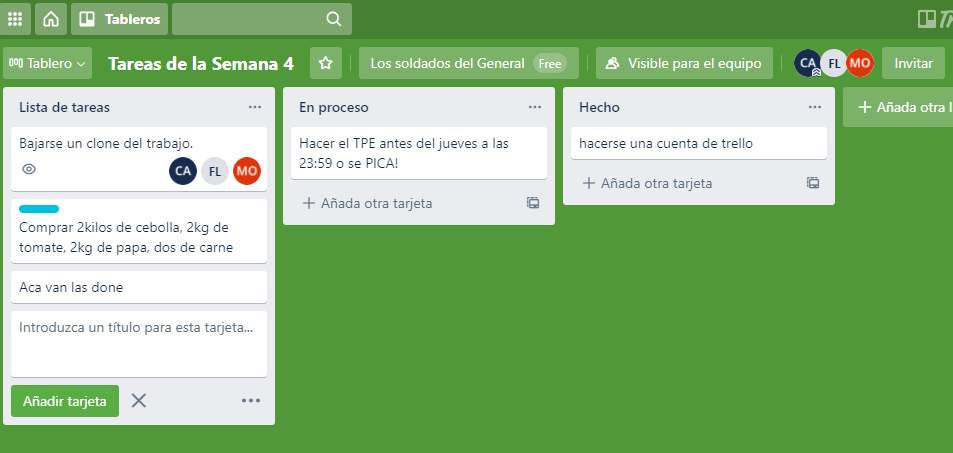
El objetivo del trabajo fue “mejorar” una página WEB vinculada al arte, utilizando principalmente lenguaje PHP, en el que destacaron un trabajo hecho con lenguaje smarty plasmado en archivos .tpl, también se dio estilo propio con un archivo .css, también utilizaron archivos .html e incorporaron archivos .jpg. Para agregar funcionalidades a la página se trabajó con JavaScript. De esta forma a priori ya se puede observar la amplia versatilidad y capacidad de los integrantes del grupo.

***Realizando el Trabajo:***

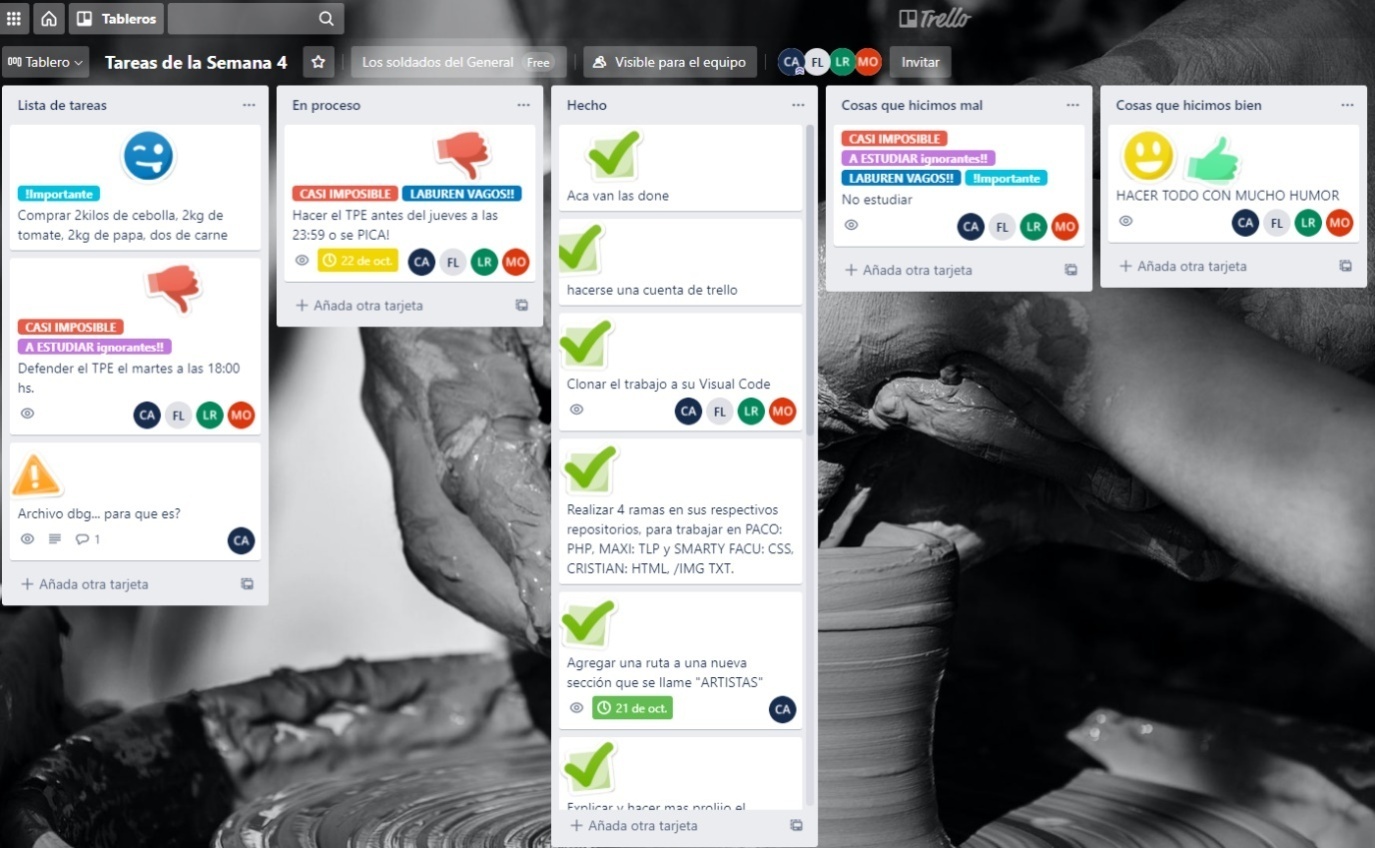
Comenzaron realizando un repositorio remoto, donde el principal MASTER (LUIS ROSIELLO, desde ahora PACO) compartió una página WEB en la que estaba trabajando, con la intención de que el resto de sus amigues pueda ayudarle y aportar mejores y nuevas perspectivas a su diseño y estilo (Pobre Iluso). Recibieron el resto de compañeros una invitación a dicho repositorio, el repositorio se inicio a través del comando **“git init”** en su capeta TPETIO utilizando la consola. Se envió el repositorio de forma remota a GitHub a traves del comando “git remote add origin <https://github.com/rosiello/TPETIO.git>” (no tenemos captura del mismo, perdón)



A partir de ahí, decidieron que era conveniente tener un cronograma de tareas para que no surgieran problemas a la hora de trabajar colaborativamente y a su vez poder proyectar un mismo objetivo y plan a seguir. Para ello decidieron trabajar con la plataforma TRELLO, ya que ella misma cumplía absolutamente con CASI todo lo necesario para realizar este tipo de actividades, salvo por algo… faltaba comunicación en tiempo real, para ello utilizaron un mecanismo de reuniones por DISCORD, plataforma que usaron brindo la posibilidad de mantener la comunicación activa a la hora de diagramar las reuniones y plantear los siguientes objetivos a cumplir:

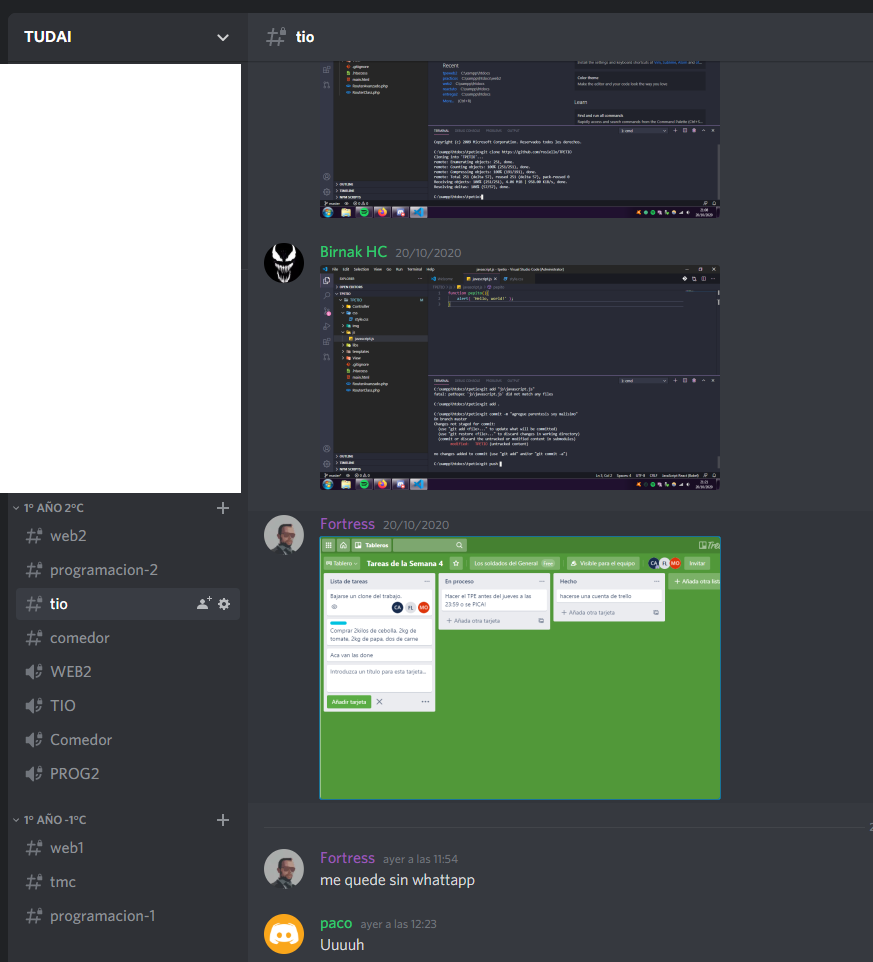


En este momento fue donde surgió el **primer problema**: El equipo de trabajo no sabía usar esta plataforma, como se ve en la imagen comenzaron a probar entre todos como utilizarla, para ello también investigaron y recordaron todo lo trabajado en clase, el resultado fue sorprendente: Se crean “Los soldados del General”



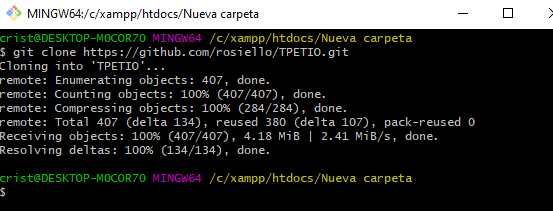
Pudieron no solo agregar tareas relacionadas al trabajo en equipo, sino también etiquetas con mensajes super importantes, stikers (pegatinas) para tener una visual rápida de lo que hay o no que hacer y buscar más rápido aquello que debían de dar importancia.

Mientras todo el trabajo colaborativo se plasmaba en cumplir las tareas, también mantenían comunicación por un canal de voz utilizando como antes dijimos DISCORD:

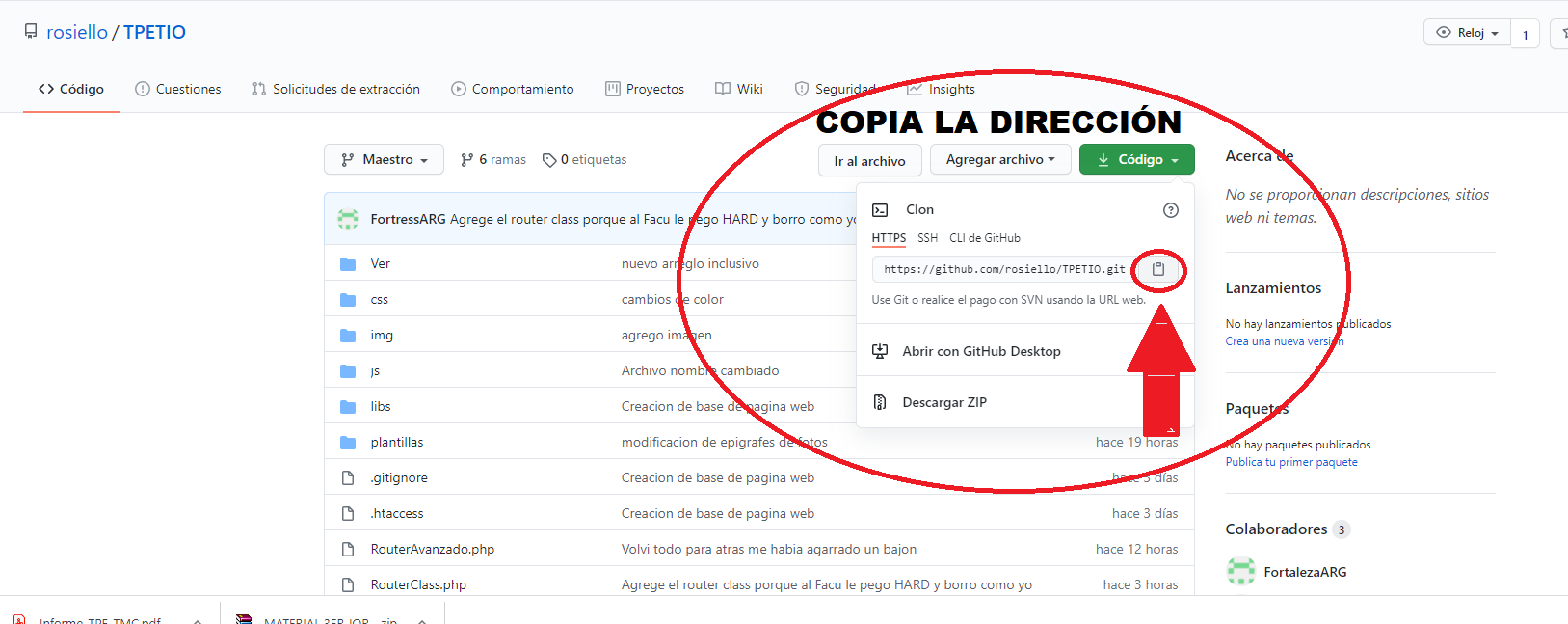


Y así fue como comenzó esta hermosa historia de trabajo en equipo…

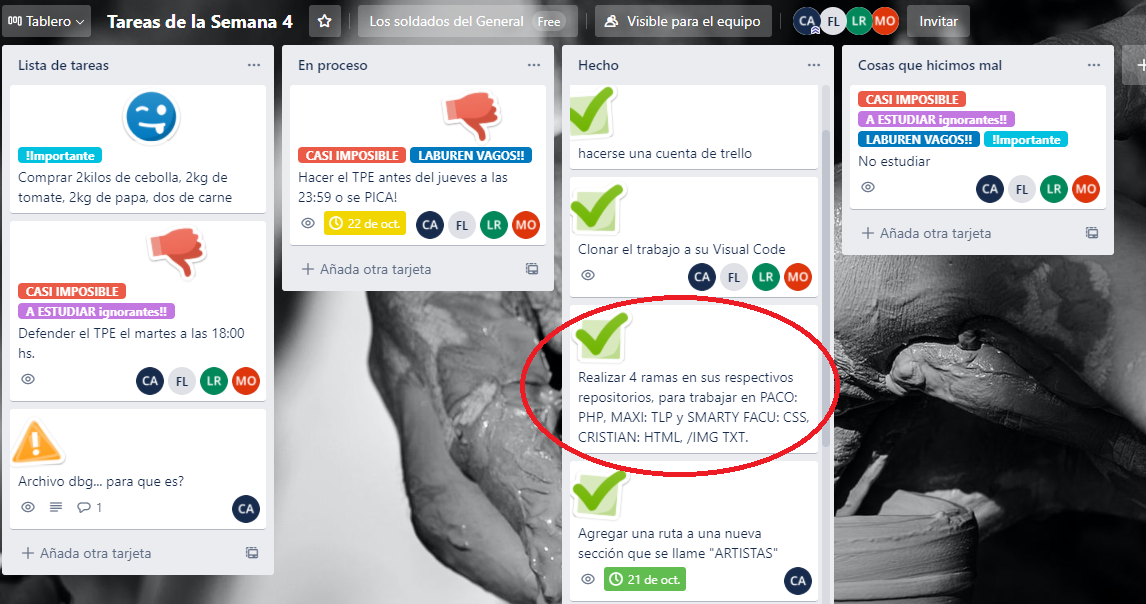


La primea tarea que se asignó a cada miembro del equipo fue clonar el repositorio a la carpeta local de trabajo de cada miembro. Esa tarea la realizaron utilizando el comando **gitclone** de consola de Git Bash:

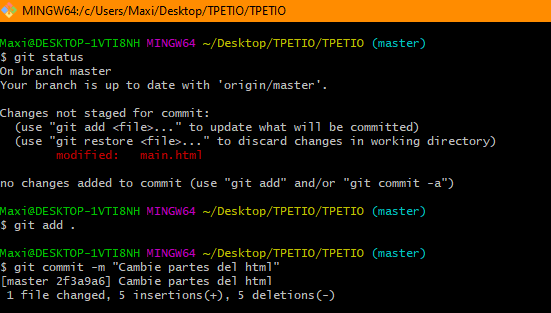
Para poder traer el archivo se tuvo que copiar la dirección del mismo en la carpeta de GITHUB como se muestra a continuación y pasarle al git local esa dirección:



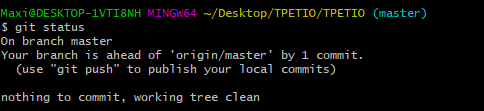
La siguiente tarea que se asigno fue dividiéndose el lenguaje a trabajar:



A modo de explicación y resumen de los comandos utilizados para no ser repetitivos, cada vez que algún miembro del equipo realizó una modificación en su repositorio local, utilizo los comandos en la consola de **“git status”** para poder ver el estado de su trabajo y conocer las modificaciones realizadas que todavía no figuran en la zona de intercambio, la consola nos muestra con color rojo, aquellos archivos modificados y no incorporados. Para ellos luego se ejecuta en la consola el comando **“git add .”** donde podrá agregar esas modificaciones a la zona de intercambio temporal o staging area. También se puedo realizar el mismo comando con el nombre específico del archivo, pero les fue mucho más cómodo agregar todos los cambios a través del “.”. Al mismo tiempo se confirmaron las modificaciones a través del comando de **git commit -m** “mensaje”, que les permitió dar aviso de los cambios realizados a sus colegas. Es importante tener en cuenta que los mensajes deben siempre ser lo más específico posible para que sus colaboradores puedan entender a lo que se hace referencia, aclarar esto es importante, ya que muchos de los miembros de este informe poco cumplieron con dicho punto.

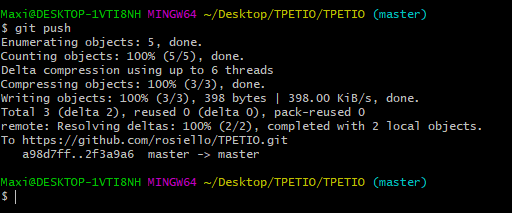
Con estas tres aclaraciones podemos mostrar varios ejemplos de cómo fueron usados esos comandos en diferentes situaciones y los mensajes implementados:

Posteriormente a la confirmación se ejecutó un **“git status”,** con la intención de poder confirmar que todo esta preparado para realizar la incorporación a la rama principal en GITHub.

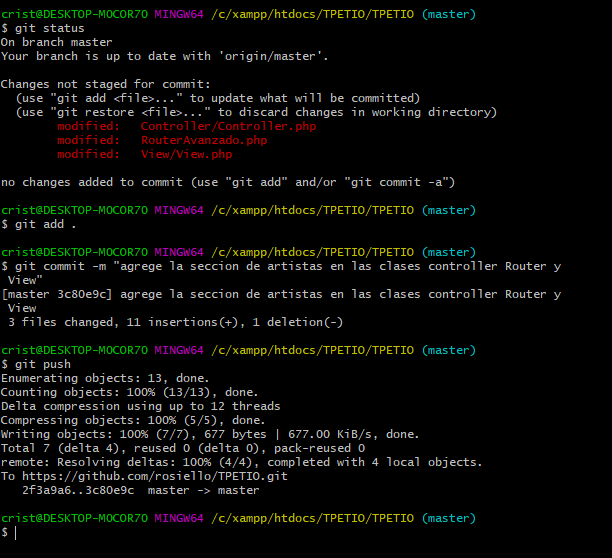


Como se puede apreciar en la imagen ya estaba todo preparado para poder realizar el comando **“git push”** recomendado ya por la consola.

Para realizar las modificaciones y agregado de información a la rama MASTER del repositorio en GITHub se utilizó el comando “**git push”**, el cual está destinado a enviar todos esos cambios para que el resto de los compañeros puedan verlos e incorporarlos.

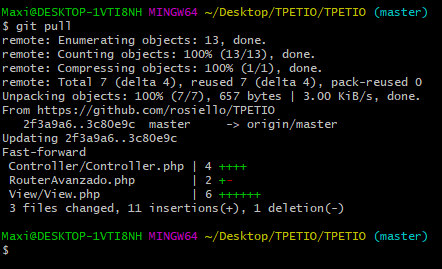


Otro ejemplo de modificaciones realizadas en el trabajo que ejemplifican los comandos explicados anteriormente:



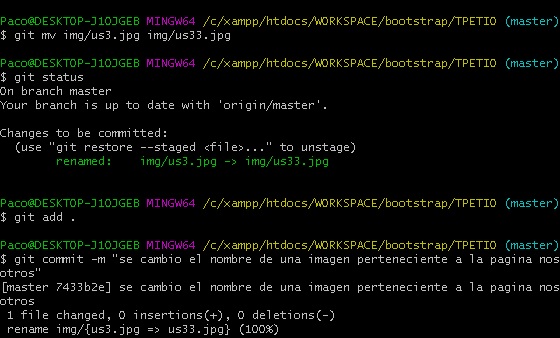
El siguiente comando que más se usó durante el trabajo fue el **“git pull”,** debido a que es indispensable para poder trabajar en equipo, con dicho comando se puede bajar del repositorio remoto en GitHub a un repositorio local y actualizar todo aquello que fue modificado por los colaboradores.

A continuación, podemos ver el ejemplo siguiente:

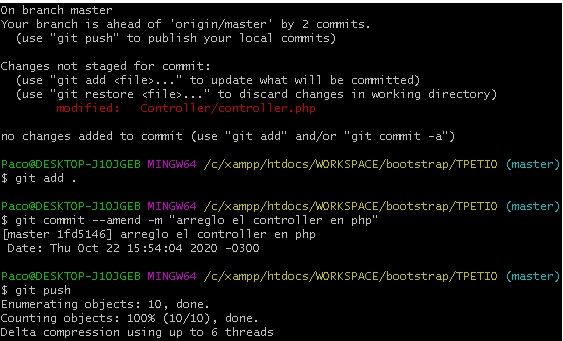


Como se puede ver en cada archivo figura las líneas que se sumaron modificaron y las que se borraron.

Se estuvieron realizando además modificaciones de los nombres de archivos a través de “**git mv”** con el fin de corregir los nombres de fotos que se tienen en cuenta en los templates.

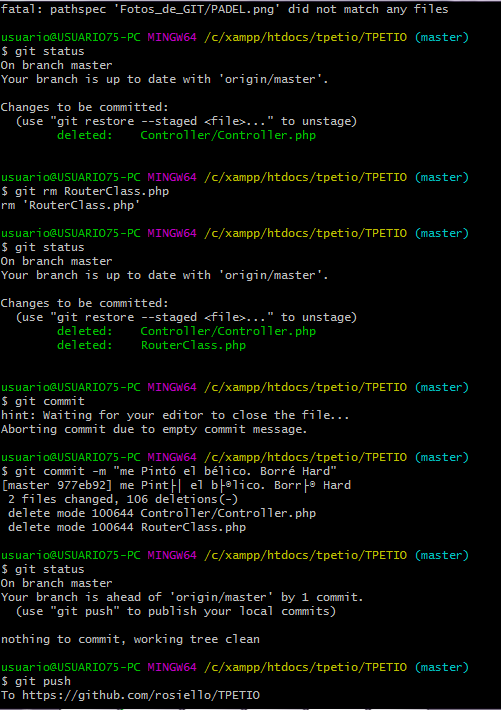


Sin dudas por problemas de interpretación, hubo integrantes que han roto una parte fundamental de la página eliminando el controller.php, por lo que al crearse uno nuevo, durante ese proceso erróneamente se lo nombro con minúscula y se commiteó, al modificar el nombre en el código de las clase Controller, para incorporar ese commiteo previo, se utilizó el método **git commit –amend** como se muestra a continuación.

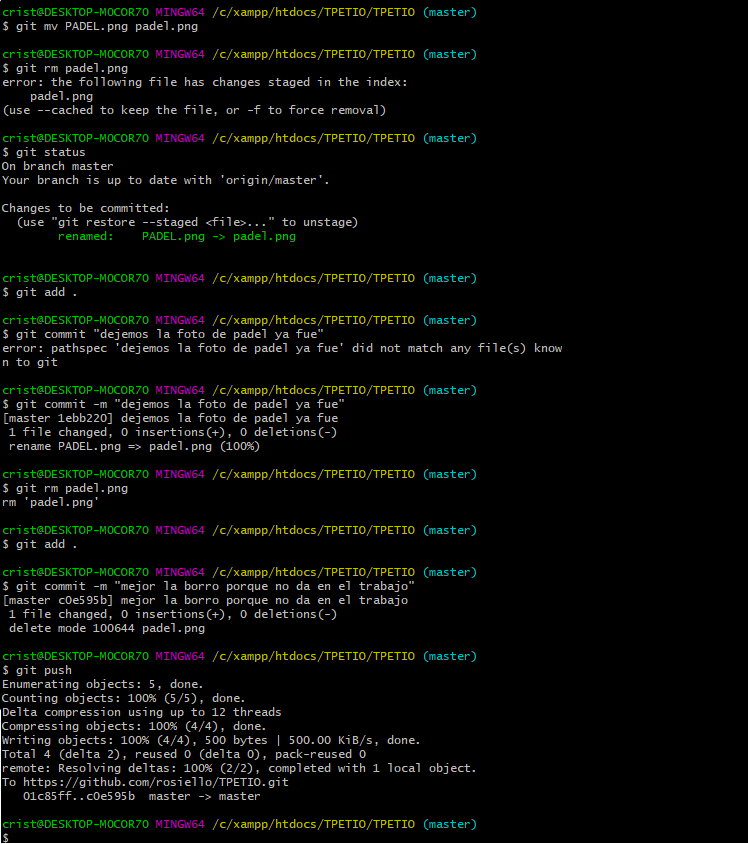


Otro comando que se usó mucho y fue el más peligroso de todos durante el proyecto colaborativo fue el **“git rm (nombreDelArchivo.Extencion)”,** lo que permite borrar cualquier archivo que se desee eliminar. El equipo en situaciones desesperantes de trabajo utilizo este comando en repetidas situaciones, generando una rotura general del proyecto.

Algunos ejemplos:



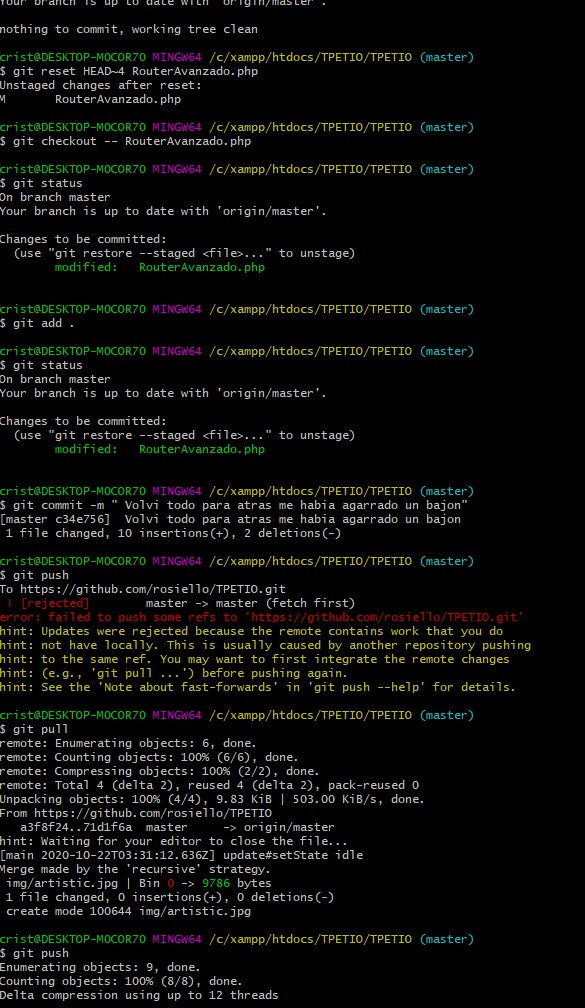
En dicha situación se rompió el archivo PHP por la falta de la clase principal. En el siguiente caso, se incorpora por error una imagen de Padel, se realizan comandos ya nombrados para modificar algunos comportamientos y al final para colaborar con el proyecto de forma seria se elimina la imagen:



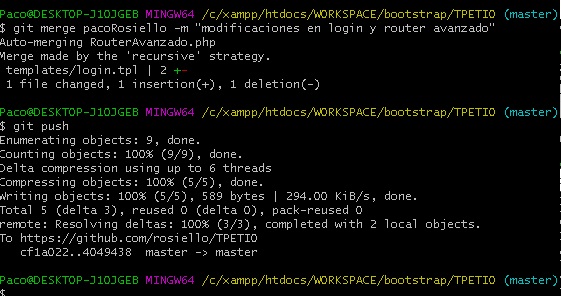
Un comando muy usado por el equipo, fue el “**git reset”** o **“git reset HEAD~(numero) “**

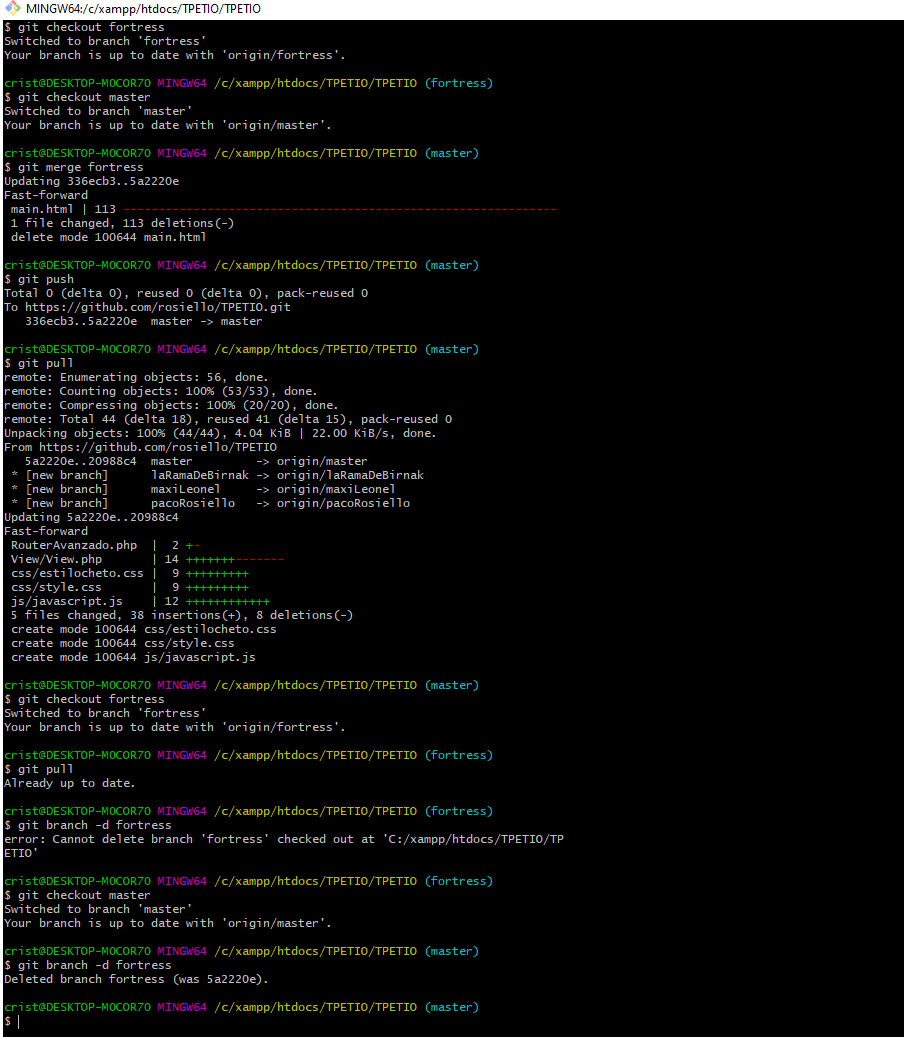
que les permitió volver para atrás esas ganas de borrar todo el trabajo realizadoen esos momentos de frustraciones que no les salía nada. Si utilizan el **“git resest”** se retrocede la rama actual una confirmacion, ahora bien, si utilizan además el HEAD se puede enumerar el momento atrás al que se quiere llegar.

Ejemplo del siguiente caso es el cambio realizado en las rutas del RouterAvanzado.php que por motivos de frustración fue borrado varias confirmaciones atrás.

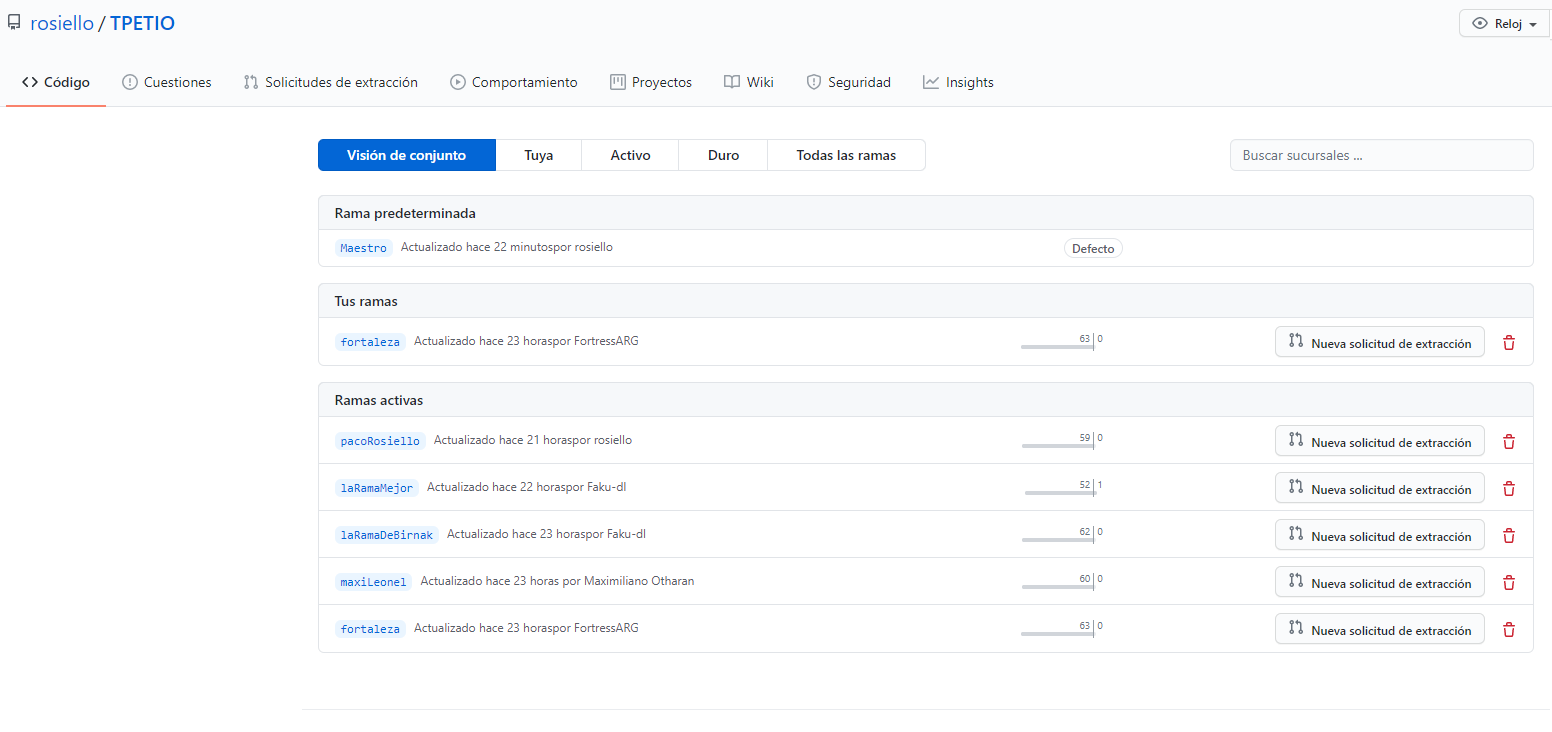


Inevitablemente como una de las soluciones a commiteos precipitados perjudiciales, cada integrante creó una rama local, utilizando los comandos “**checkout”,** al ejecutar la operación de cambio de rama, se actualizan los archivos en el directorio de trabajo para reflejar la versión almacenada en esa rama y se indica a Git que registre todas las confirmaciones nuevas y luego realizó modificaciones a través del método “**git merge”,** el cual incorpora cambios de las confirmaciones nombradas (desde el momento en que sus historiales divergieron de la rama actual) a la rama actual. Este comando lo usa git pull para incorporar cambios de otro repositorio y se puede usar manualmente para fusionar los cambios de una rama a otra. Posteriormente el equipo de trabajo utilizo el comando “**git Branch -d”** para borrar las ramas individuales, debido a que les pareció aburrido trabajar por separado y no poder romper el trabajo del compañero.

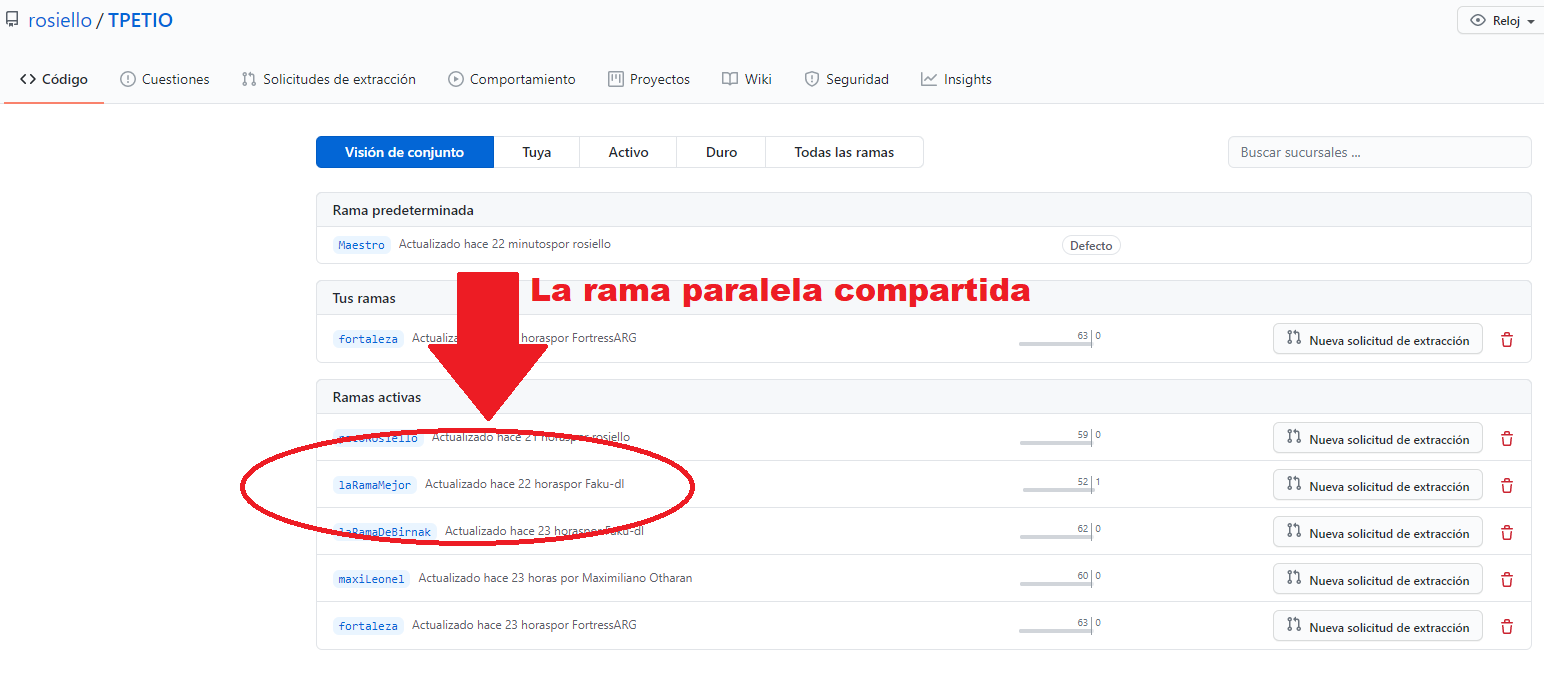




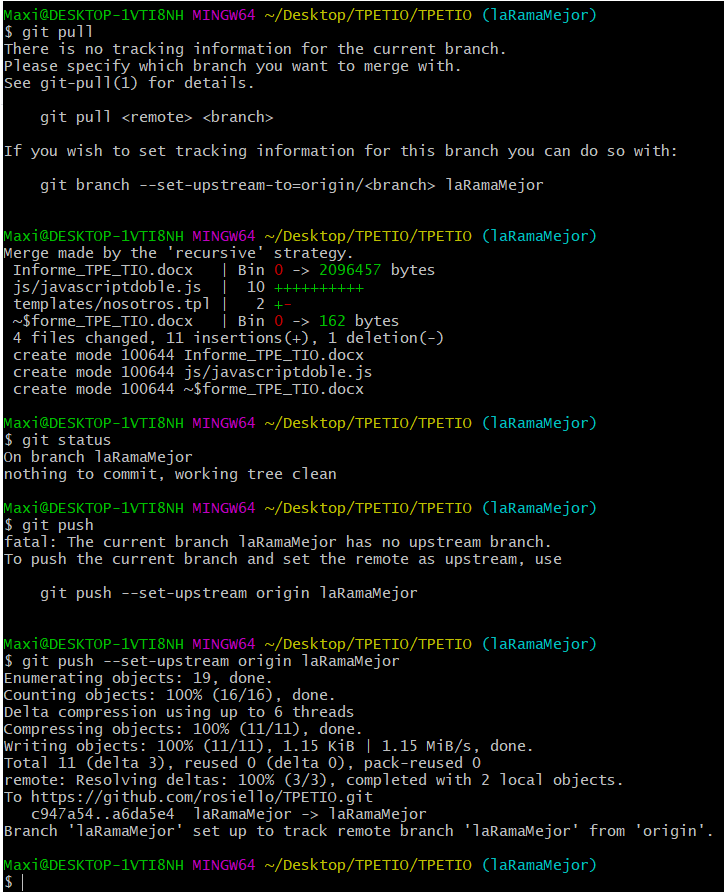
A continuación, las ramas creadas:

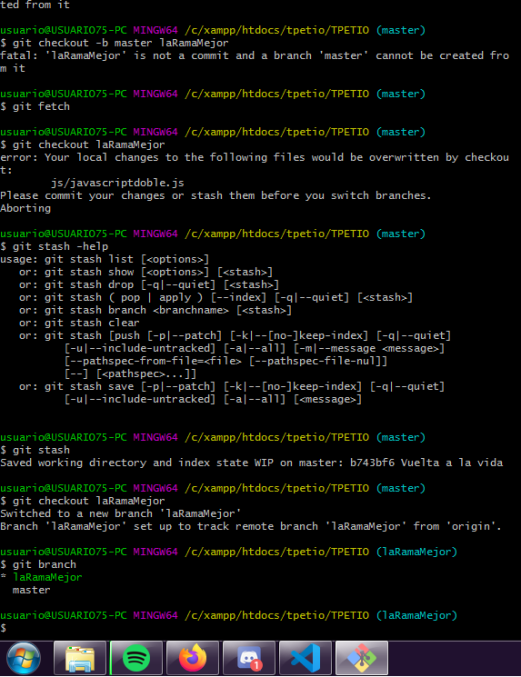


También se creó una rama paralela remota a la masterpara que todos los integrantes puedan hacer pruebas sin modificar ni tocarla.

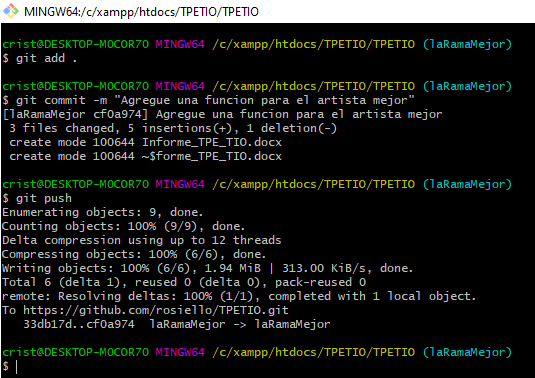


. Como se puede apreciar en la imagen anterior, para realizar ese trabajo un integrante del grupo utilizo el comando **“git fetch”** que les permitió obtener ramas y / o etiquetas (colectivamente, "referencias") de uno o más repositorios. Sin especificar una rama, se llamó a “laRamaMejor” y comenzaron a trabajar en ella para realizar pruebas en JavaScript con efectos de .css, así el equipo de a poco comenzó a trabajar más profesionalmente y dejaron de ser un desorden total.

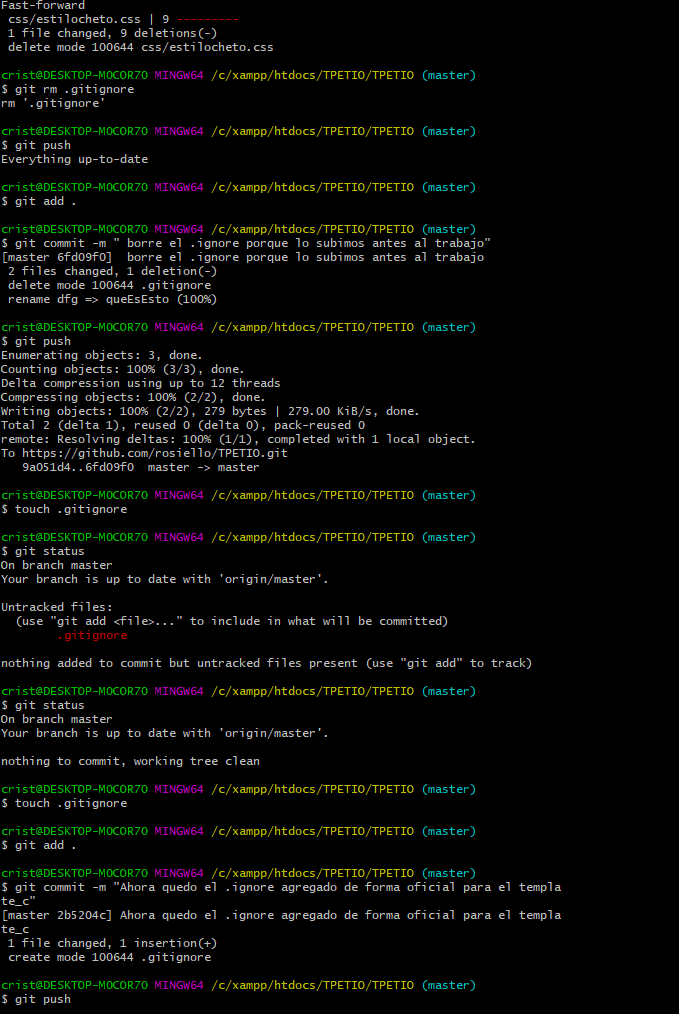




Luego de bajar la rama, todos participaron de las pruebas en esa rama alternativa modificando los archivos .css y .js.



Una de las últimas modificaciones que hizo fue agregar el .gitignore que en realidad se había agregado de forma manual. Para ellos los comandos son simples, en consola se escribe **“touch .gitignore”** y dentro se colocó el nombre de la carpeta temporal de PHP “template\_c” para que no se tengan en cuenta sus modificaciones.



HASTA ACA LLEGUEE…………………

FALTA SEGUIR CON LOS CAMBIOS QUE HICIMOS YO DIRIA LAS RAMAS Y LAS INCORPORACIONES DE ESAS RAMAS

----------------------ESTO DEJENLO PARA EL FINAL:--------------------------------------------

En ese momento fue justo donde todo comenzó a desmoronarse...

Una vez que cada miembro del grupo tuvo el PODER de modificar el trabajo realizado por PACO con gran esfuerzo, comenzó el problema:

***CONCLUSIÓN:***

