## Git es un sistema controlador de versionamiento local en nuestro pc que tiene integracion con muchas plataformas una de estas es Github que es una plataforma de desarrollo colaborativo de software, propiedad de Microsoft, para alojar proyectos. La version Gratuita de GitHub nos da:

### Repositorios ilimitados

### Crear páginas HTML, CSS y JS ilimitadas

### Push, pull y clones ilimitados

### Sistema de issues, wikis, estadisiticas.

### Crear Organizaciones ilimitadas

### Participacion Gratuita en proyectos privados

## Git no maneja controles de accesos a los repositorios para eso tenemos los servicios de terceros como GitHub y BitBucket o Gitosis que es manejado por nosotros mismos.

# Inicio Git

## Cuando colocamos -- (doble guion medio) significa que luego viene una palabra completa

## Cuando colocamos - (un solo guion medio) significa que luego viene una abreviatura

Ej:

git --version

git commit -am (abreviatura de “add message”)

## Obtenemos la documentacion del comando

git help commit

## Configurar Usuario

git config --global user.name “Pablo Sanabria”

Si no da error o feedback está Ok

git config --global user.email “pablosanabria\_1986@hotmail.com”

para confirmar podemos ejecutar:

git config --global -e

## Inicializar Repositorio

git init “ruta de la carpeta local”

## Informacion de la rama donde estamos posicionados

git status

A screen shot of a computer

Description automatically generated

Tambien existen algunos comandos que nos brindan info mas resumida:

git status --short

## Preparar archivos en el Stage para el commit

git add nombredelarchivo nombredelarchivo (separado con espacio se puede agregar mas de un archivo)

git add . (Prepara todos los archivos, no incluye cambios en archivos que se han eliminado)

git add -A (Prepara todos los archivos, incluye cambios en archivos que se han eliminado)

### Comodines

git add \*.html (agrega todoslos html)

git add js/\*.js (agrega todos los .js dentro de la carpeta determinada)

git add css/ (agrea todo lo que esté dentro de la carpeta)

git no da seguimiento a directorios/carpetas vacios. Es común que esta exista una carpeta en los proyectos para los archivos que suben los usuarios, y que si no existe esa carpeta de un error y reviente la aplicación. Es por eso, que se tiene la convención de crear dentro de la carpeta un archivo llamado ".gitkeep" para que git sí haga tracking de la carpeta y podamos añadirla al repositorio, esta sería su funcionalidad.

### Add y commit Juntos

git commit -am “comentario del commit”

## Quitar Achivos del Stage

git reset nombredelarchivo

## Realizamos el Commit que es como tomar una “foto” de los archivos al momento para tener un respaldo

git commit -m “nombre significativo corto del commit”

## Reconstruir archivos del proyecto a como estaban en el ultimo Commit

git checkout -- . (todos los archivos modificados)

git checkout -- nombredelarchivo (solo el especificado)

git checkout hashdelcommit nombredelarchivo (solo el archivo especificado del commit especificado)

## Listar rama sobre la cual estamos trabajando

git branch

## Cambiar nombre de una rama

git branch -m master main

git config --global init.defaultBranch main (para que siempre que hagamos el init no se cree mas la rama master y sea main)

## Buena Practica: Agregar comentarios en ingles

## Combinar git Add y git commit

Git commit -am “comentario” (solo funciona si el archivo que estamos queriendo agregr ya fue agregado al menos una vez)

## Crear Alias de comandos para que sean mas cortos de escribir

git config --global alias.s “status --short” (cuando ejecutemos git s, va a ejecutar git status --short)

git config --global alias.s “status --short --branch” o “-sb” (cuando ejecutemos git s, va a ejecutar git status --short --branch)

git config --global alias.lg "log --graph --abbrev-commit --decorate --format=format:'%C(bold blue)%h%C(reset) - %C(bold green)(%ar)%C(reset) %C(white)%s%C(reset) %C(dim white)- %an%C(reset)%C(bold yellow)%d%C(reset)' --all"

(cuando ejecutemos git lg ejecuta este comando entero)

## Log y Log resumido

git log

git log --oneline

## Ver diferencias en un archivo

git diff (que no esten en el stage)

git diff --stage (que esten en el stage)

## Modificar el comentario/mensaje de el ultimo commit

git commit --amend -m “nuevo comentario”

## Hacer que cambios(commits) posteriores sean parte de un commit anterior

Solo se recomienda con pocos commits intemedios y sin push mediante.

soft(mantiene los cambios) , mixed(no es destructivo pero saca todo del stage y los cambios quedan listos para volver a añadir si hacemos un reset sin poner nada luedo se ejecuta este por defecto), hard (Es destructivo, deja el repositorio tal como estaba en ese punto)

git reset --soft HEAD^(ultimo commit)

git reset --soft HEAD^2( anteultimo commit y asi… git reset --soft HEAD^3 )

#### Otra Manera:

git reset --soft HASHDELCOMMIT

### Log de las referencias de todo lo que sucedió en orden cronologico

git reflog

esto nos mostrará todos los hashes de todos los commits y entonces podemos agarrar el hash de un commit “posterior” al que queremos volver y ejecutar git reset --soft HASHDELCOMMIT para restaurar todo a ese commit. Deberian aparecer los cambios/modificaciones incluyendo archivos borrados.

## Cambiar nombre y eliminar archivos agregandolo al stage

git mv nombreopathdearchivoorigen nombreopathdearchivodestino (cambiar nombre-move)

git rm nombredelarchivo (eliminar-remove)

Hacer commit para finalizar

SI MODICAMOS EL NOMBRE DE UN ARCHIVO POR FUERA DE GIT POR EJEMPLO DESDE EL EXPLORADOR DE ARCHIVOS DE VSCODE TENER CUIDADO Y REVISAR QUE SIEMPRE APAREZCA COMO RENAMED ”R”. LUEGO DE MODIFICARLO HACER UN git add . Y LUEGO EL COMMIT

SI ELIMINAMOS UN ARCHIVO POR FUERA DE GIT ESTE NO SE AGREGA AL STAGE POR LOQUE TAMBIEN TENEMOS QUE HACER UN ADD Y LUEGO EL COMMIT

## Ignorar archivos que no deseamos darle seguimiento, por ejemplo archivos de compilacion

Crear un archivo llamado .gitignore en la raiz del proyecto y escribir dentro de este las rutas y nombres de las carpetas y archivos que queremos que se ignoren por ejemplo:

dist/

node\_modules/

\*.log

## Simbologia de un archivo sin seguimiento en el repositorio

?? nombredearchivo (por ejemplo: ?? Historial.md )

# Ramas, uniones, conflictos y tags

Una rama es una linea de tiempo de commits.

Un Merge es cuando queremos unir una rama con otra. Hay 3 tipos:

Fast-foward: se dispara cuando Git detecta que no hay ningun cambio en la rama principal y los cambios pueden ser integrados de forma transparente, cada uno de los commits formará parte de la rama principal como si nunca hubiesen estado separados, esta caracteristica se puede desactivar.

Uniones Automaticas: Git detecta que en la rama principal hubo algun cambio que la rama secundaria desconoce pero no hay conflictos

Union Manual: existen conflictos y Git no puede resolverlos de forma automatica y nos pide que lo hagamos nosotros con un Merge Commit.

Para hacer un Merge es importante posicionarse en la rama de origen por ejemplo master si queremos traer a master cambios de otra rama

## Crear una rama

git branch nombredelarama

git checkout -b nombredelarama (crea la nueva rama y nos mueve a la misma)

## Listar ramas

git branch

## Moverse a una rama

git checkout nombredelarama

## Merge

git merge nombreramadestino

## Borrar Rama

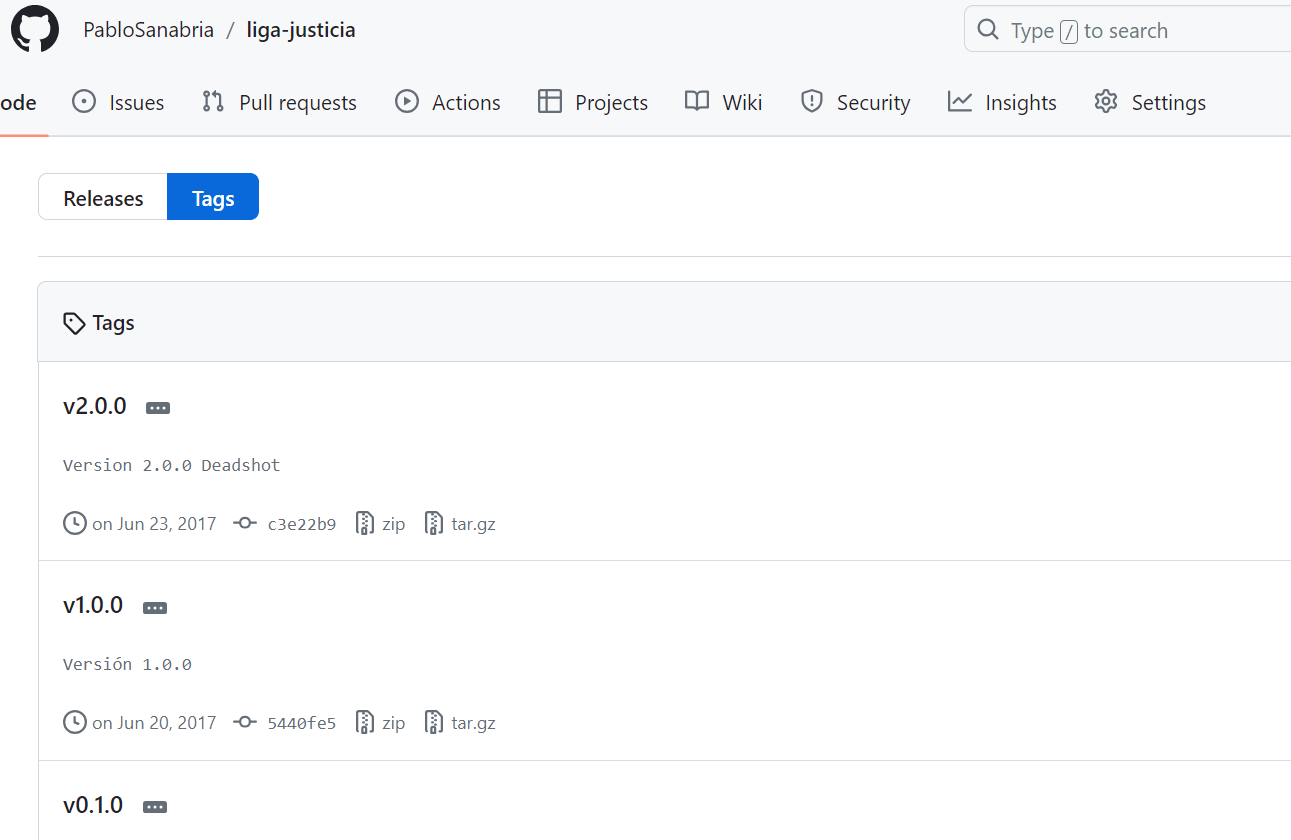
git branch -d nombredelarama

git branch -d -f nombredelarama o git branch -D nombredelarama (elimina la rama de manera forzada, los cambios no fusionados se perderán)

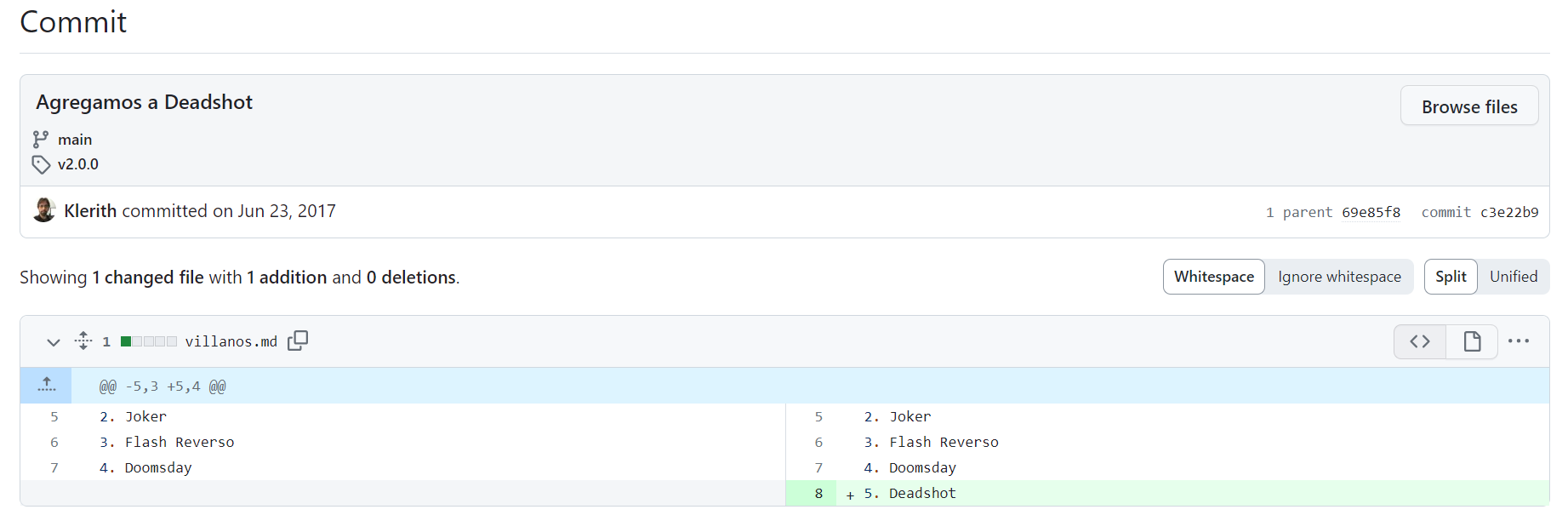
## Tags(etiquetas)

Son una referencia a un commit y todo el estado de un proyecto en un punto especifico. Normalmente son usados para marcar versiones o releases de un aplicativo. Por ejemplo podemos Explicar con un Tag que tal commit (“Creacion de ejecutable”) marca la Version 1.0.0 de nuestro programa.

Podemos ver los tags en GitHub y descargar los .zip para ver como estaba el codigo en ese momento



Tambien podemos ver los cambios que hubo en el codigo directamente en la web haciendo click en el Hash(c3e22b9)



Si presionamos el simbolo de + podemos añadir comentarios para esa seccion de codigo (add single comment)

### Crear tag

git tag nombredelTag

git tag -a v1.0.0 -m “Version 1.0.0 lista” (crea el tag con un comentario/mensaje)

git tag -a v0.1.0 hashdelCommit -m “Version 0.1.0 alpha de nuestra app”

### Listar tags

git tag

### Ver informacion de un tag

git show nombredelTag

### Eliminar tag

git tag -d nombredelTag

## Obtener todas las ramas desde la nube con su contenido

git pull --all

### Listar todas las ramas obtenidas

git branch --all o -a

### Limpiar ramas locales que ya no son necesarias y no estan en la nube

git push origin :nombredelarama (Elimina local y en la nube)

*git remote prune origin* (revisa las ramas de la nube que no existen y actualiza nuestras referencias locales)

### Recuperar rama eliminada localmente y desde la nube

Si tenemos un tag creado podemos hacer un git checkout nombredeltag. Luego volvemos a crear la rama congit checkout -b nombredelarama*.* Luego para subirla a la nube git push y copiamos y pegamos lo que nos salte en el error o el comando correcto es: git push --set-upstream origin nombredelanuevarama

Tambien lo podemos hacer desde Github yendo al tag, seleccionarlo y creando la rama

A screenshot of a computer

Description automatically generated

#### Si no tenemos Tags de esa rama

Hacemos git reflog para ver los hashes

# Stash

El stash es una boveda segura en donde podemos mover todos los cambios que tengamos en nuestra rama desde el ultimo commit, inclusive cambios de archivos que git todavia no le da seguimiento. Al volver a obtener el Stash podemos tener conflictos o no. No se recomiendo hacer muchos Stash

Se utiliza cuando se necesintan subir cambios ya listos pero actualmente tambien estamos trabajando en otros cambios, entonces se mueven los ultimos al Stash y se suben los viejos. Lo ideal es tener en master siempre los ultimos cambios listos.

una vez que obtenemos el Stash no se muestra al hacer un git lg

### Crear Stash

git stash

git stash save “comentario” (deja un comentario en el stash)

### Listar Stash

git stash list

git stash list --stat (admeas muestra informacion de cada stash)

### Obtener ultimo Stash y eliminarlo

git stash pop

Borrar todos los Stash

git stash clear

### Obtener Stash especifico

git stash apply iddelstash(ej: stash@{0})

### Borrar Stash Especifico

git stash drop iddelstash(ej: stash@{1})

git stash drop(borra el ultimo)

### Ver informacion de un Stash

git stash show iddelstash

# Rebase

El Rebase permite unir y separar commits, hacer un Squash (una fusion), se recomienda hacer si todavia los cambios no se han subido.

Por ejemplo, cuando tenemos que actualizar una rama desde master. Al hacer git rebase master mueve los commits de la rama a un area temporal, luego mueve el puntero de la rama al final del ultimo commit de master y luego vuelve colocar los commits de la rama. El Rebase Interactivo por ejemplo git rebase -i HEAD~3 o hashdelcommit (virgulilla media es Alt-126) mueve al area temporal los ultimos 3 commits. Sirve para ordenar, corregir mensajes, unir y separar commits.

### Hacer Rebase

git rebase “nombredelarama”(siempre se recomienda a master)

### Squash - Unificar commits(se recomienda que sean commits locales)

git rebase -i HEAD~4(nos une 4 commits antes del head podemos poner el hash de un commit tambien)

Esto nos lista ultimos 4 commits y nos muestra las opciones de edicion de commits. Presionamos la tecla a para poder editar los commits y en el ultimo commit que queremos fusionar ponemos adelante la palabra squash o s

Para salir y guardar los cambios presionamos Esc luego :wq y Enter

Esto nos muestra una pantalla de edicion de mensajes de los commits para salir de esta nuevamente Esc luego :wq! y Enter

### Reword - Cambiar un Comentario de un commit

git rebase -i HEAD~4

En el modo de edicion: presionamos la tecla a para poder modificar. Colocamos la palabra reword o r delante de los commits que queramos modificar y luego Esc :wr!

Entramos en el modo del rebase interactivo. volvemos a presionar la tecla a. Podemos modificar el comentario y salimos con Esc :wr

lo mismo para el o los siguiente rewords que tengamos

### Rebase Edit- Ideal cuando se quiere separar un commit

git rebase -i HEAD~4

En el modo de edicion: presionamos la tecla a para poder modificar. . Colocamos la palabra edit o e delante del commit que queramos modificar y luego Esc :wr!

Nos mostrará una pantalla de Rebase Manual en donde nos preguntará si queremos seguir haciendo correcciones con un *git commit --amend* o si deseamos contuniar con un *git rebase --continue*

*Entones ahora podemos hacer un reset para posicionarnos en antes del ultimo commit y poder luego dividirlo*

*git reset HEAD*^ (para que nos traiga el ultimo commit)

git add nombredelarchivo le damos Enter

git commit -m “comentario”

Lo mismo para el siguiente commit

y para finalizar hacemos el *git rebase --continue*

## QUITAR ADVERTENCIA: LF will be replaced by CRLF…

git config core.autocrlf true

# GITHUB

Guardar su contraseña de GitHub en la máquina **WINDOWS**

[Guardar usuario y contraseña de GitHub](https://help.github.com/articles/caching-your-github-password-in-git/#platform-windows)

Luego de Crear un repositorio en la web. Github nos muestra los comandos necesarios para hacer nuestro primer Push ya sea que no tengamos ningun fuente o si ya tenemos.

### Agregar repositorio remoto

git remote add origin urldelrepositorioremoto

el add significa que es un nuevo remote, el origin es el nombre de nuestro repositorio por lo tanto si agregamos mas repositorios le podemos poner otros nombres, el standard es que se llame origin para referirnos al Origen. Podemos tener varios remotes en un mismo repositorio

### Listar todos los remotos que tenemos en el repositorio

git remote -v

### Push

git push -u origin master

*-u* nos ayuda a que la proxima vez que queramos hacer un push no sea necesario especificar la rama (git push), por ejemplo si queremos que master sea la rama por defecto.

*origin* es el nombre del repositorio.

*master* es la rama que que queremos subir.

### Push Tags

git push --tags

### Release Tags

Hacerlos es fundamental para saber que tal version de la aplicación es la indicada para descargar o publicar, sirve como respaldo

A screenshot of a computer

Description automatically generated

### Ver Path de repositorio

git remote -v

### Clonar repositorio

git clone urldelrepo

### Markdown

<http://www.markdowntutorial.com/>

<https://www.webpagefx.com/tools/emoji-cheat-sheet/>

### Pull Request

Cuando queremos unir o integrar una rama que se desprendio de la rama principal lo podemos hacer mediante un PR, nos sirve para que alguine pueda hacer una revision y analisis previo al merge, por ejemplo standares de codificacion o funcionalidad

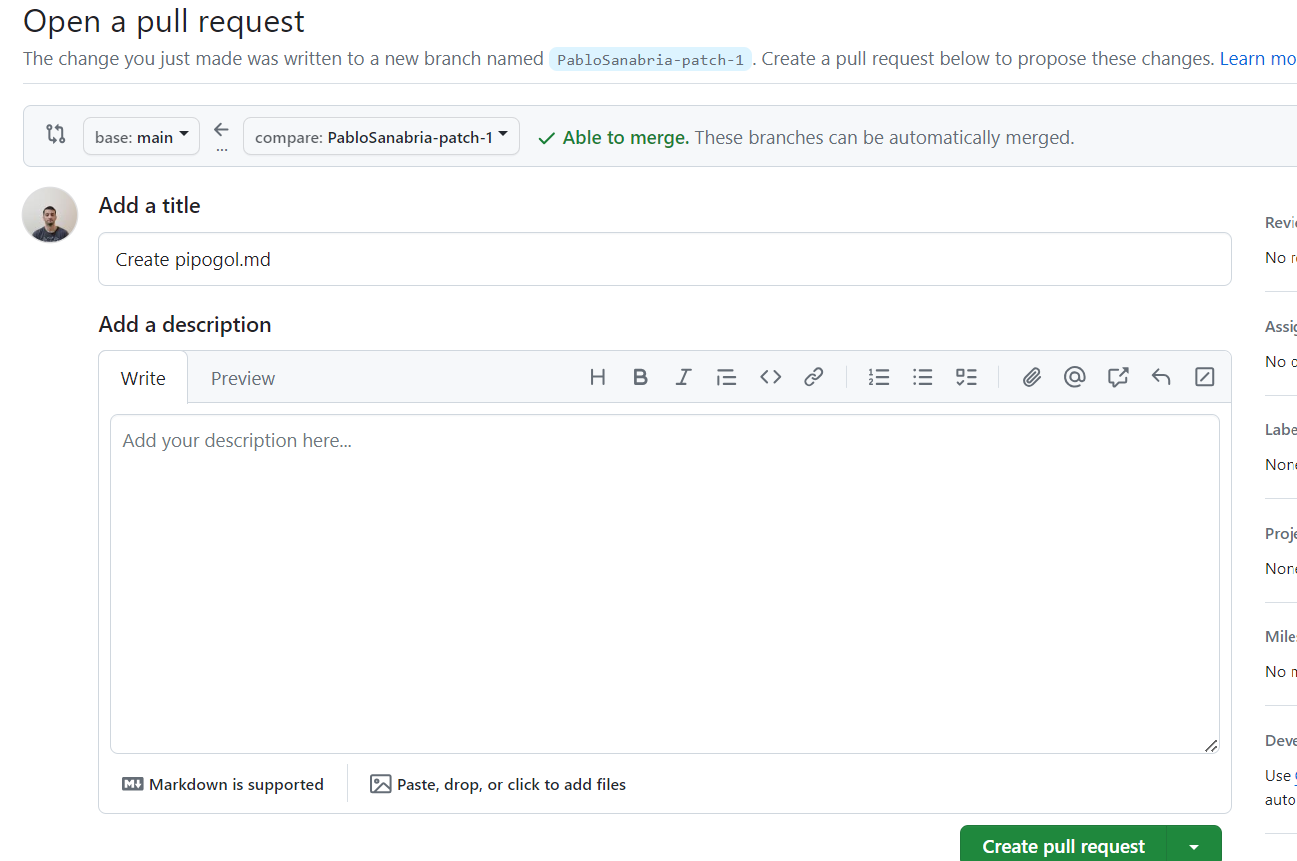
#### Commit con PR

Creamos un nuevo archivo, luego presionamos el boton “Commit Changes…”, en el popup agregamos un commit message y una descripcion si queremos, seleccionamos “Create new branch for…” y presionamos “Propose Changes”

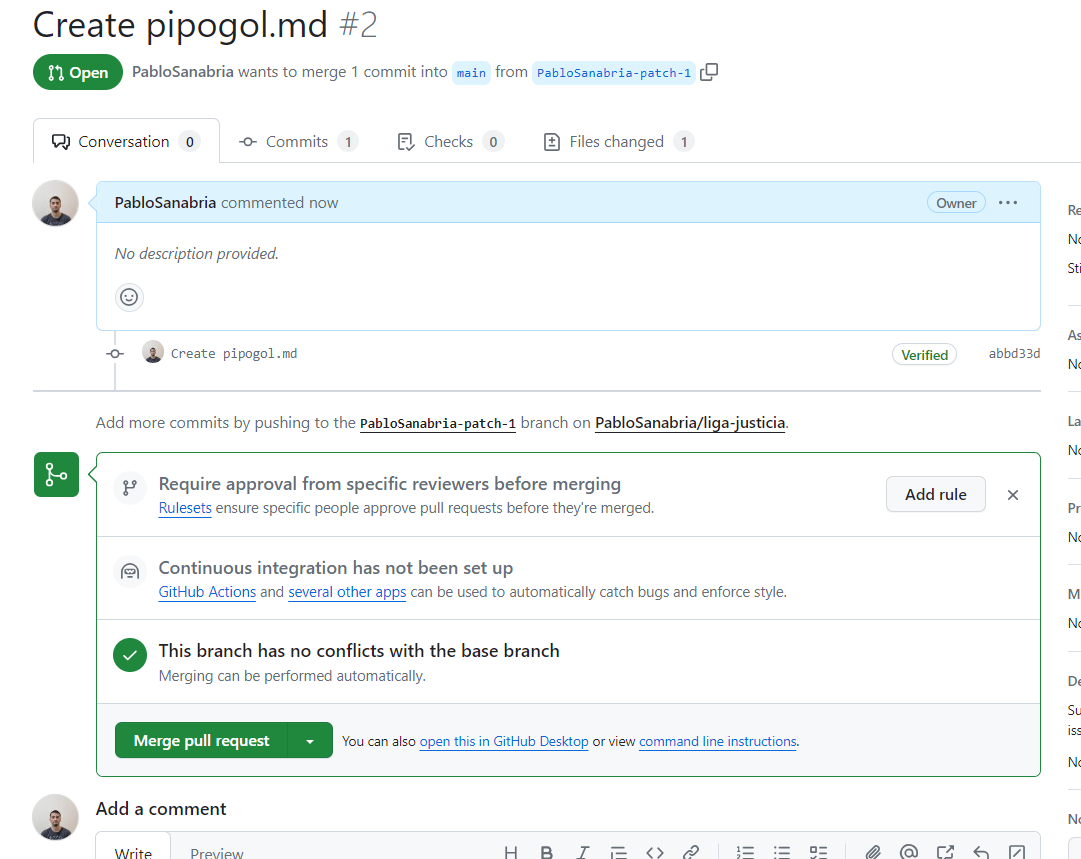
A screenshot of a computer

Description automatically generated

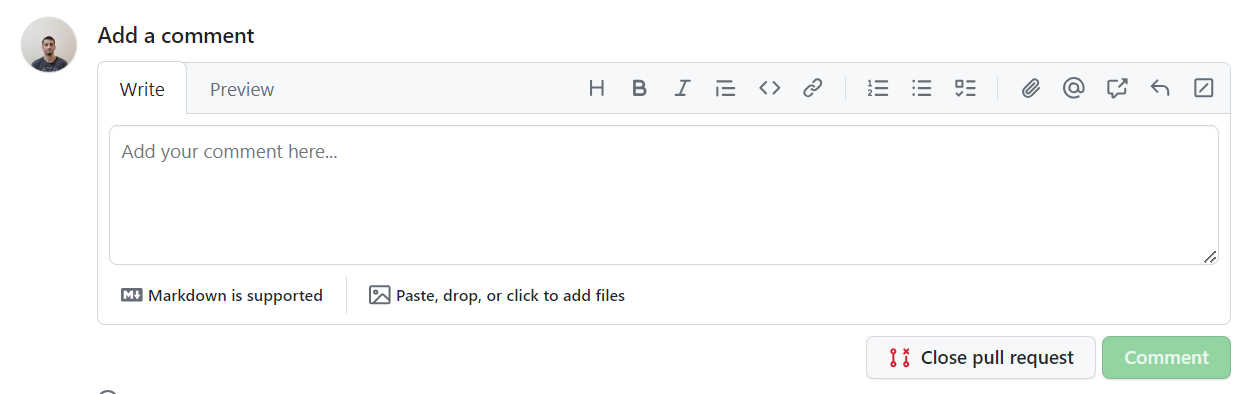
Nos va a mostrar una pantalla donde aparece el PR abierto y las ramas a fusionar



En Write podemos colocar algun comentario que queramos que los demas vean para este PR y presionamos “Create pull request”

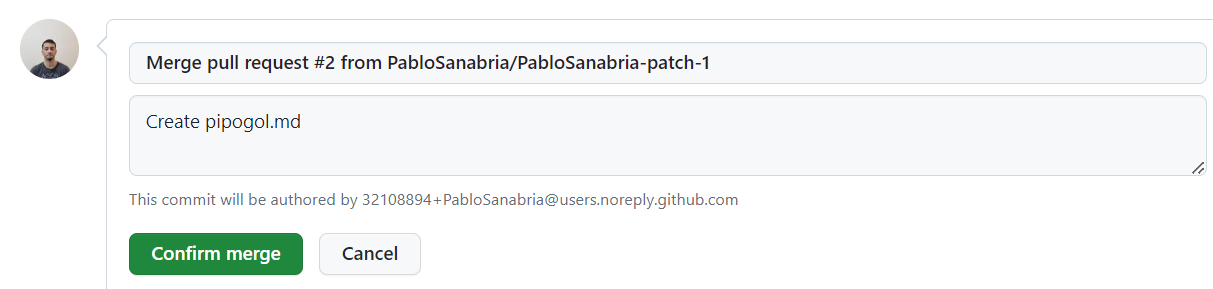


Aquí las otras personas pueden agregar sur comentarios o cerrar el PR



Tenemos 2 maneras mas usadas Create merge commit (se crea un nuevo commit independiente para este PR) y Squash and merge(se fusiona sin commit)

##### Create merge commit



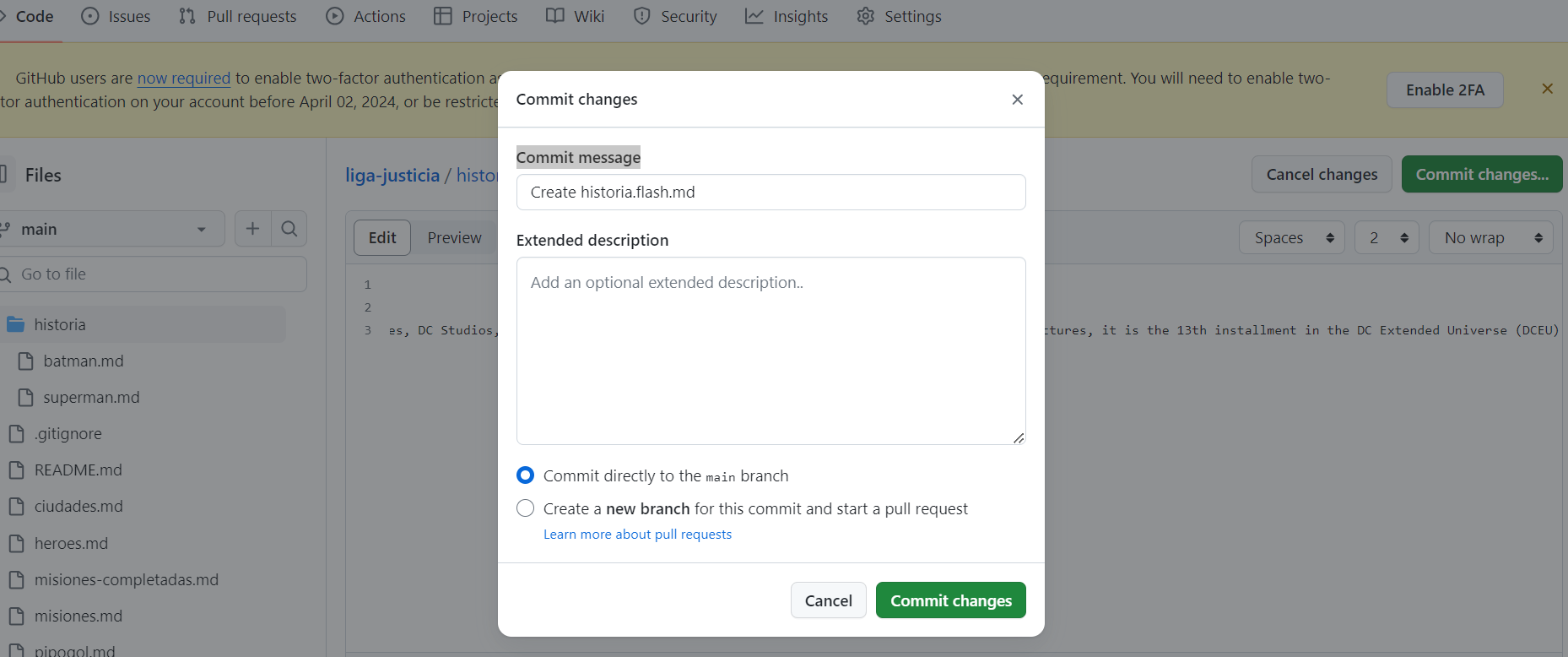
Confirmamos y luego podemos eliminar la rama

A screenshot of a computer

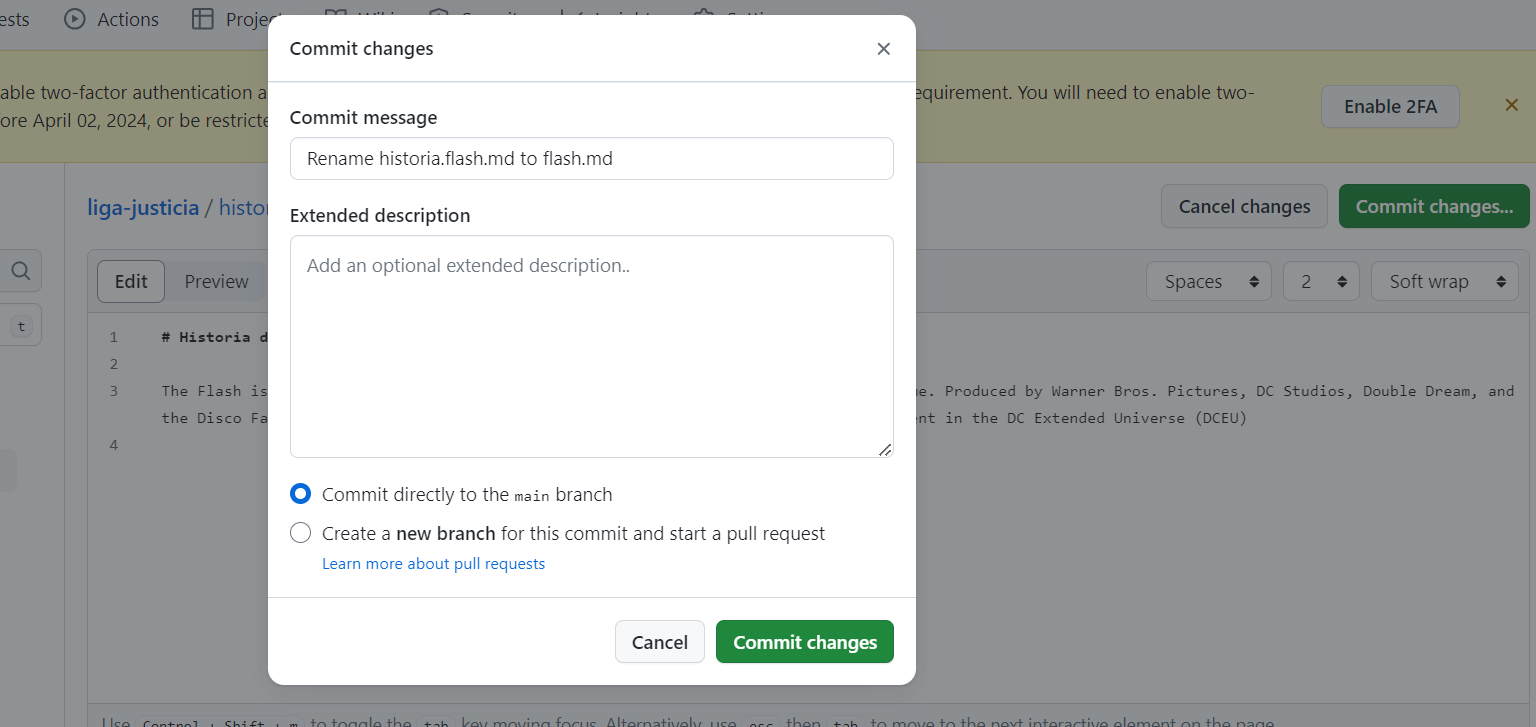
Description automatically generated

### Git Fetch

Creamos nuevo file



Renombramos el archivo:

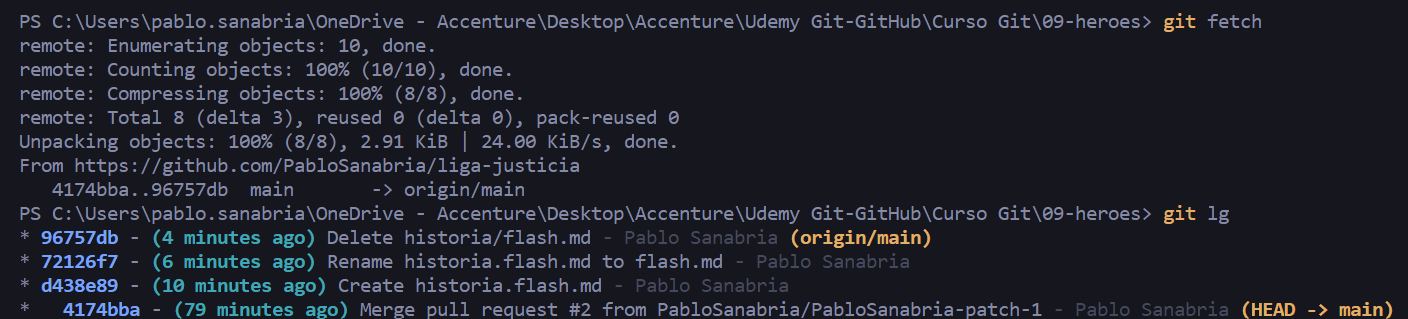


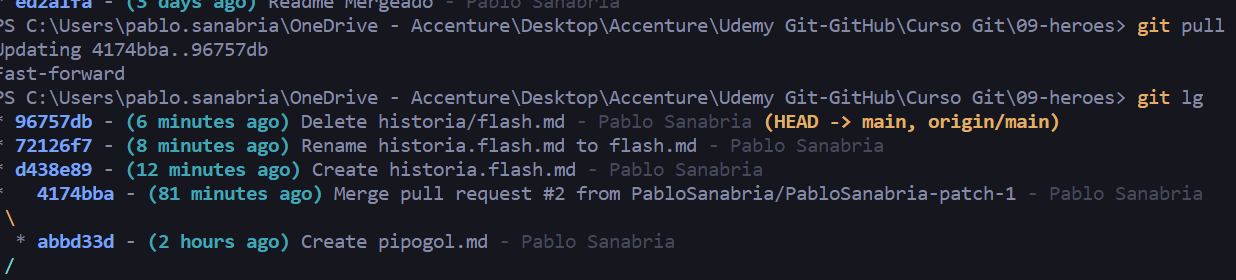
Borramos el archivo:

A screenshot of a computer

Description automatically generated

Actualizar el repositorio local con los ultimos cambios sin hacer merge nuestro HEAD local sigue apuntando al ultimo commit local

Luego con un git pull apuntamos el HEAD al ultimo commit del repositorio de github



### Flujo Github

[Understanding the GitHub Pull Request Workflow (mergify.com)](https://blog.mergify.com/understanding-the-github-pull-request-workflow/)

### Fork, Clone Y Colaboraciones a repositorios de terceros

Si alguien es agregado a un repositorio como “Colaborador” puede manipular el mismo como si fuera propietario es decir hacer clone, pull, push. Si no sos colaborador no se puede hacer un push pero si queremos colaborar podemos hacerlo con un Fork(seria como hacer una rama adicional en donde el tercero no sabe que es lo que hacemos pero puede ser unida al repo original), que es tomar el repo original y clonarlo a un lugar donde tengamos acceso total en github y luego de hacer nuestras modificaciones hacemos un Pull Request para que el dueño realice la revision.

Luego de querer crear un nuevo archivo en un rapo que no nos pertenece aparece lo siguiente:

A screenshot of a computer

Description automatically generated

Para hacerlo de manera mas apropiada podemos tocar el boton Fork:

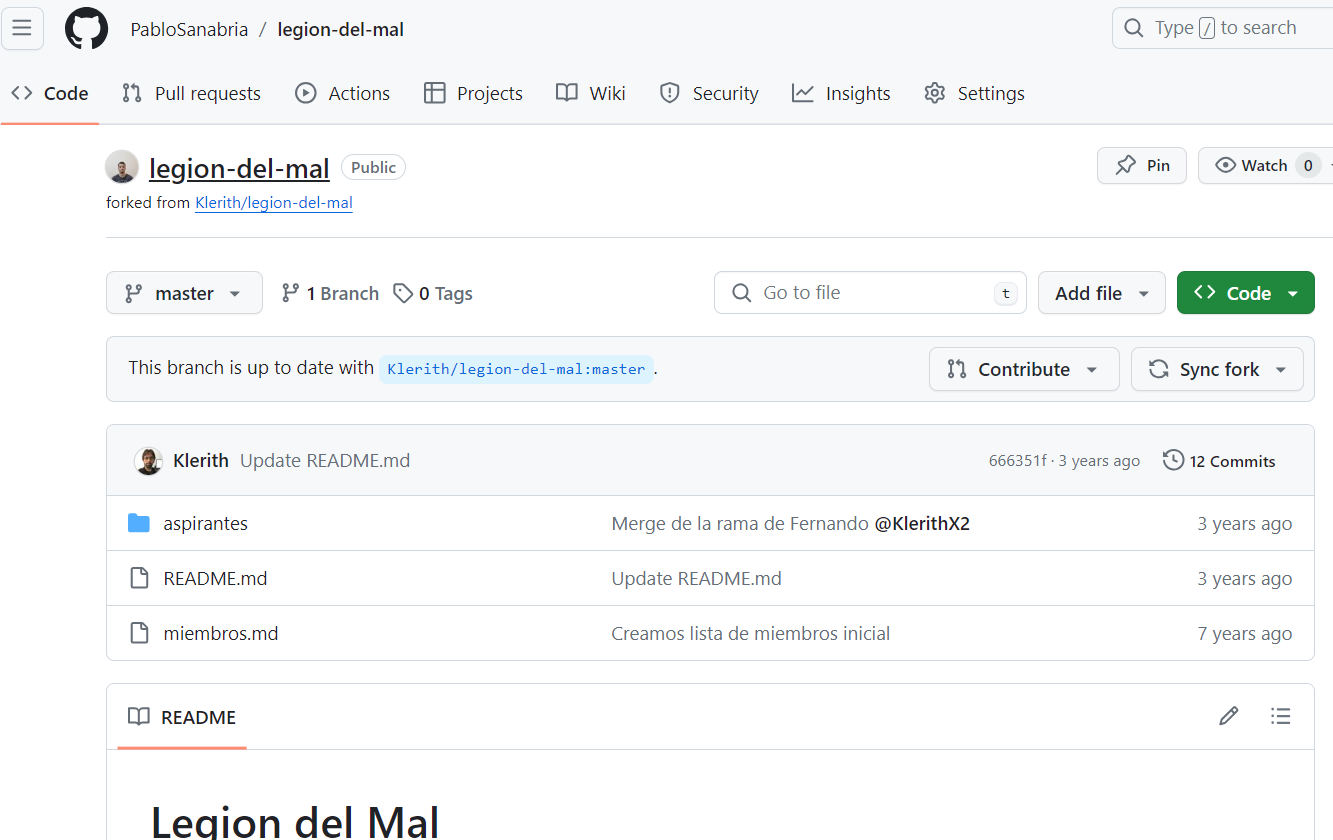
A screenshot of a computer

Description automatically generated

Nos da la opcion de modificar algunas cosasA screenshot of a computer

Description automatically generated

Presionamos Create fork y nos crea la copia en nuestro area



Para contruibuir vamos al boton Contribute - Open pull request :

A screenshot of a chat

Description automatically generated

Nos va a mostrar los cambios y podemos creaer el PR:

A screenshot of a computer

Description automatically generated

Todos los cambios que subamos hasta que el PR sea aceptado se veran reflejados

Des el lado del autorizador para en el caso de que quiera rechazar o cancelar un PR, hay que ir a la pestaña Files Changed aparecerá un boton Review changes donde se puede hacer Comment, Approve o Request Changes(solicitar cambios) y obviamente hacer el Merge

#### Actualizacion de Fork desde el UPSTREAM

Debemos tener permisos de lectura sobre el repositorio origin, entonces copiamos la url del repo y en la consola colocamos:

git remote add upstream urldelrepoorigin (Upstream, repo del cual obtenemos informacion, de esta manera será el alias del repo original)

Podemos hacer un *git remote -v* para ver que esté todo bien y luego:

*git pull upstream master*

*luego commit y push*

### Feature branch

Es una rama en que un usuario desprende de la rama principal o master y en algun momento va a ser incuida nuevamente a la rama principal

#### Push de nueva Feature branch

git push y copiamos y pegamos lo que nos salte en el error o el comando correcto es:  
git push --set-upstream origin nombredelanuevarama

### Issues

Suelen ser utilizados para crear preguntas sobre errores, solicitar features y darle seguimiento a las caracteristicas que debe tener el proyecto.

Podemos setear diferentes templates en Settings-> Issues->Set Up Templates para que por defecto al crear un Issue se vean ciertas caracteristicas

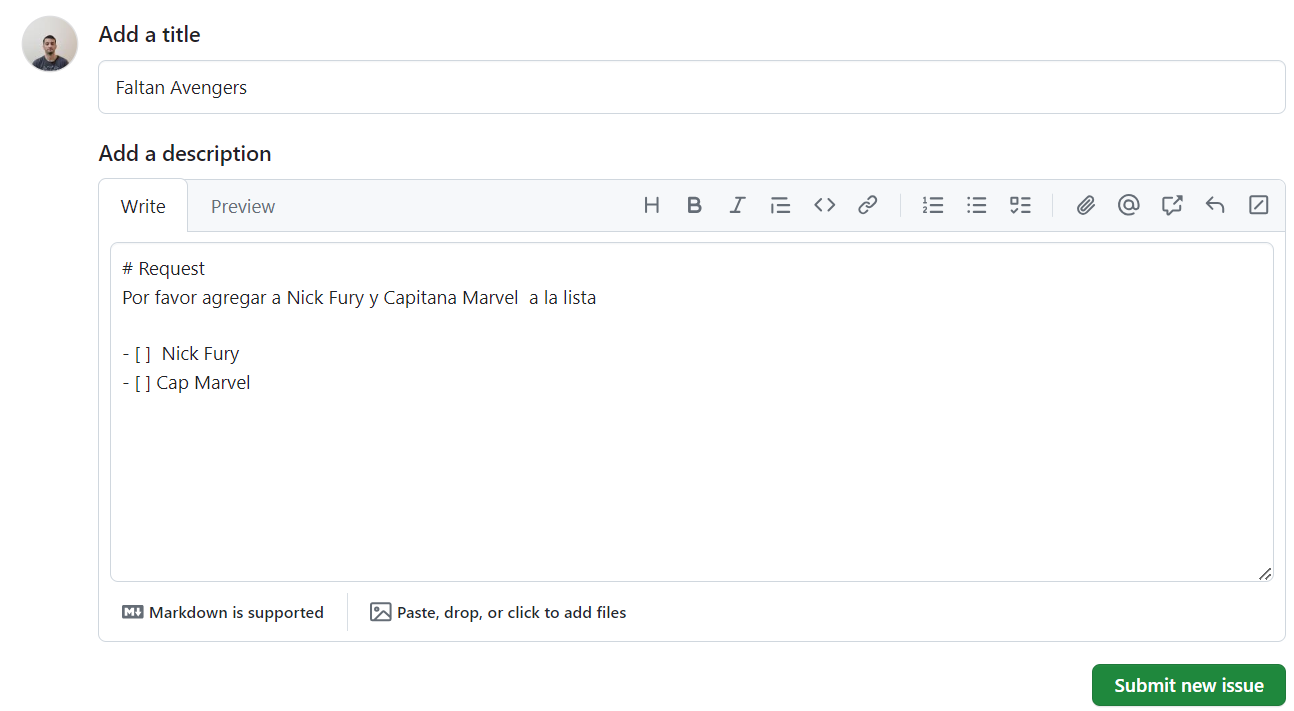
**A screenshot of a computer

Description automatically generated**

#### Crear Issue

A screenshot of a computer

Description automatically generated

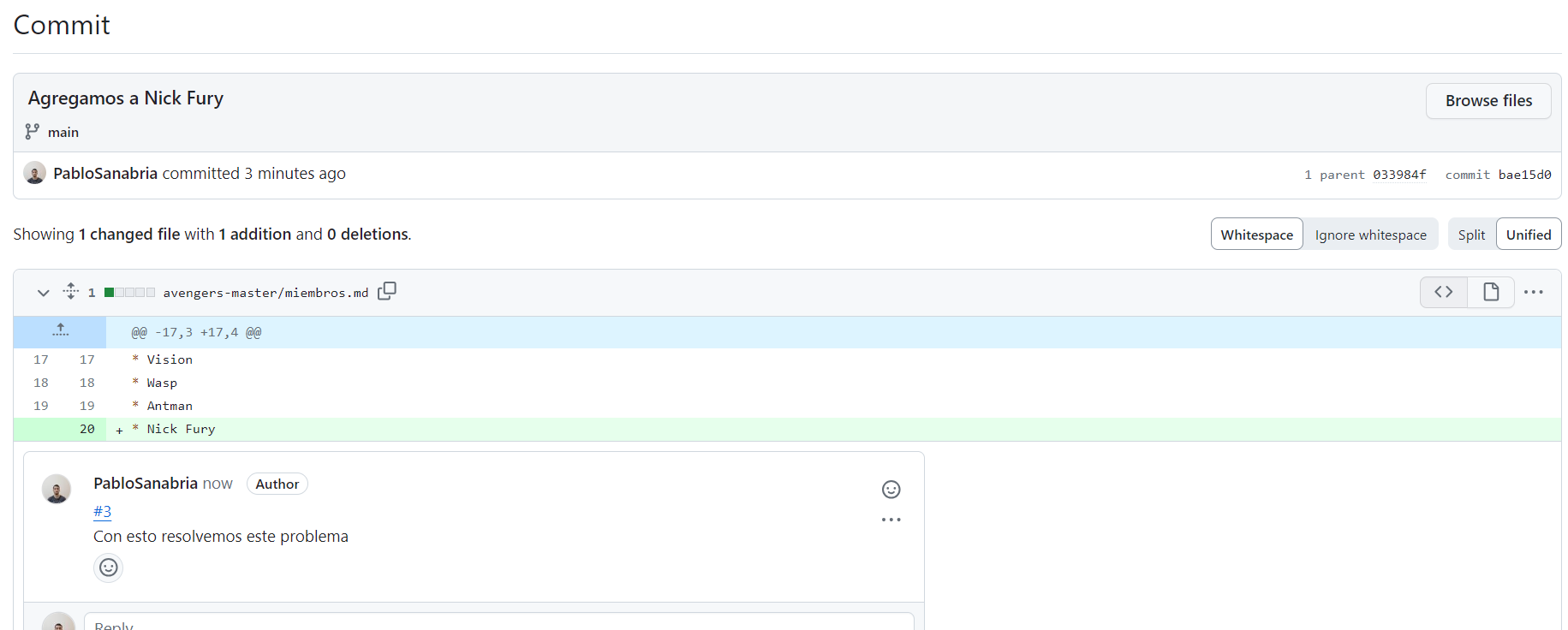


A screenshot of a computer

Description automatically generated

#### Cerrar Issue

Luego de hacer los cambios con sus respectivos commist y push podemos verlo en Github y comentar asociandolo con el Hashtag o tambien colocando el Hash del commit en un comentario dentro del Issue



A screenshot of a computer

Description automatically generated

Luego podemos cerrarlo y si queremos bloquer posteriores acciones

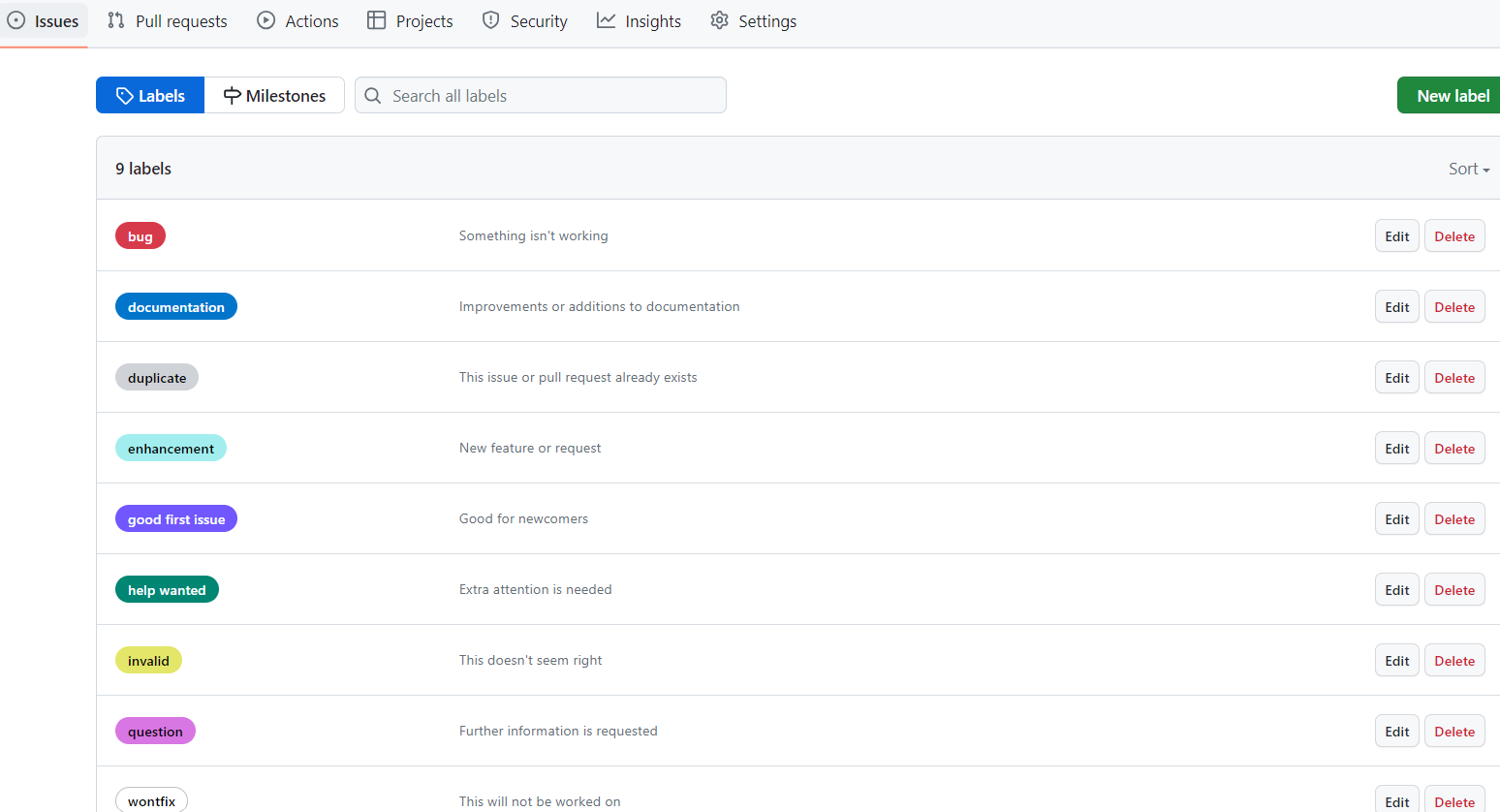
A screenshot of a computer

Description automatically generated

##### Desde el commit en git apuntando al Hashtag del Issue:

git commit -am “Fixes #5: Hecho, Capitan Marvel borrado”

#### Labels



A screenshot of a computer

Description automatically generated

#### Milestones

Es un agrupador de Issues de un proyecto

A screenshot of a computer

Description automatically generated

A screenshot of a computer

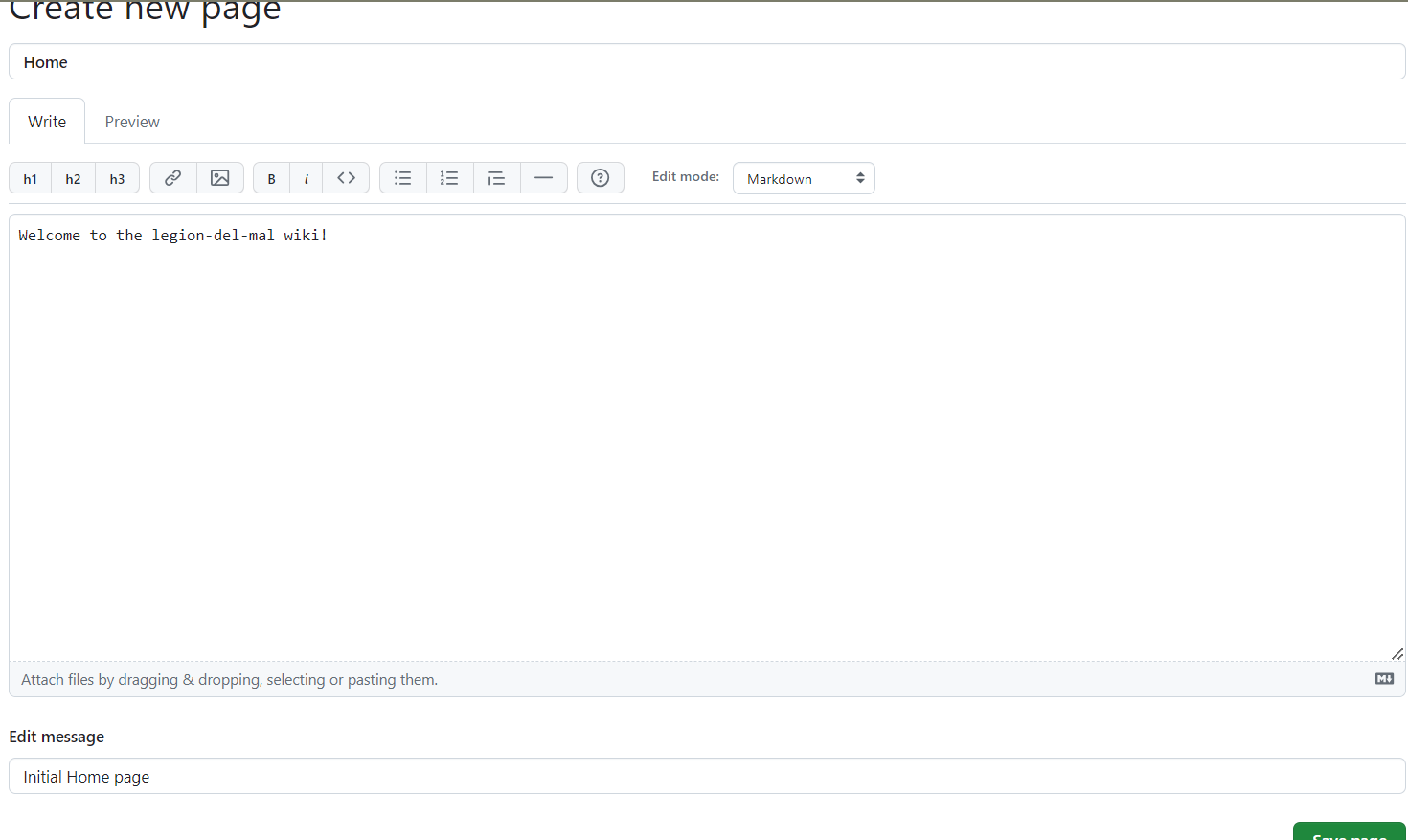
Description automatically generated

### Wiki

Para repositorios privados se requiere un pago para habilitar las Wikis desde las settings del repositorio. Para repositorios públicos, no hay inconveniente y están disponibles. Sirve para crear documentacion, manual de usuario, instalaciones, etc.

A screenshot of a computer

Description automatically generated



### Proyectos

Es un administrador de proyecto tipo JIRA que podemos customizar

A screenshot of a computer

Description automatically generated

Creamos un nuevo proyecto. Seleccionamos por ejemplo el template Board:

A screenshot of a computer

Description automatically generated

### Github Pages

Hosting gratuito para montar sitios webs en HTML, CSS y Javascript. Es ideal para colocar contenido estatico o sitios web dinamicos hechos en Javascript, React, Angular, Vue o cualquier lenguaje que no tenga codigo de servidor

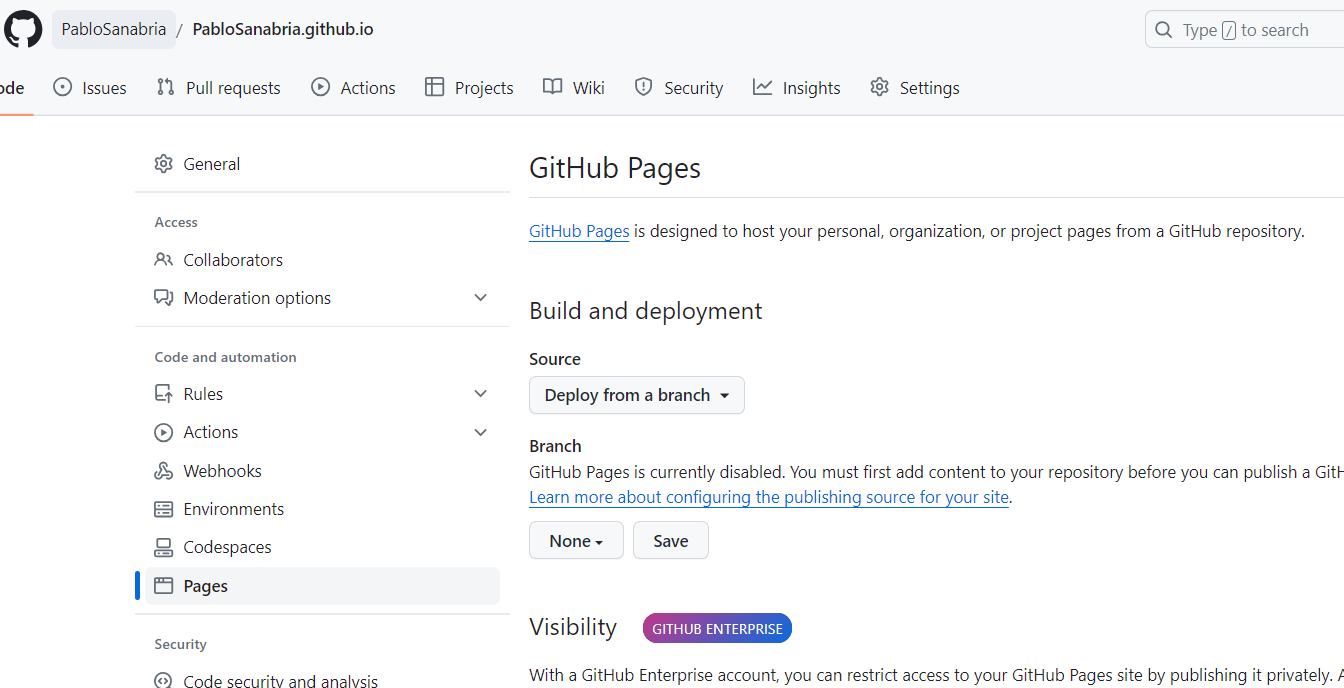
Hay 2 Tipos Para Usuario o para el Proyecto

#### Usuario:

A screenshot of a computer

Description automatically generated

Luego en Settings -> Pages



#### Proyecto:

Cramos una nueva rama que podemos llamar rama-web, creamos una nueva carpeta con nuestro sitio web html y lo mergeamos al main/master

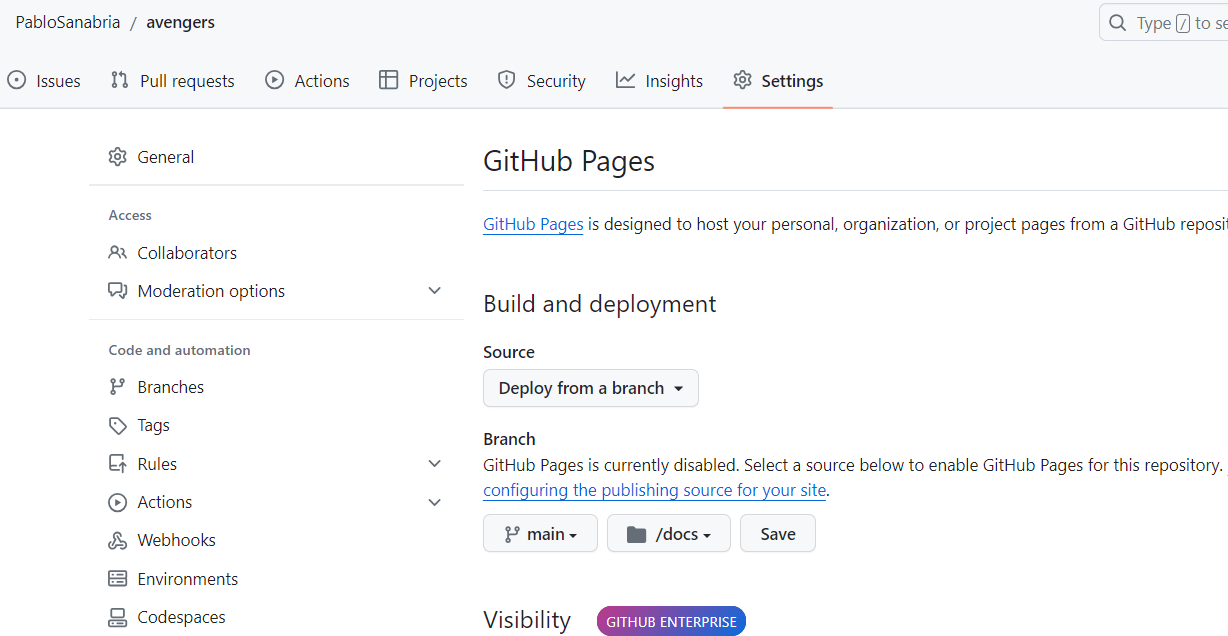
A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

En branch seleccionamos el branch y la carpeta que contiene la pagina y le damos Save



no funcionó al momento hubo un error en el deploy, me llegó un mail

### Organización

Es un grupo de personas con un objetivo en comun, ejemplo el area de desarrollo. Se crean para dar acceso a las personas a los repositorios y configurar permisos.

A screenshot of a computer

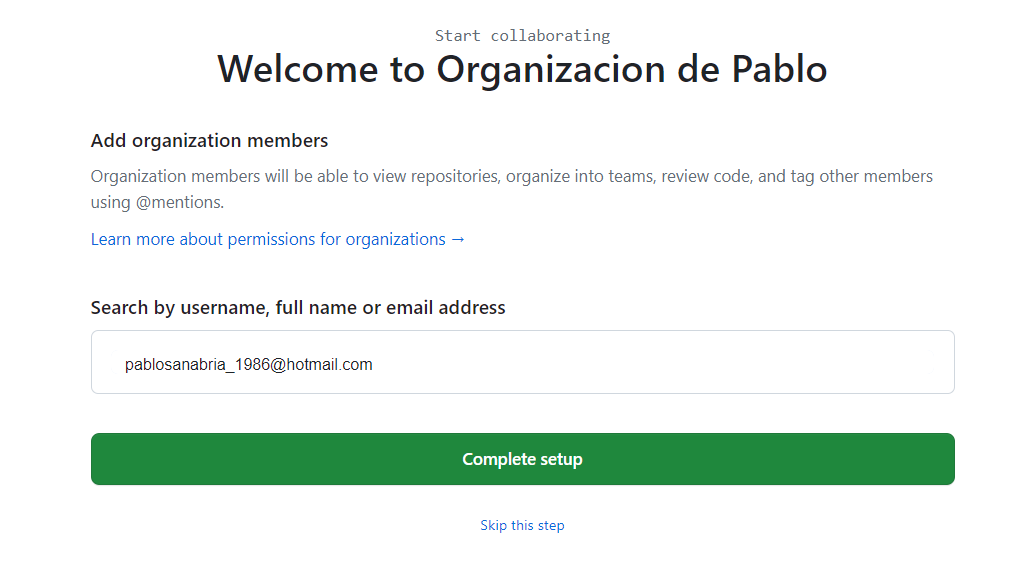
Description automatically generated

Nos da la opcion de sleccionar versiones pagas o la gratuita:

A screenshot of a computer

Description automatically generated

Añadimos miembros



A screenshot of a computer

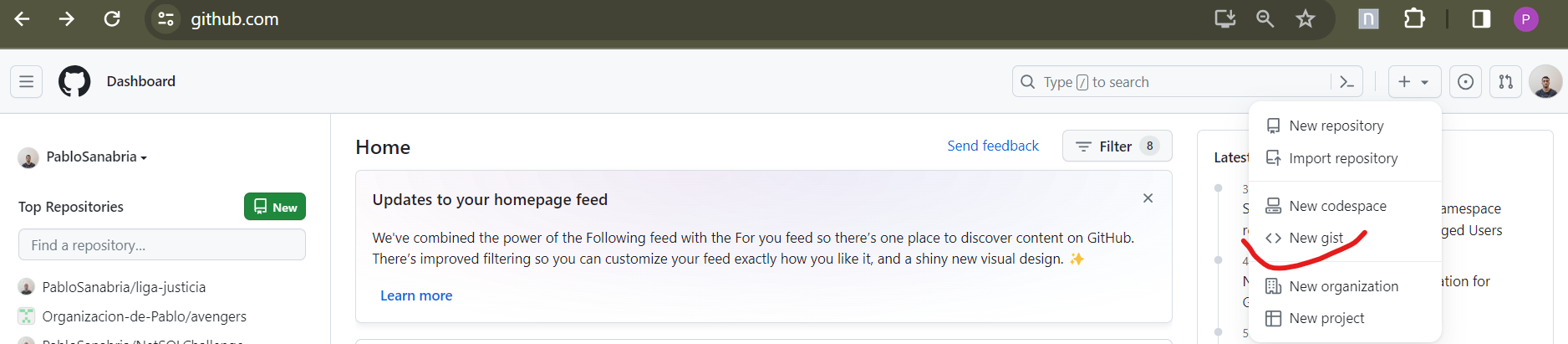
Description automatically generated

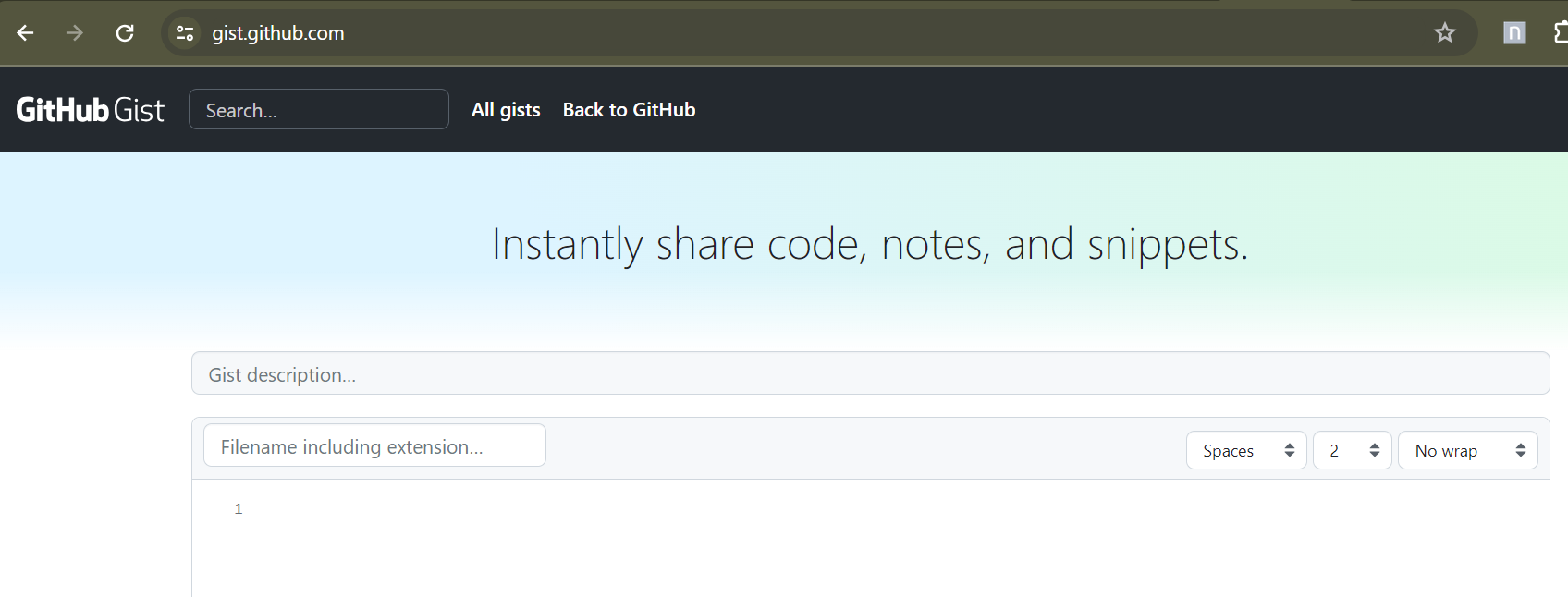
Podemos crear Teams para diferenciar cierto grupo de personas y privilegios

### Gist

Mini repositorio publico. Uno de sus usos mas frecuentes es la generacion de bibliotecas de codigo reutilizable. Otro uso es para compartir codigo con otras personas.

#### Creacion





Ejemplo:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

#### Plugins

En VS Code

A screenshot of a computer

Description automatically generated

Creamos token de acceso:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

En Vs Code en View -> Command Pallet: Seleccionamos o tipeamos -> Gist: Select Profile. Esto pide el token de Github y luego nombre de perfil en vs code(ej: VSCode-Gists) y listo. Luego cuando hagamos Open Gists van a aparecer los nuestros y podemos buscar entre ellos, tambien podemos insertar el codigo del Gist:

