Git se basa en un servidor, en repositorios y ramas.

Cuando en un comando se vea un solo signo de menos ( - ) es una abreviatura y cuando sean 2 signos menos ( - - ) significa que es una palabra completa.

**Git help** => para ayuda sobre los comandos

**git config –global user.name “Jose Luis” =>** Primero establecemos el nombre de quien ocupa la consola

**git config –global user.email “joe\_vm92@hotmail.com” =>** registra nuestro correo

**git config --global –e** => lista los datos del usuario que modifica el código

**Git init** => inicializa el repositorio

**Git status** => te indica los archivos que están pendientes de commit tanto normales como los que ya se encuentran en stage o los que no se le están dando seguimiento que son los marcados en rojo

**git status --short** => muestra de forma resumida los cambios que se hicieron en el código

**Para quitarlo de stage se colocafet**

git rm --cached <file>…

**Para agregarlo al stage se usa**

git add <file>…

**git add .** => sube al stage todos los archivos modificados

**git reset <archivo>** => quita el archivo del stage

**git commit** => es para hacer la protección correspodiente en la rama

**Clear** => limpia la consola

git commit --m “Mensaje del commit” => comando para hacer el commit a la rama

git checkout -- . => lo que hace es reconstruir el proyecto a como estaba en el commit anterior

**git checkout – README.md** => regresa solo ese archivo a como estaba antes de los cambios

git branch => indica la rama en la que estoy trabajando

Cuando se ingresa a una parte de la consola para hacer modificaciones para salir primero presionamos Esc y después wq!

**git branch –m master main =>** Para cambiar el nombre de una rama: master (nombre actual) a main(nuevo nombre)

**git checkout -- .** => se usa para regresar el proyecto al commit inmediato anterior

**git add index.html main.html** => para agregar dos archivos al stage de pueden separar por espacios

**git add \*.html** => se pueden utilizar comodines para agregar los archivos al stage, en este caso, se estarán agregando todos los archivos que tengan extension HTML

**git add js/\*.js** => cuando se agregan los archivos por comodin, se debe de tomar en cuenta el directorio y colocarlo en el comando para que los agregue al stage

**Nota:** Cuando se agrega una carpeta vacía no le da seguimiento GIT hasta que se le agregue un archivo. A menos que se requiera proteger una carpeta vacía se tiene que añadir el archivo .gitkeep y ya te va a dejar proteger el directorio vacío.

Para añadir ese directorio se puede hacer con lo siguiente:

**git add uploads/** => de esta forma se añade todo el directorio aunque solo tenga dentro el archivo .gitkeep o añadiéndolo como **git add uploads/.gitkeep**

**CREACIÓN DE ALIAS**

Para evitar colocar todo el comando completo se pueden utilizar alias que resumirán el comando que quieres ejecutar, por ejemplo:

git config --global alias.s “status –short”

la s es el alias que resumirá el código, lo que se escriba después del espacio deberá de ir entre comilla simple para que te respete espacios del comando

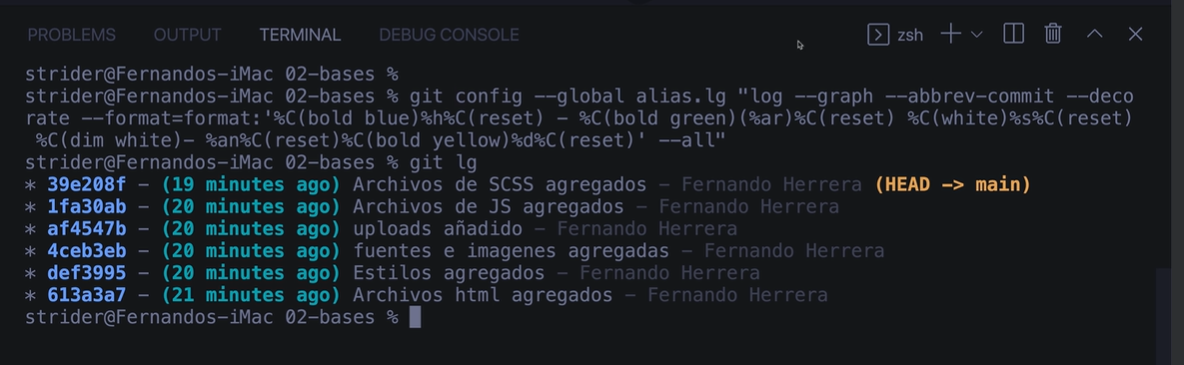
**git config --global -e** => sirve para editar un alias, al ingresar se toca la tecla **a** para hacer la modificación, se navega hasta el alias que se desea editar y para salir se presiona ESC y después al final se escribe :wq! Para que te permita salir y ya estaría el alias

y para ejecutarse sería llamando al alias por ejemplo **git s**

**git log --oneline =>** se usa para dar un resumen de lo que da git log, por ejemplo el hash corto de los commit y el comentario que se le puso

El siguiente ALIAS es para mostrar la versión corta del log, hace cuando se hizo, el comentario, quien lo ejecutó y donde estoy parado

**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"**

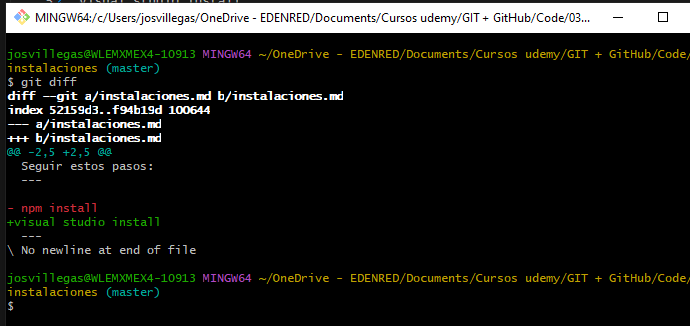


Si al colocar el alias lg no se muestra todo el listado y la línea preparada esperando código, solo colocas la letra q y finaliza la instrucción.

**Sección 3**

**git diff =>** se utiliza para saber qué fue lo que se modificó, los símbolos de + y – indican que se añadió o se removió respectivamente.

Si son 3 símbolos juntos indica que hubo varios cambios (Solo da la diferencia de los cambios que NO están en el stage)

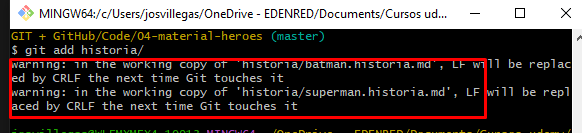


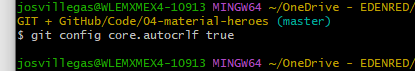
**git diff –staged =>** Si queremos obtener los cambios de las clases que están en el stage

**git commit --amend -m “Instalaciones actualizadas”** => modifica el comentario realizado al commit anterior donde amend está relacionado a arreglar y el -m es referente al mensaje y después el mensaje que vamos a colocar.

Para cambiar los comentarios en antes de que se mande al repositorio externo para que no marque errores en los merge

**git config core.autocrlf true** => Cuando en la consola nos sale un error de CRLF ejecutamos el comando

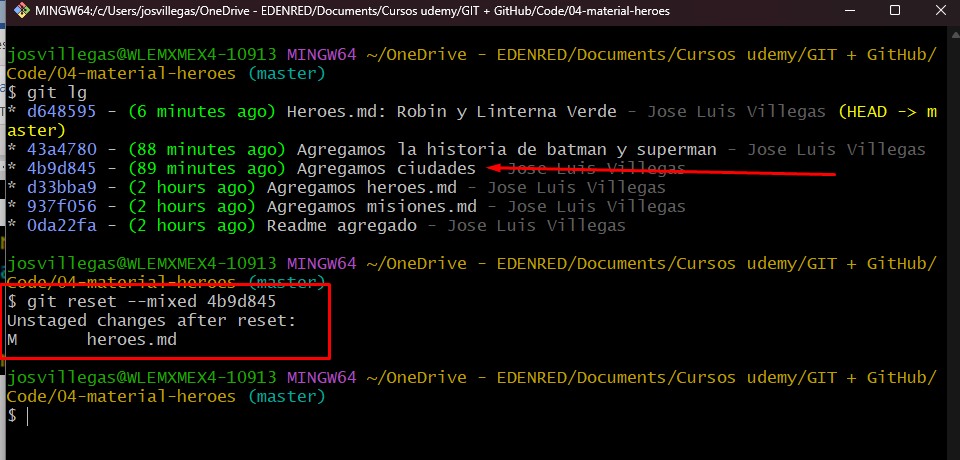


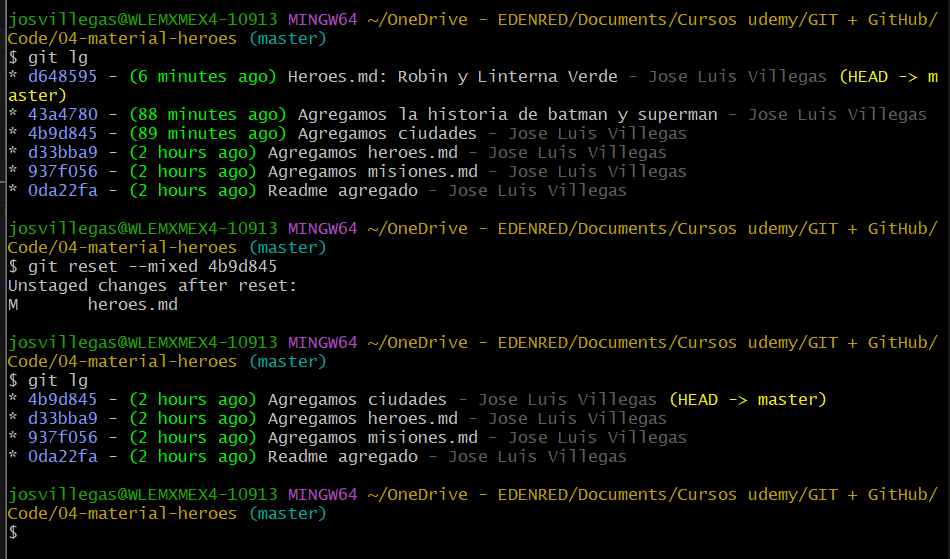


**git reset --soft 43a4780 =>** Para posicionarnos en otro commit necesitamos el ID que genera GIT

la parte de --soft es para que el restablecimiento al commit no sea destructivo y me mantenga los cambios.

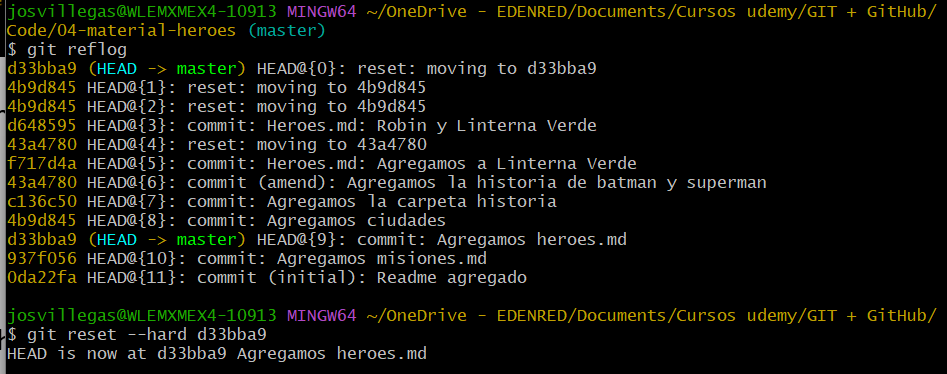
**git reset --mixed 43a4780 =>** Para posicionarnos en el commit y que saque todos los cambios del stage y los cambios los podemos volver a añadir se usa --mixed



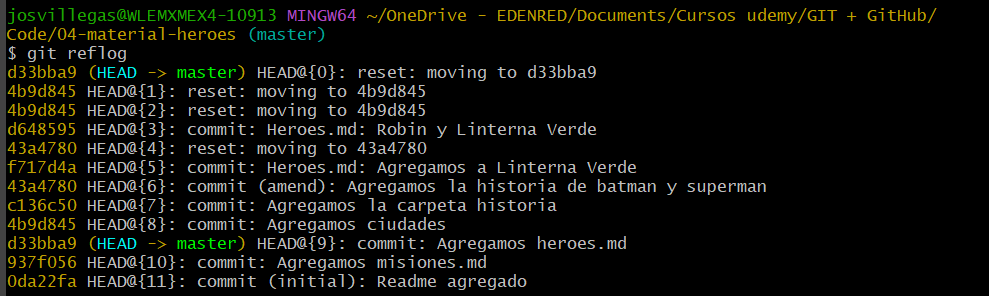


**git reset --hard d648595 =>** Para posicionarse en otro commit y que descartemos los commit que estaban por delante se usa

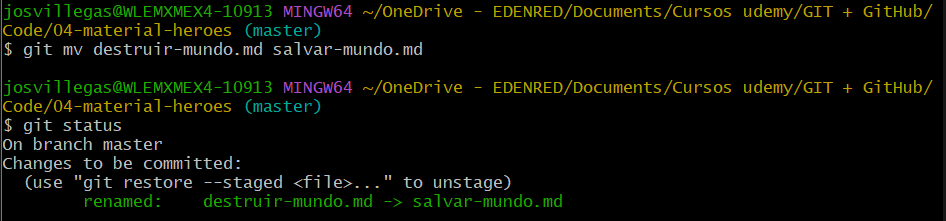
--HARD



**git reflog =>** es una referencia de todo lo que se ha generado en GIT



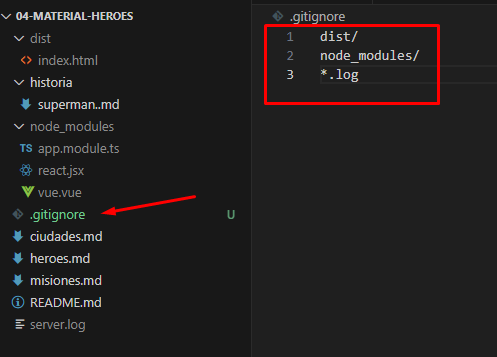
**git mv destruir-mundo.md salvar-mundo.md =>** git mov lo que hace es mover un archivo, se indica primero el directorio y como en el caso anterior, si es el mismo directorio no se pone la ruta y se aprovecha para renombrar



**git rm salvar-mundo.md =>** elimina el archivo y después se tiene que hacer commit para confirmarlo.

Para que no se haga el seguimiento se algunos archivos se puede crear un archivo **.gitignore**

Ese archivo tiene que estar en la misma carpeta que se encuentra la carpeta git que es la que hace el seguimiento.



Una rama en GIT es una línea de tiempo de commits

Existen 3 tipos de Merge:

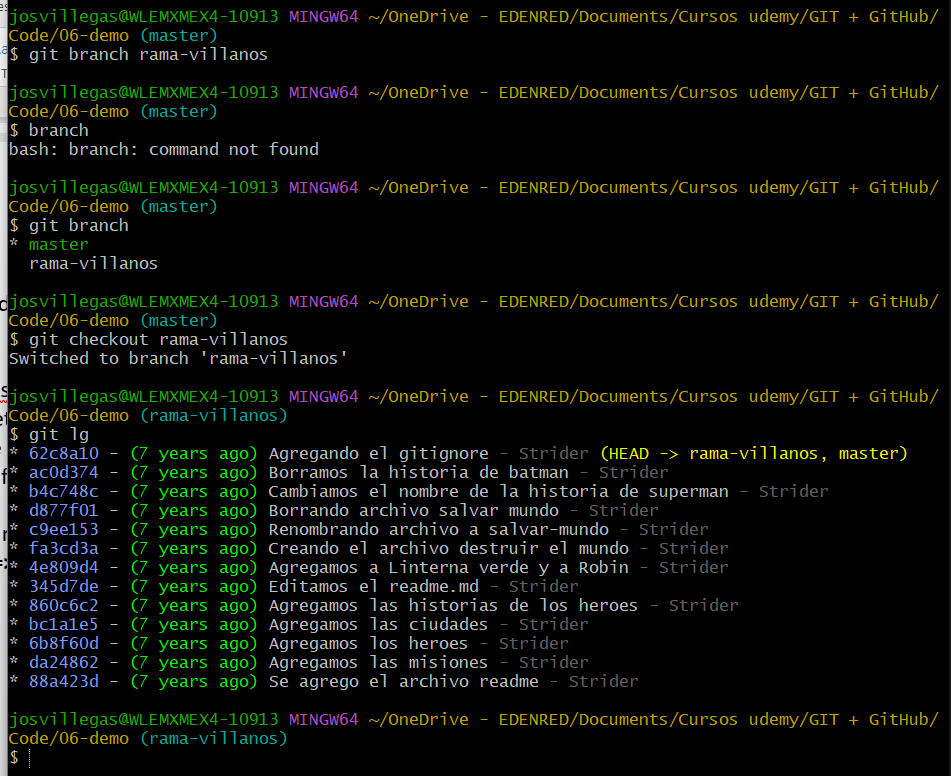
**Fast-forward:** Une los commits a la rama principal como si nunca hubieras generado otra rama.

**Uniones automáticas:** Git detecta que en la rama principal hubo un cambio que en la rama secundaria desconoce, git une las ramas de forma automática.

**Manual:** Gitencuentra un conflicto en el merge entre los cambios de los commits.

**git branch =>** para ver o crear ramas.

**git checkout [nombre rama] =>** se utiliza para moverte de rama.

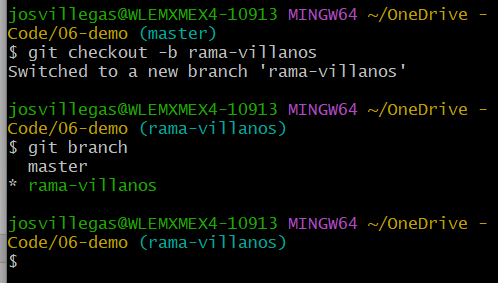


Si queremos hacer un merge debemos estar parados en la rama que espera los últimos cambios.

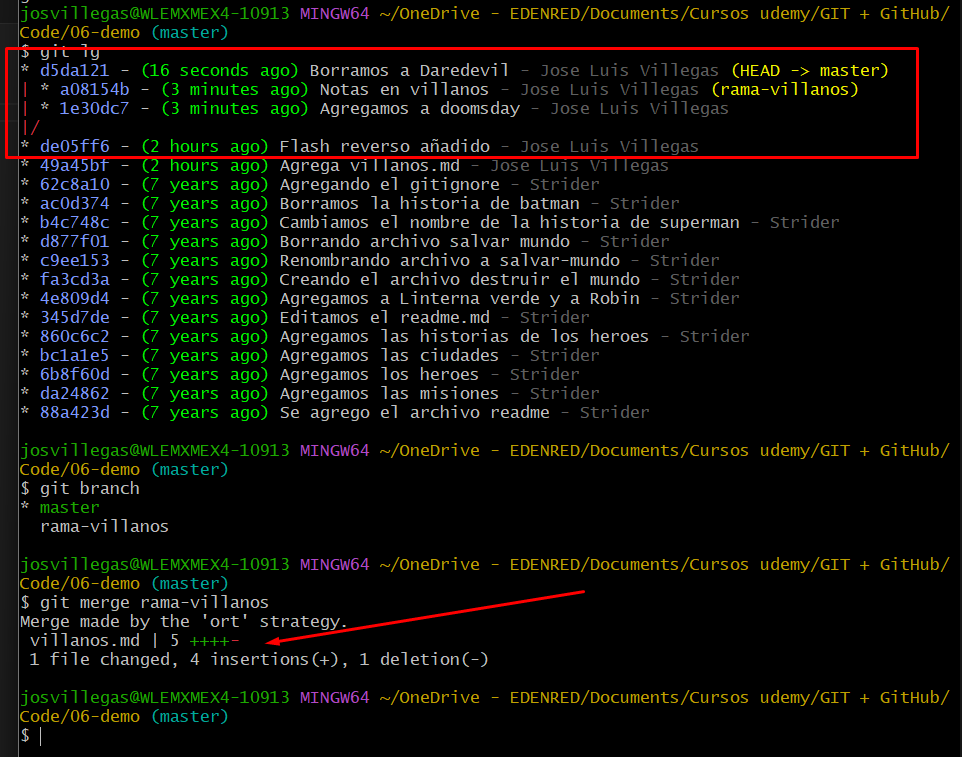
**git branch -d rama-villanos =>** borrar una rama después de hacer el merge

**git branch -d rama-villanos -f =>** borra la rama de manera forzada en caso de que se tengan commit que no han realizado el merge

**git checkout -b rama-villanos =>** Crea la rama y se posiciona en la nueva rama para hacer los cambios

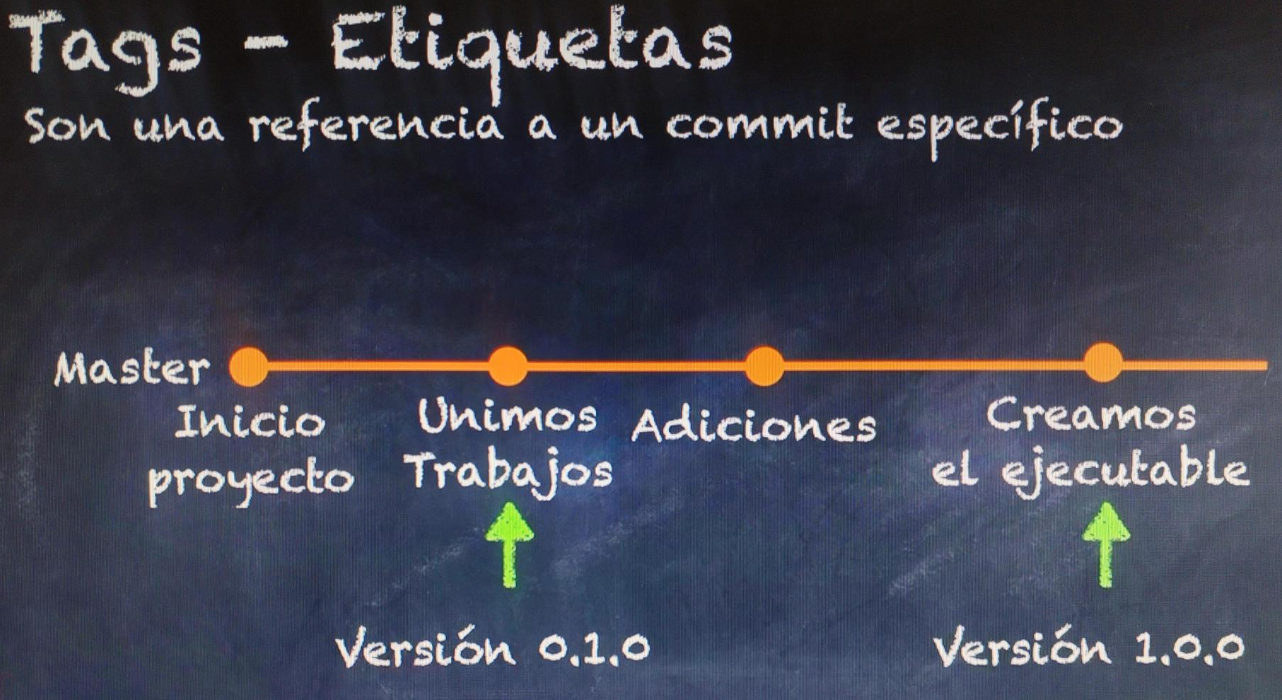


Cuando se provocan conflictos se puede visualizar las ramas de la siguiente manera y cuando se hace merge se hace la estrategia recursiva donde GIT analiza los conflictos y de manera automática lo logró unificar.

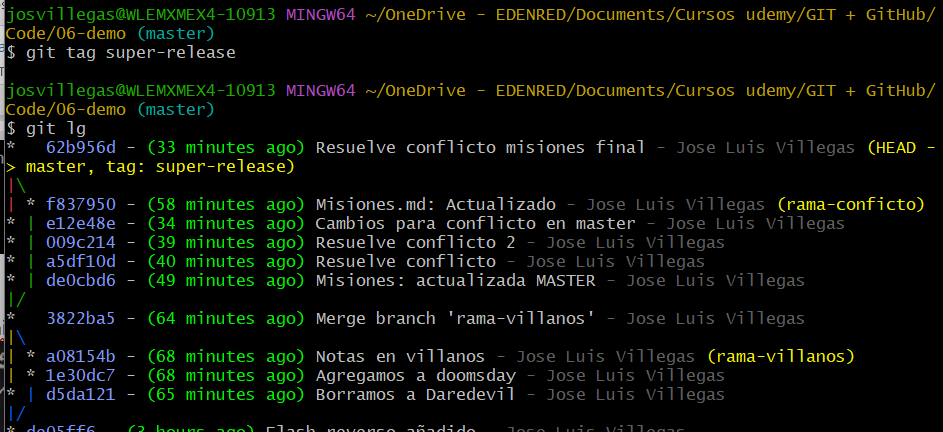


**TAGS Y ETIQUETAS**

Son una referencia a un commit específico en la línea de tiempo:



**git tag super-release =>** crea un tag con el commit en el que se encuentra en ese momento.

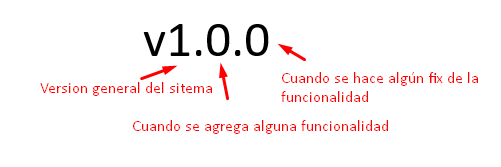


Git tag => se usa para ver los tags creados

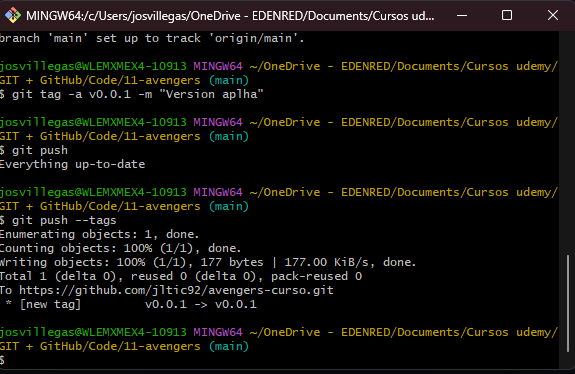
git tag -d super-release => se utiliza para eliminar el tag

Se acostumbra trabajar con estructura semántica para los tags.

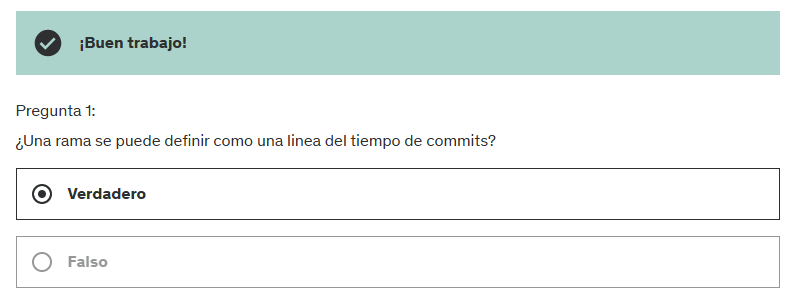
**git tag -a v1.0.0 -m “Version 1.0.0 lista” =>** Crea el tag y utiliza un versionamiento

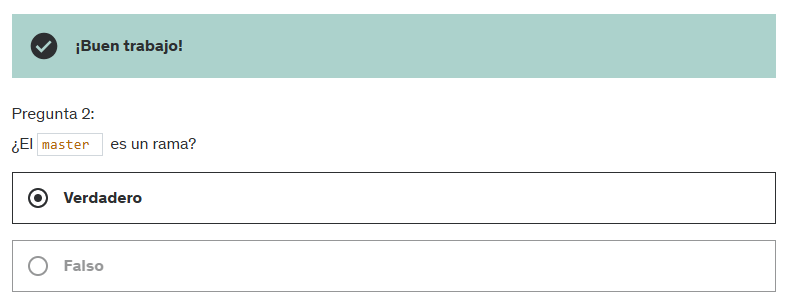


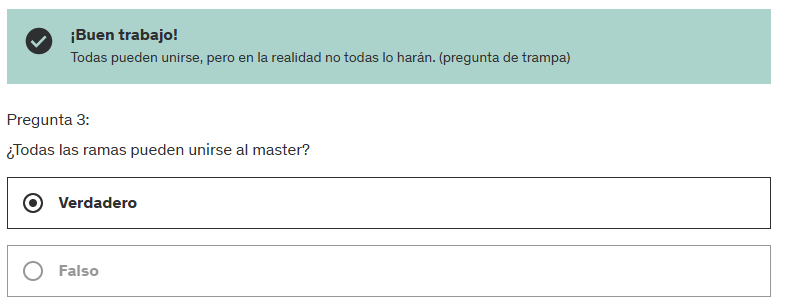
**git push –tags** => se utiliza para subir al repositorio los tags cuando marca que no han cambios pendientes por enviar el repositorio remoto

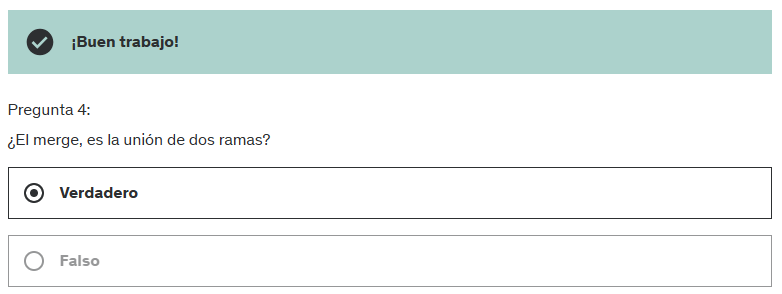


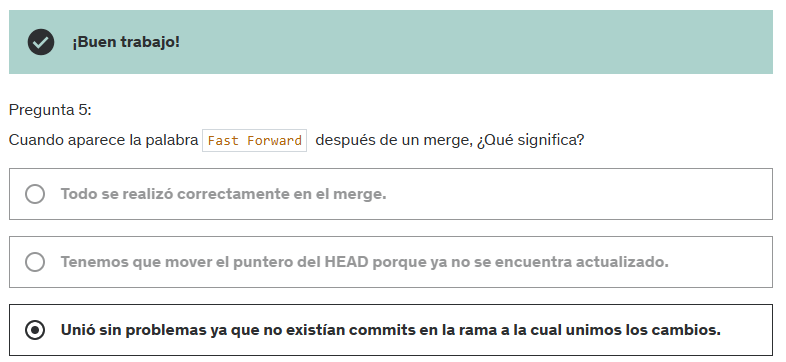
**git show v0.1.0 =>** Muestra el detalle del tag

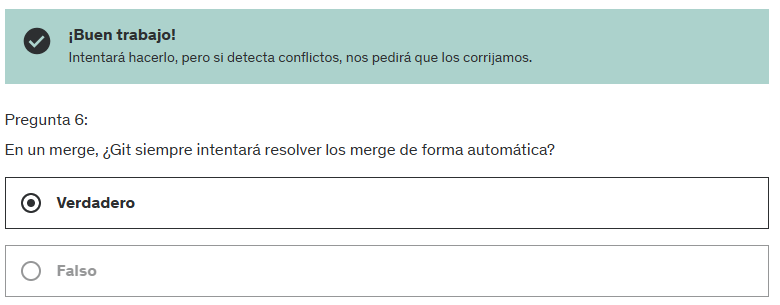




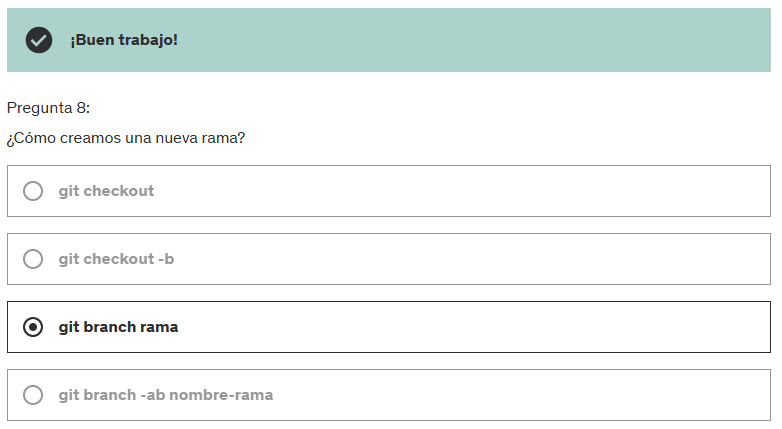












**Sección 5**

**git stash =>** permite guardar modificaciones sin protegerlas en el código.

**git stash list =>** muestra el listado de cambios guardados del más reciente al más antiguo

**git stash apply stash@{2} =>** recupera los cambios almacenados y los aplica

**git stash drop =>** borra el primer stash que está en la posición cero {0}

**git stash drop stash@{0} =>** borra el número de stash indicado, en caso de ser el primero no es necesario colocar el número, se puede ejecutar como el anterior solo con el drop.

Para revisar los stash

**git stash show stash@{1} =>** muestra las clases o elementos que se cambiaron y se almacenaron en ese stash.

**git stash save “Agregamos X” =>** Guarda en el stash con un comentario personalizado.

**git stash list --stat =>** Muestra un mayor detalle de lo que almacenamos en el stash

**git stash pop =>** toma el ultimo stash, aplica los cambios en el código y elimina el stash

**git stash clear =>** borra todos los stash

**REBASE**

Cuando se crea una rama por ejemplo desde master, se hacen 2 commits pero a la par en master se suben dos commits, entonces, si queremos actualizar la rama creada podemos usar Rebase.

Pasos

1.- Se crea rama-misiones

2.- Se mueve a la rama con “git checkout rama-misiones”

3.- Se hace un “git rebase master” esto lo que ocasiona es que los commits que se tengan pendientes de integrar a master se muevan a un espacio temporal mientras se incorporan los otros cambios previos

4.- Mueve el puntero de la rama al último commit una vez incorporando los últimos cambios

5.- Regresa los commits que estaban en el espacio temporal.

**Rebase interactivo**

**git rebase -i HEAD~3 =>** Con este puedes obtener el número de commits hacia atrás desde HEAD o desde cualquier commit, crea una línea temporal donde almacenará los commis indicados después del símbolo (~) y después los regresa

Funciones:

1.- Ordenar commits

2.- Corregir mensajes de los commits

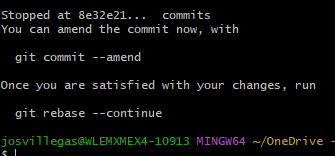
3.- Unir commits

4.- Separar commits

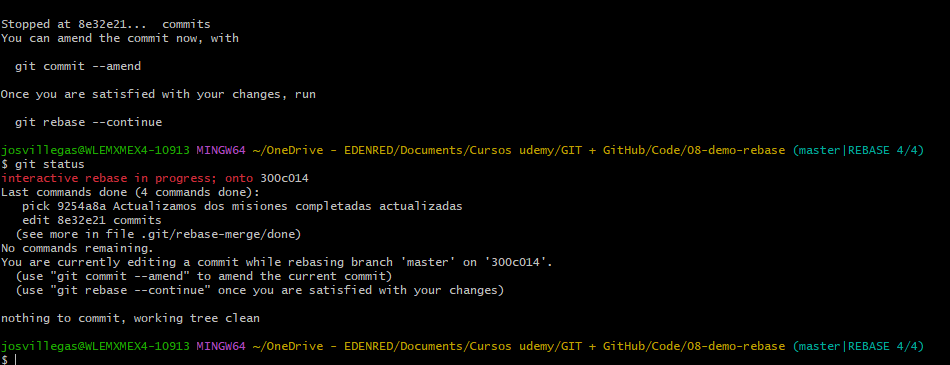
NOTA!! Las modificaciones del rebase, nombre del commit se tienen que hacer solo si los commit se encuentran todavía de forma local, si ya se hizo push al repositorio remoto es mejor no moverlo o se puede causar conflictos

**git reword** se usa después del comando anterior, en lugar de poner pick se pone r y te va a cambiar las ventanas para los N commit que deseas cambiar el mensaje

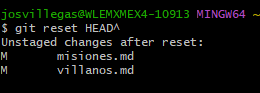
git edit se utiliza para modificar commits, por cada commit se pueden incluso separar las modificaciones y crear commits por separado



Aquí si hacer un git status no te deja por que estás en medio de un rebase



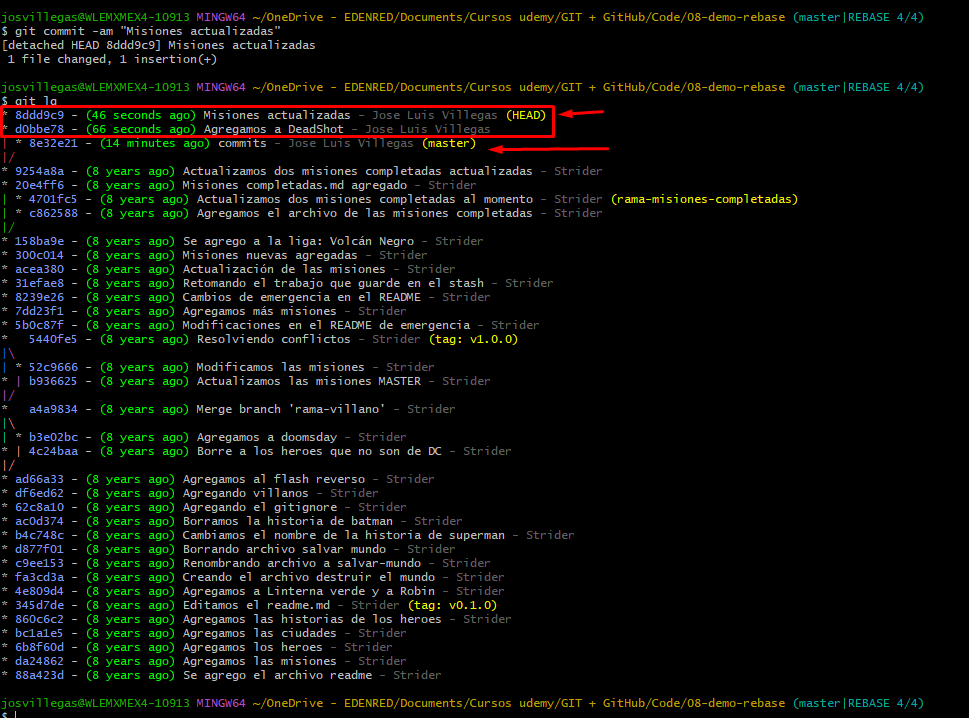
Se coloca git reset HEAD^ para ir al commit anterior



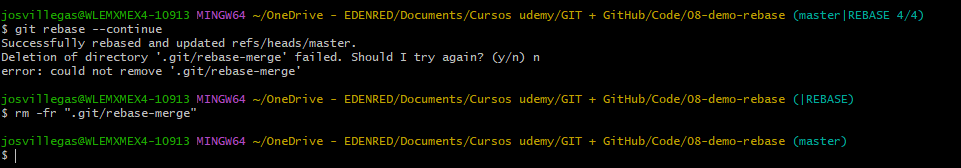
Esto te colocará en el estado antes de hacer el commit para ambos archivos

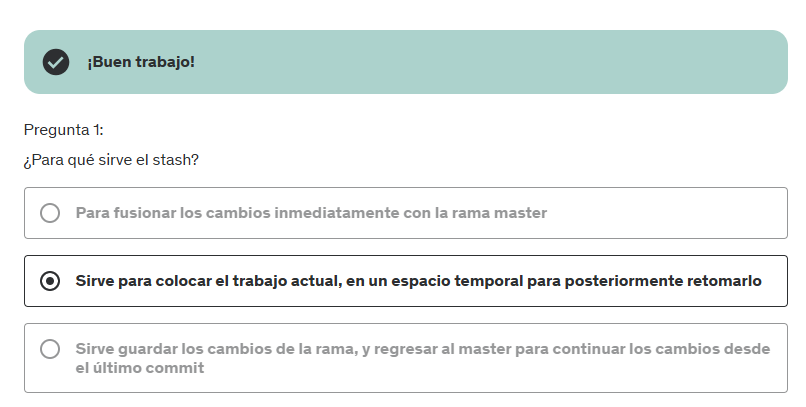


Esto ya realiza los dos commits pero el HEAD está en lugar diferente al master

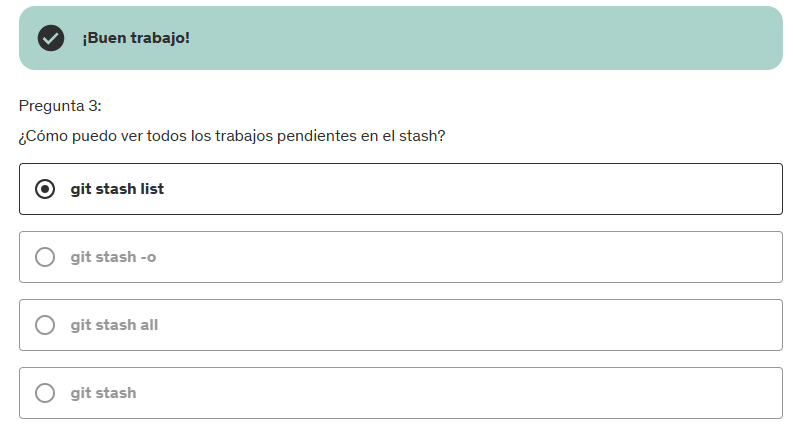


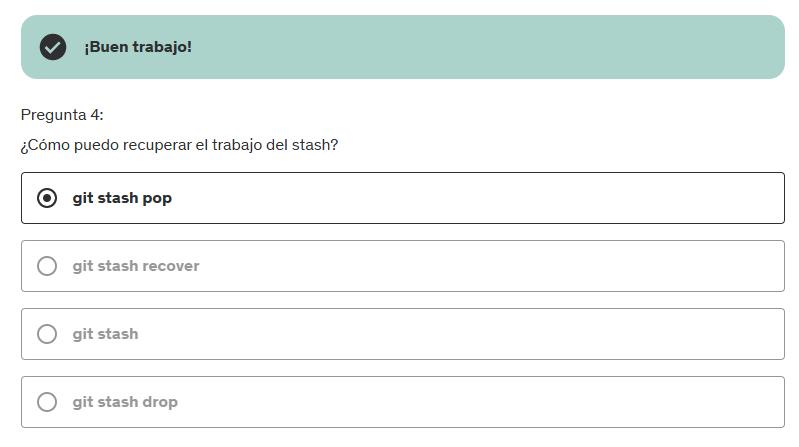
Para terminar el rebase de usa git rebase –continue

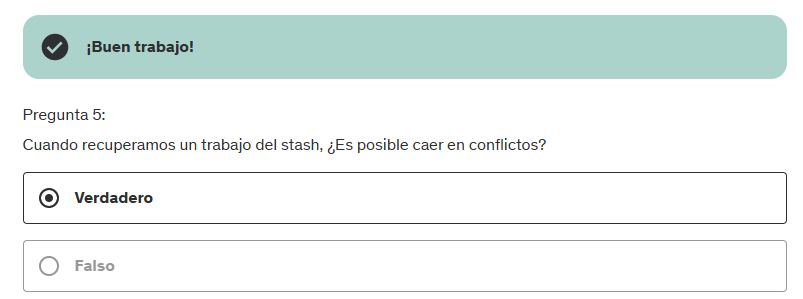


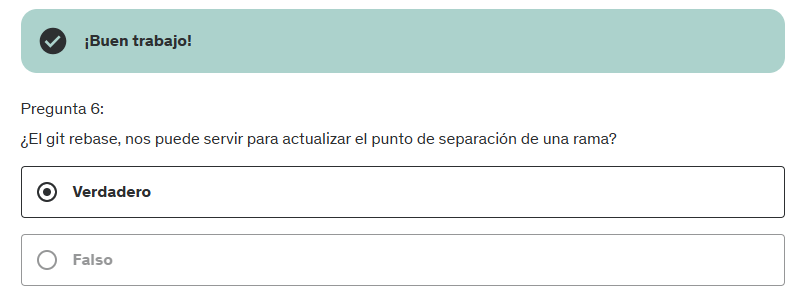














**Sección 6**

**Push** => tomar nuestros últimos cambios de nuestro repositorio local y subirlo a un servidor

**Pull** => es para obtener los últimos cambios del repositorio remoto

**Github** => es una plataforma de desarrollo colaborativo de software para alojar proyectos.

Ventajas de Github:

1. Repositorio ilimitados
2. Paginas HTML, CSS y JS ilimitadas
3. Push, pull, clones ilimitados
4. Issues, wikis, estadísticas ilimitadas
5. Organizaciones ilimitadas
6. Participación gratuita de proyectos ilimitados.
7. Permite crear repositorios públicos y privados.

**git remote add origin <URL>:** Cuando estamos en nuestro repositorio local y queremos subirlo a un repositorio remoto

**add:** nuevo remote

**origin:** el nombre de nuestro remote

**URL:** dirección de GITHUB que se da al finalizar la configuración en github

**git remote -v** : Cuando queremos revisar las fuentes remotas que tenemos agregadas en el repositorio

**git push -u origin master:** indica la rama que se desea enviar al remoto

**-u:** Nos ayuda a que la próxima vez que queremos hacer push, no necesitamos especificar la rama.

**origin:** nombre del repositorio

**master:** rama que deseamos enviar

**¿Qué es Gitosis?**

<https://wiki.archlinux.org/title/Gitosis#:~:text=Gitosis%20is%20a%20tool%20which,system%20accounts%20on%20the%20server>.

**Instalación y configuración de Gitosis**

<https://github.com/res0nat0r/gitosis>

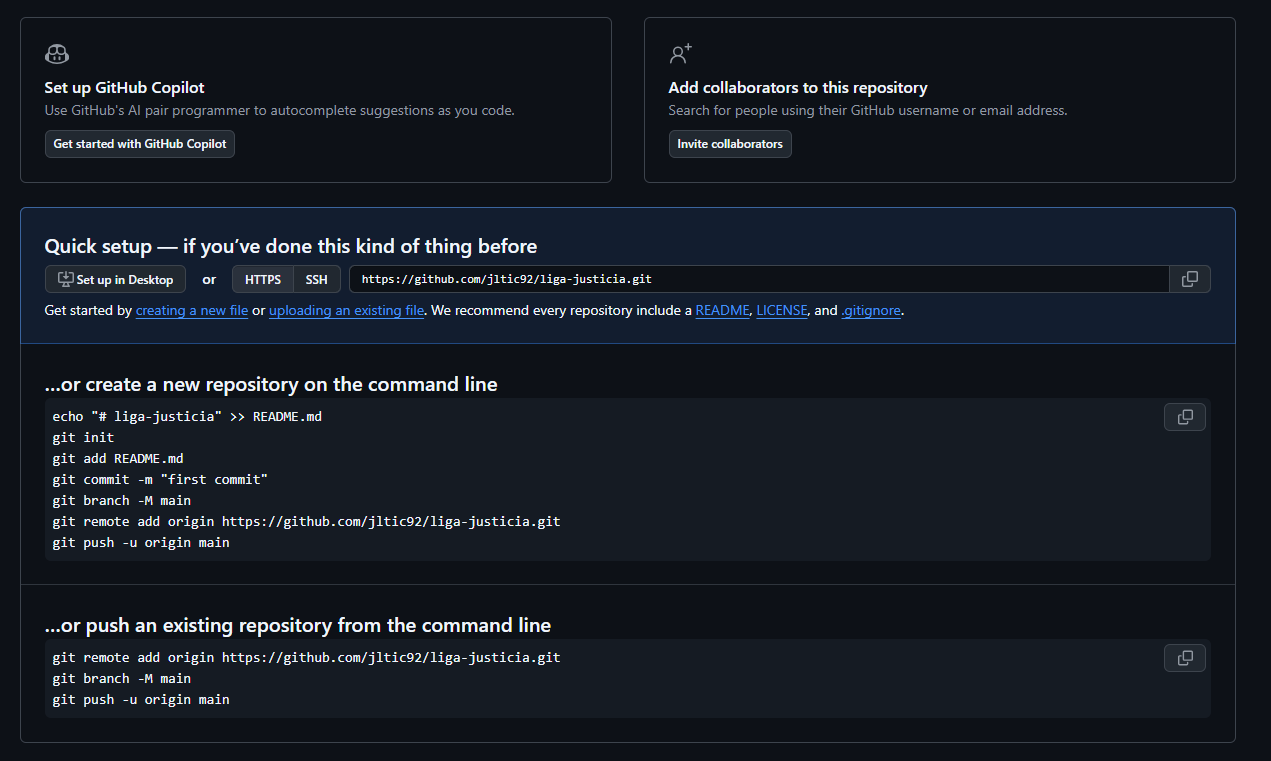
**Guardar su contraseña de GitHub en la máquina WINDOWS**

<https://docs.github.com/es/get-started/getting-started-with-git/caching-your-github-credentials-in-git#platform-windows>

Cuando se crea un repositorio en GitHub se debe de colocar un nombre único en nuestro usuario e indicar el tipo de permisos:

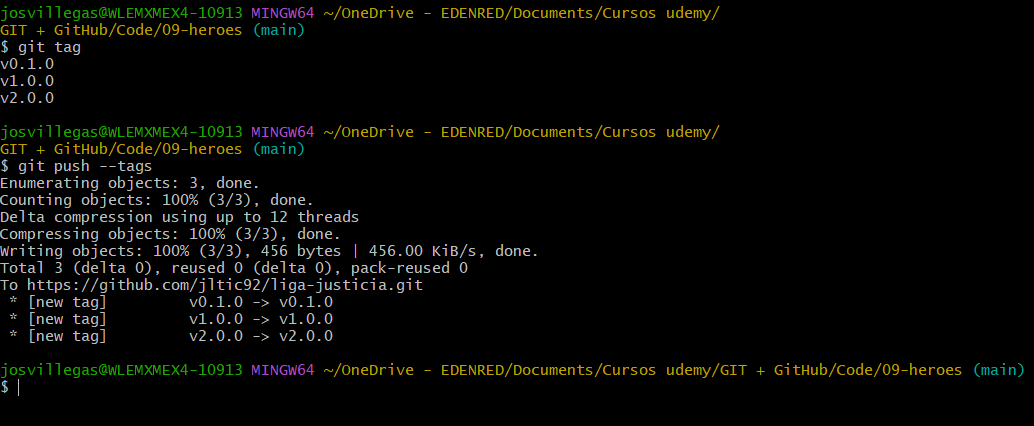
1. **Público**: todos tendrán a acceso a el repositorio, podrán clonarlo pero no hacer commits
2. **Private**: nosotros administramos a los usuarios que tendrán acceso

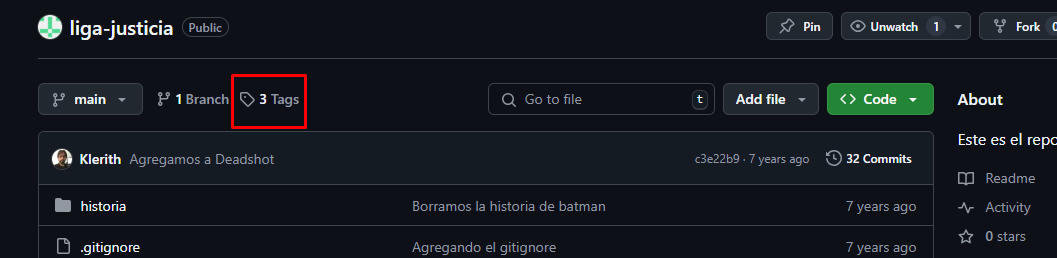
Al crear el repositorio da la opción para crear archivos o para solo subir archivos previamente creados

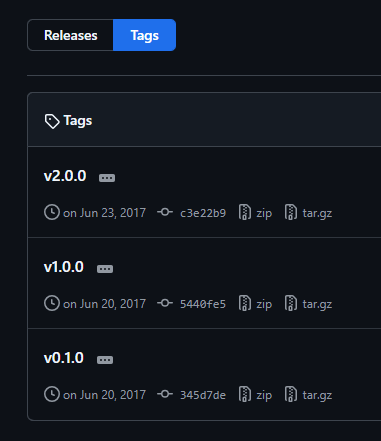


Se pueden copiar y pegar en la consola de git bash para que se ejecuten todos de una sola pasada.

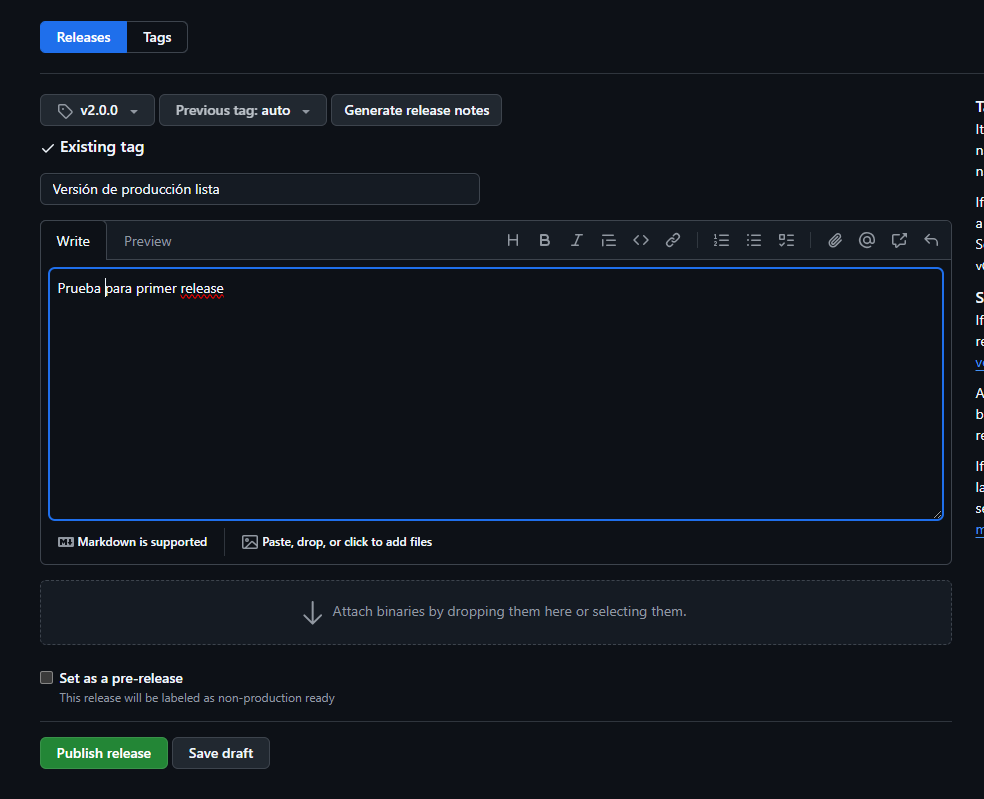
**git push –tags** => revisa contra el repositorio remoto todos los tags que se tienen de forma local y, en caso de no existir, los va creando



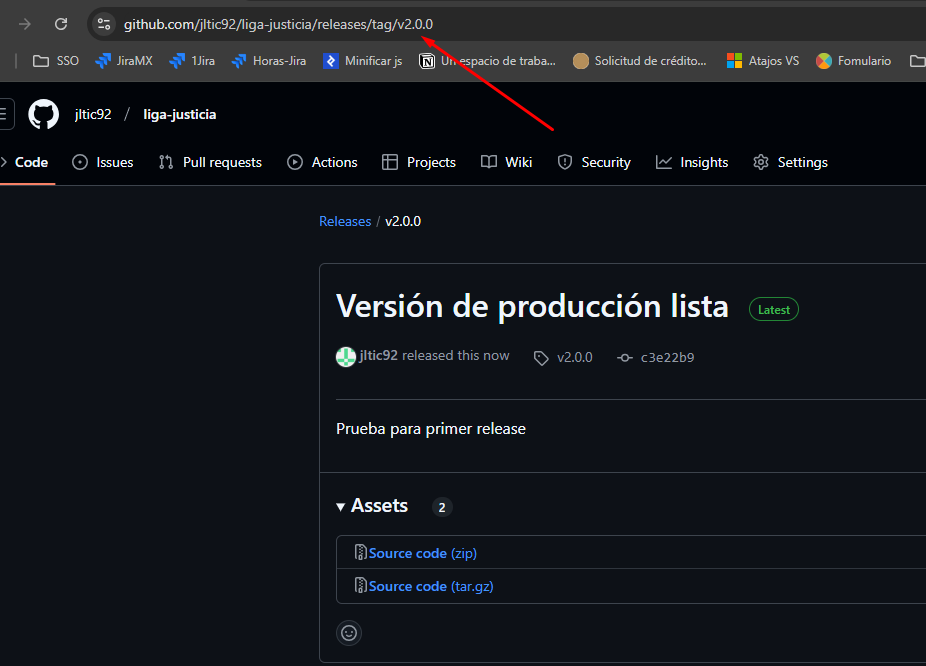


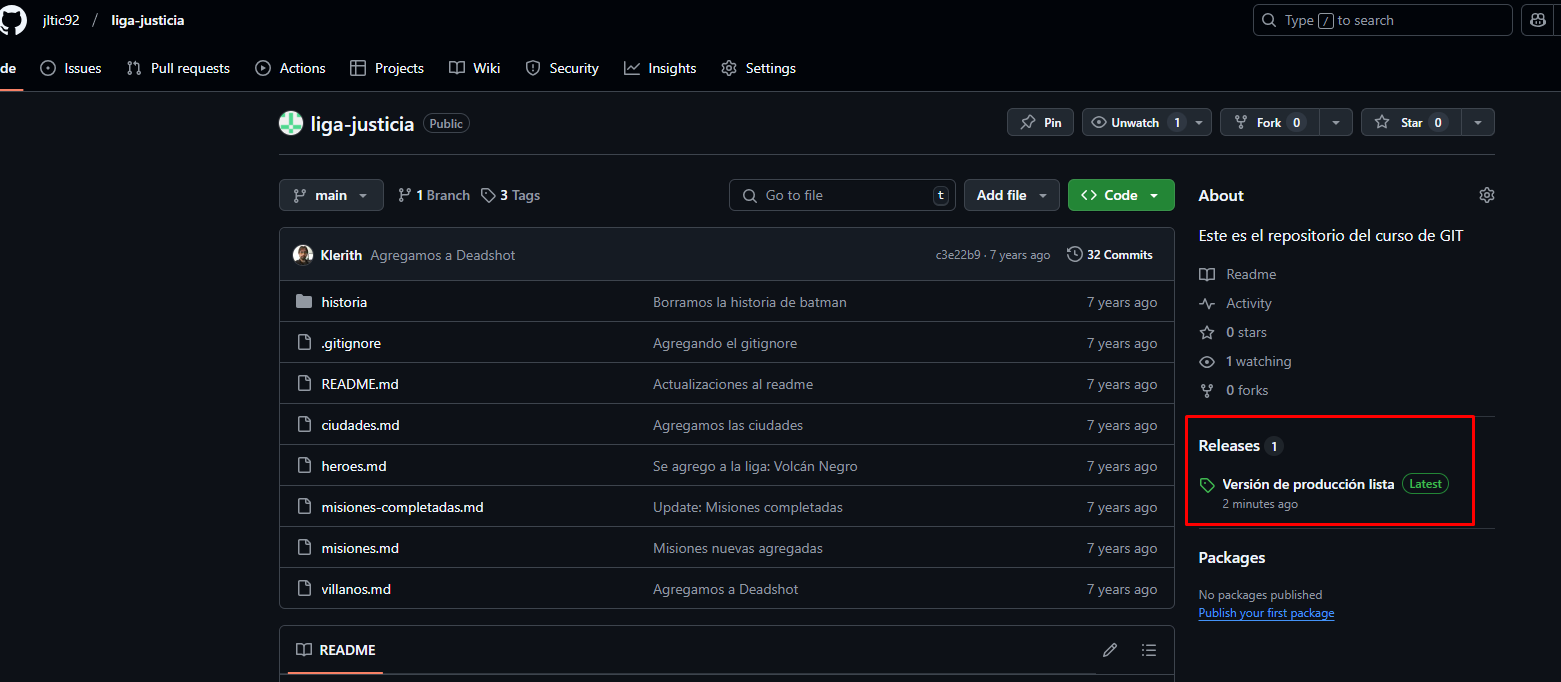
 Se pueden ver los cambios o descargar los archivos a como estaban en ese momento pero no clona el repositorio completo.

A partir de los tags se pueden crear elease en Git hub para que se descarguen los archivos conforme a los permisos de la versión indicada. Se pueden adjuntar imágenes, archivos binarios o el código fuente para que lo puedan descargar como apoyo.



Te crea una URL específica

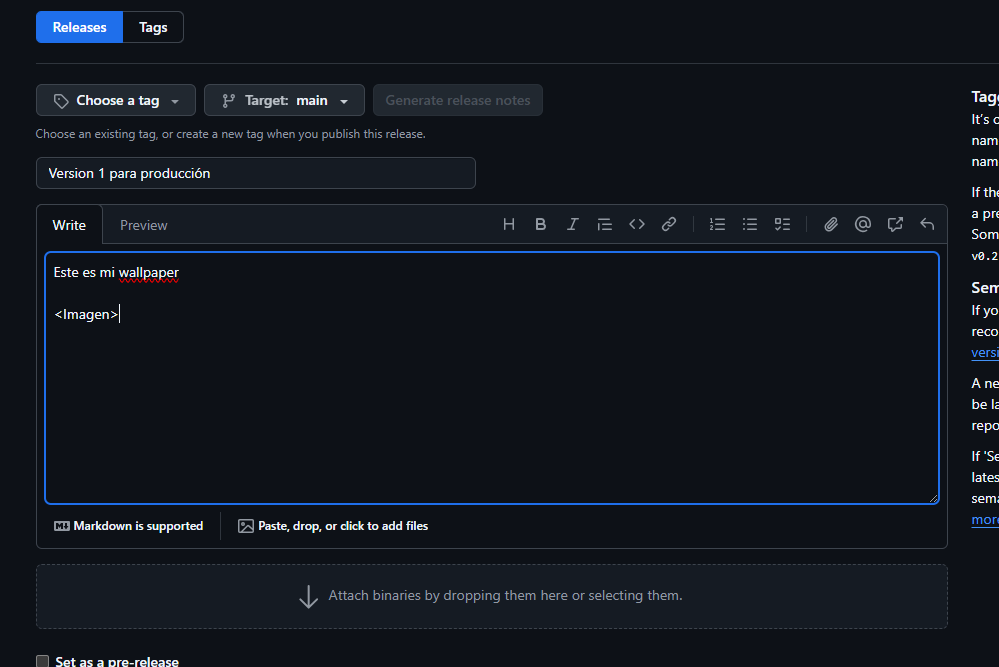




RELEASE

Cuando se crea se puede poner una etiqueta y una imagen pequeña que se mostrará en la vista preliminar del reléase donde dice <Imagen>, solo soltamos y arrastramos.

Se puede añadir la imagen en la parte de abajo donde dice adjuntar binarios, sin embargo, la diferencia es que será contenido que las personas podrán descargar junto con el compilado.



**Pull de cambios a repositorio**

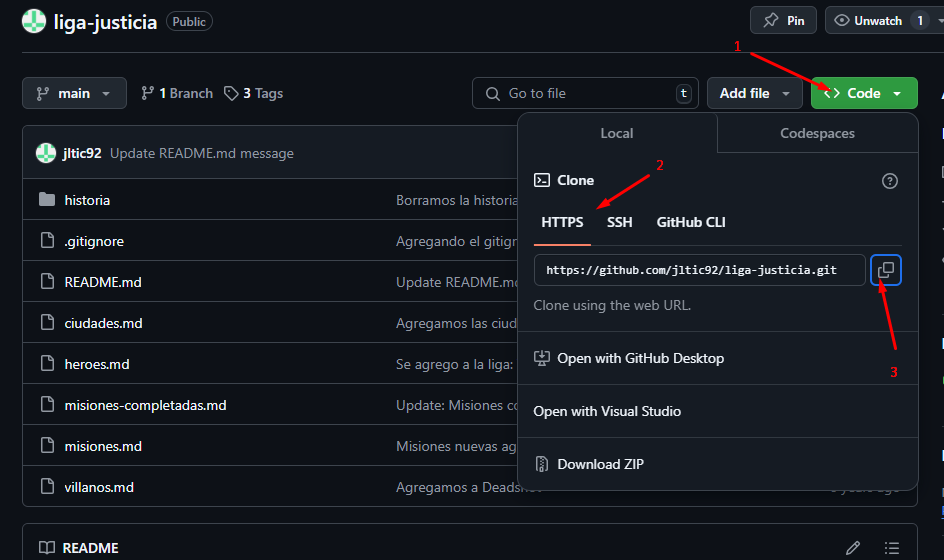
git pull => trae los cambios del repositorio remoto al local.

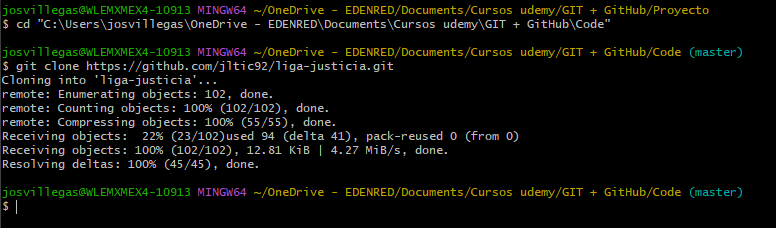
git remote -v => indica la ruta donde se va a subir los archivos



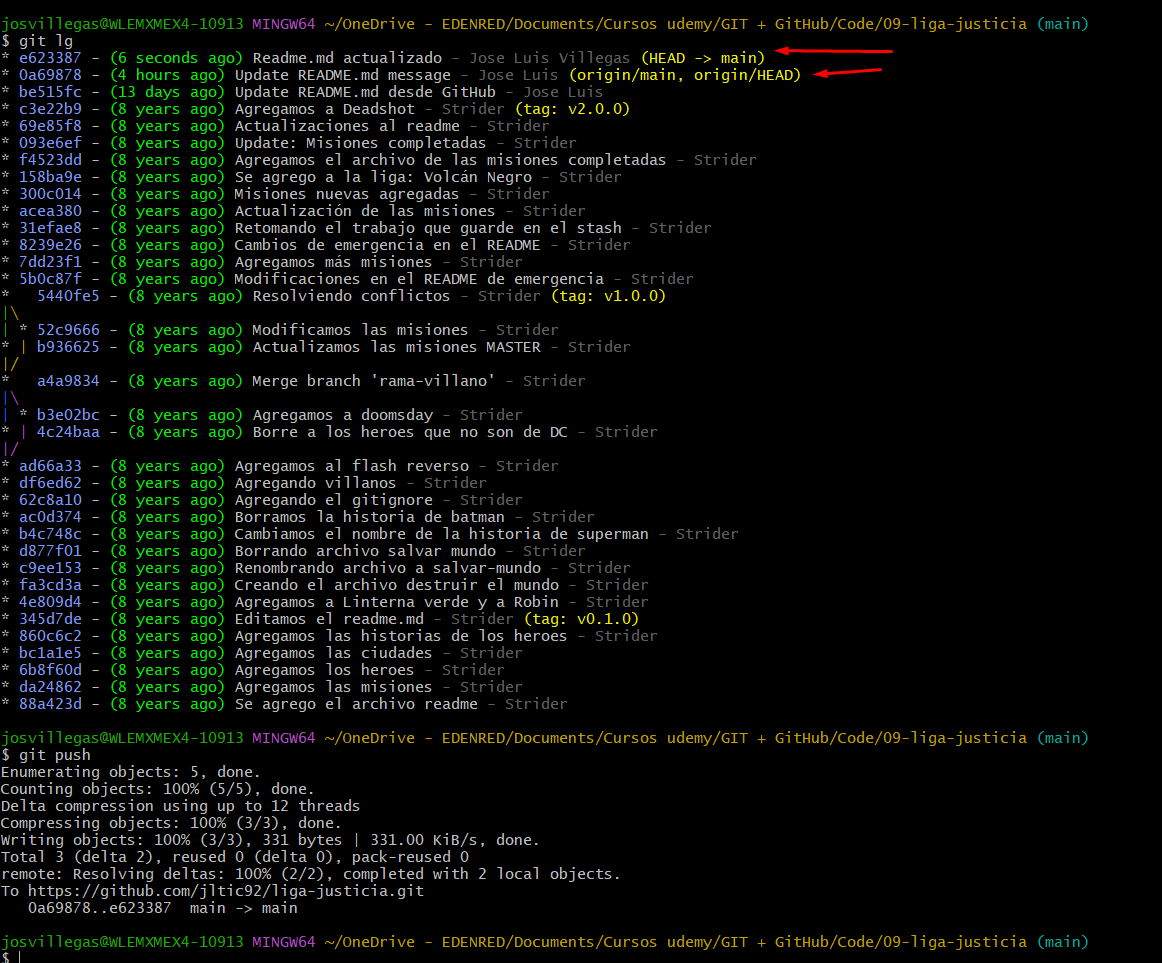
git config --global -e => muestra la configuración que se tiene para git.

git clone https://github.com/jltic92/liga-justicia.git => clona un repositorio dando clic en lo siguiente:

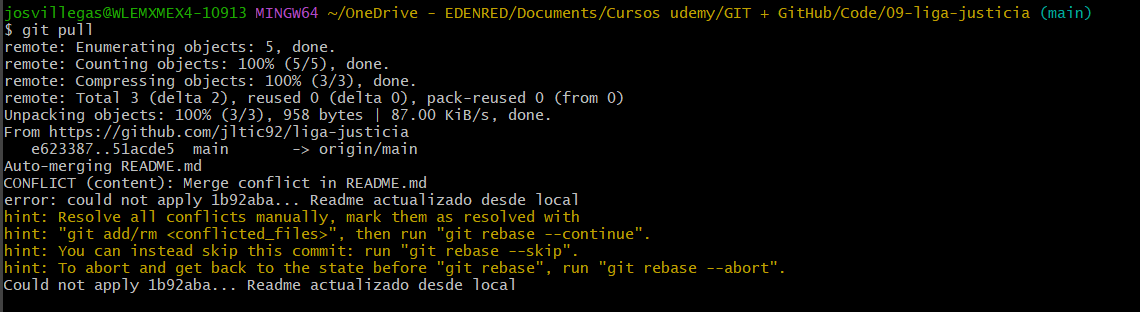




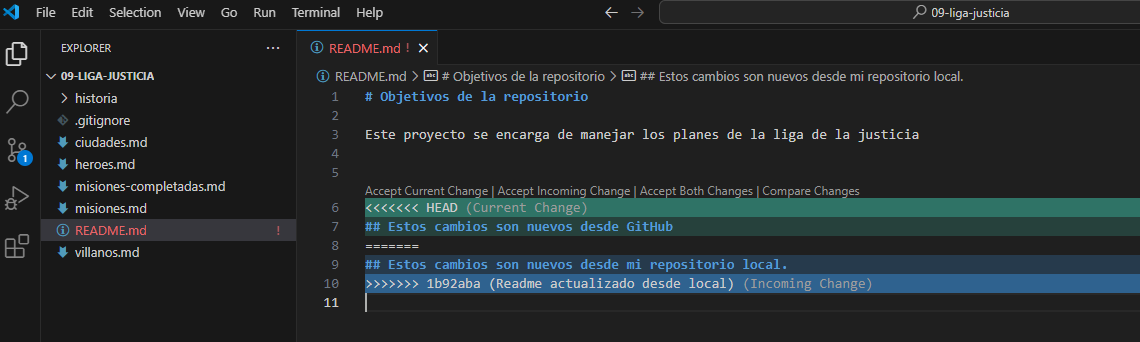
**git push** => Cuando se hacen cambios localmente y se quiere subir los commits del local al remoto



Si se hace cambio desde git hub y en local sobre el mismo archivo se ocasionará conflicto al hacer un git pull, para resolverlo

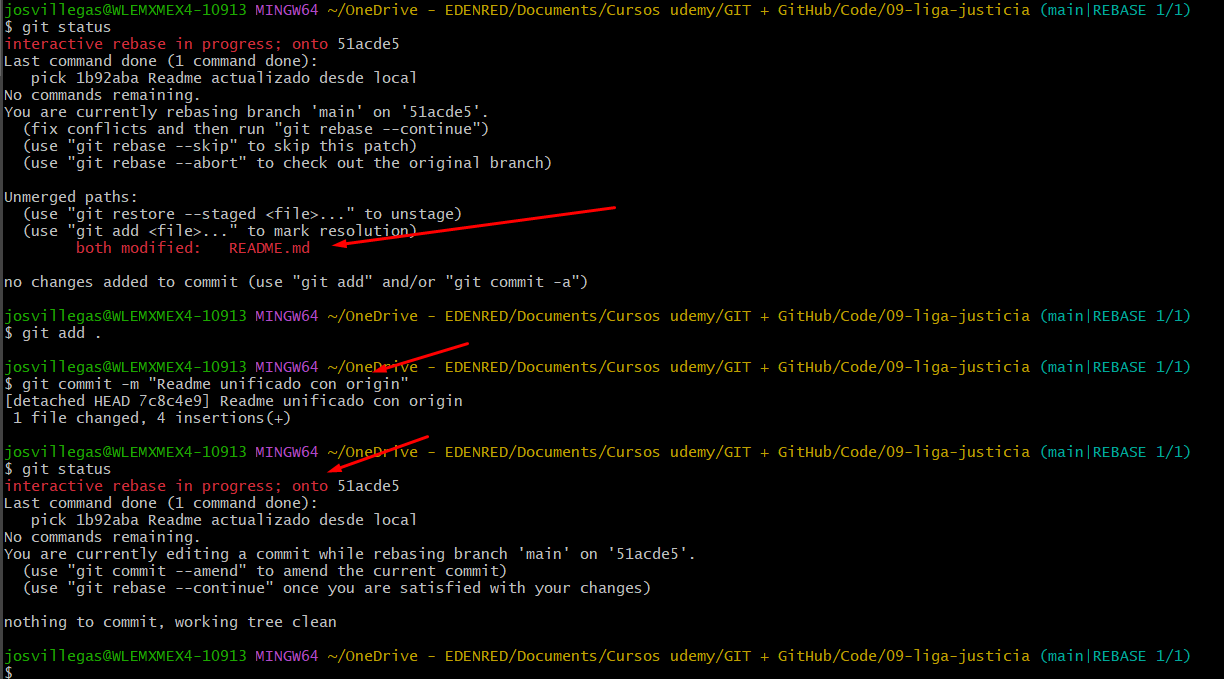


En el caso de visual studio code se puede hacer el cambio de forma manual y se mostrará lo siguiente:



**git config pull.rebase true** => En el caso que solo esté por fast forward y queremos ver lo de la pantalla anterior se tiene que ejecutar y después se vuelve a ejecutar el git pull.

En este ejemplo de aceptarán ambos cambios, se hace el commit pero la consola indicará que sigo dentro de un rebase



Se hace un git rebase --continue

**Sección 7**

**Markdown**

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

Forma de dar estilo a los textos en Github (ver documento PDF)

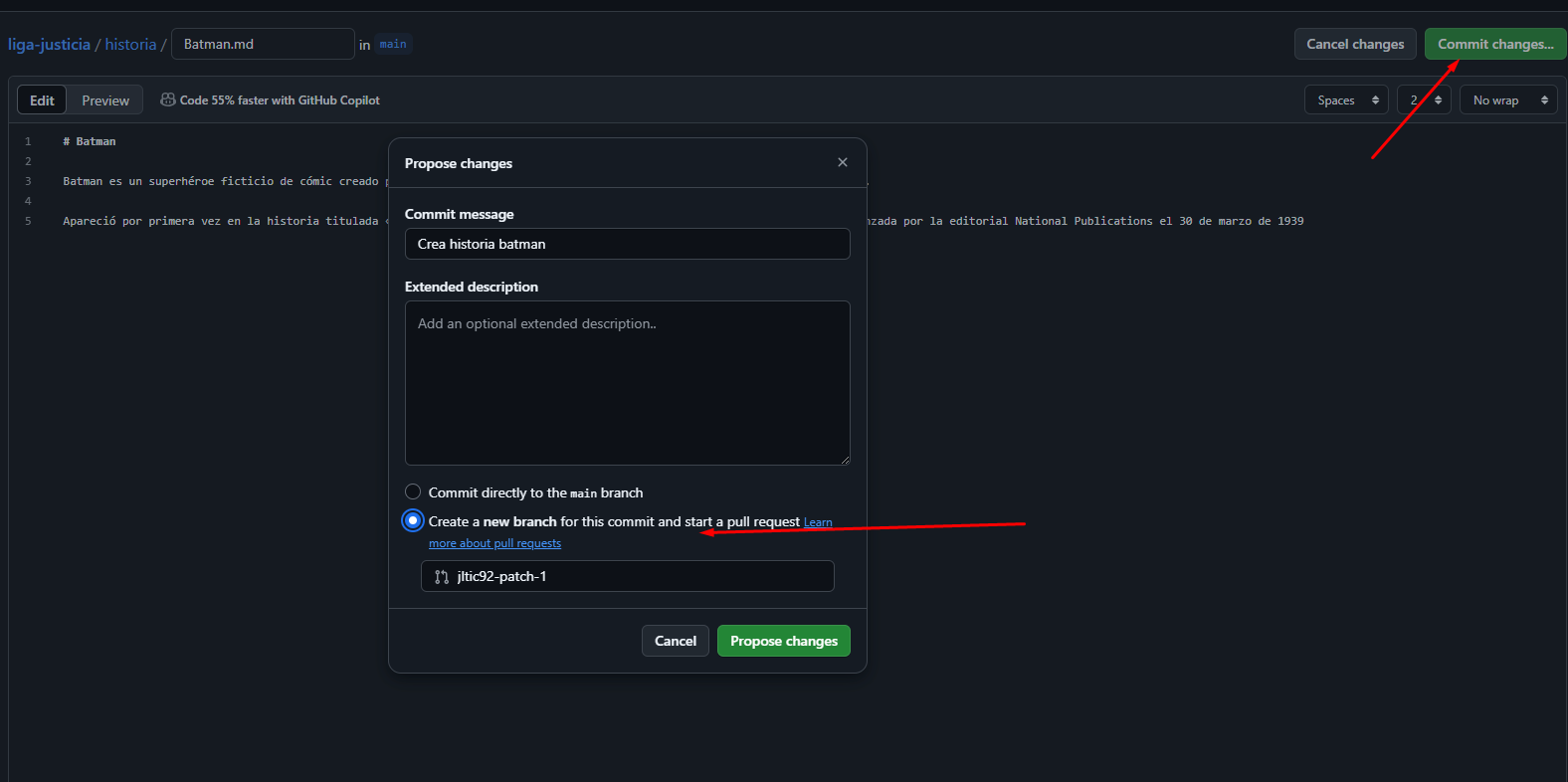
Documentación GITHUB

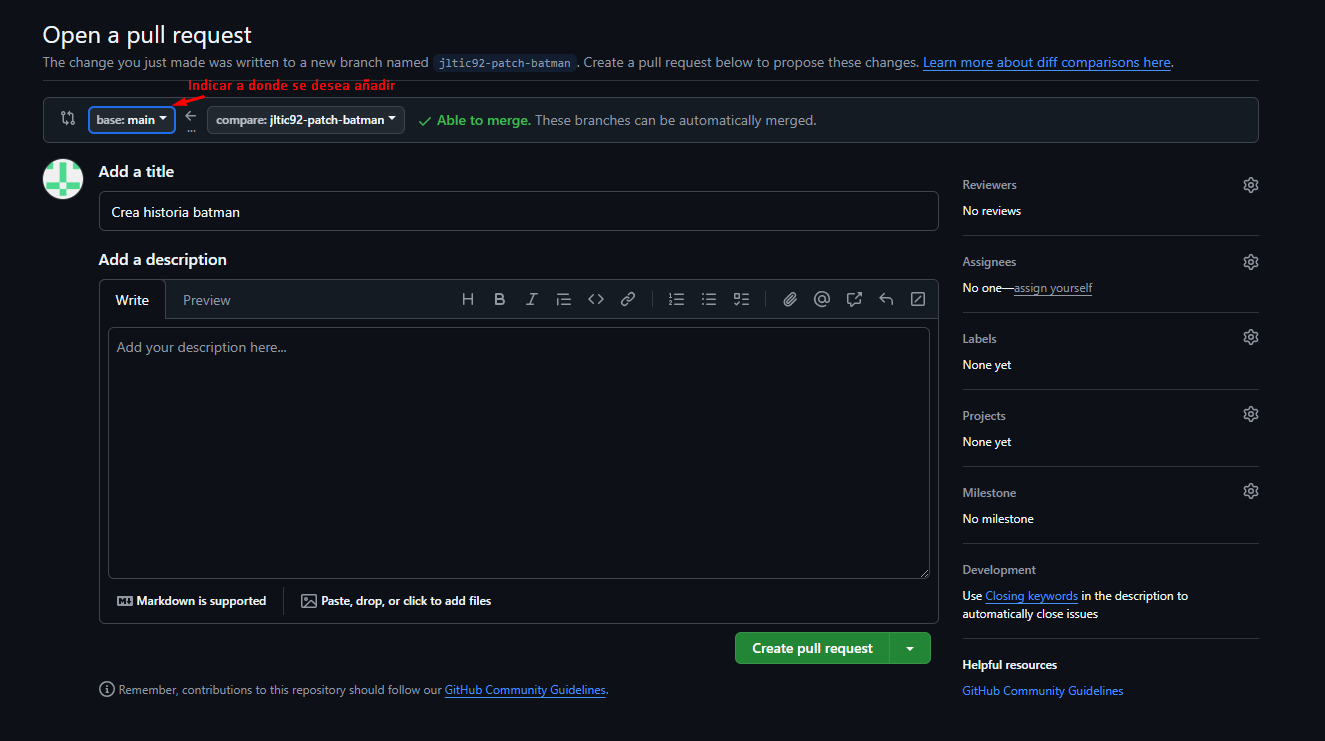
<https://docs.github.com/es>

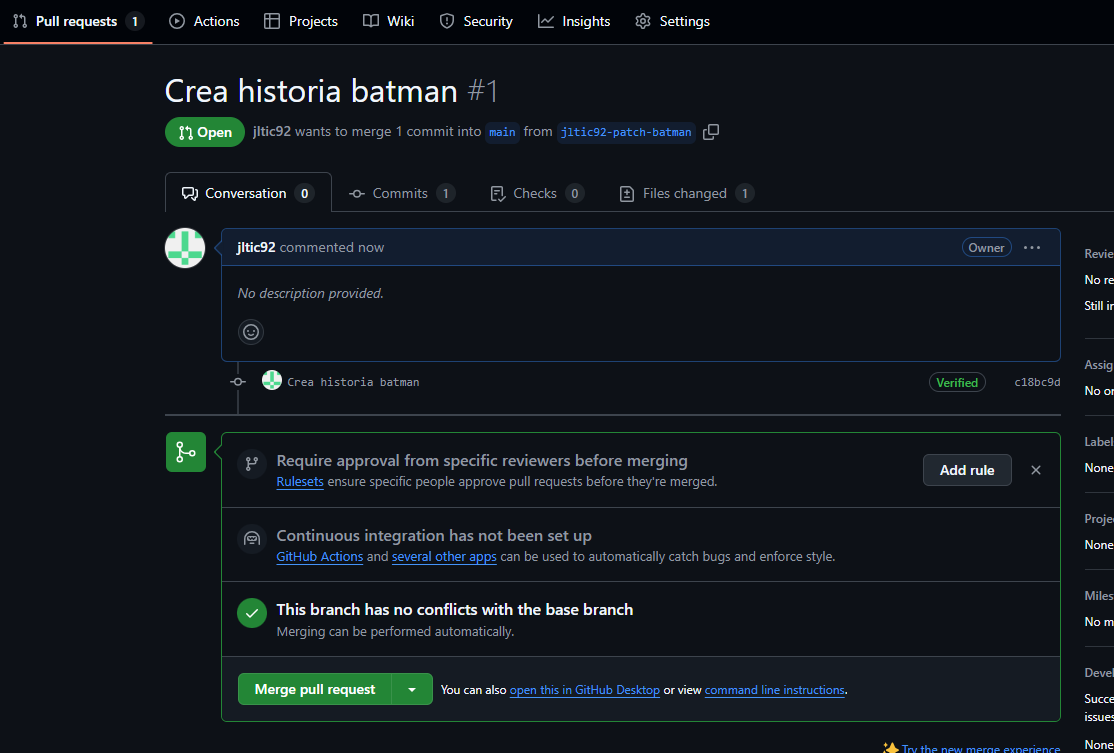
**Pull request**

Se puede hacer desde Github o desde el editor de código

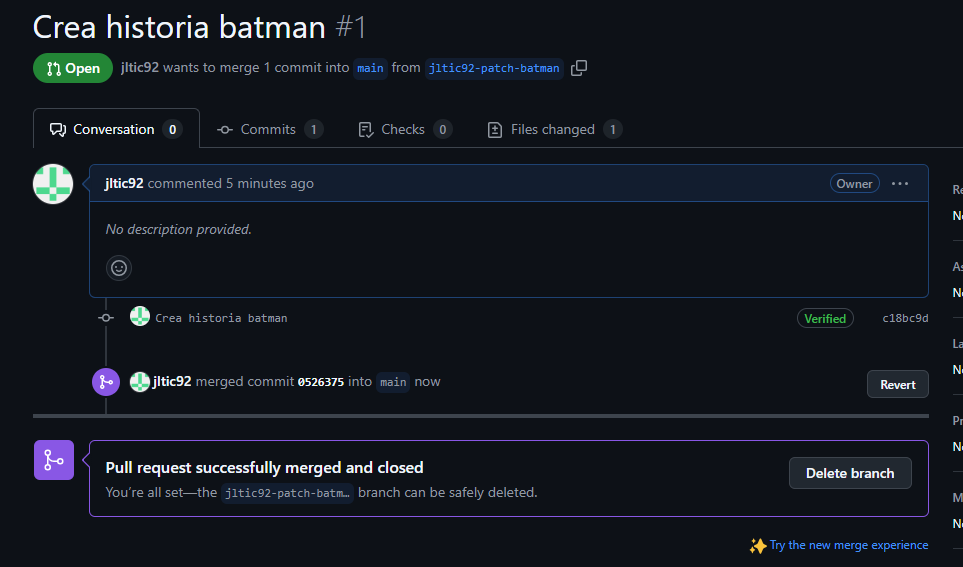
Desde git hub se debe de indicar la rama







Cuando se le da clic en merge aparecerá el mensaje si se tiene algún conflicto en el merge, de lo contrario indicará que se completó y dará opción para borrar la rama creada. Si se borra por error te da la opción para restaurar la rama.



Después de hacer un pull request te asigna un identificador sobre el cual puedes ver los commits y varias acciones relacionadas.

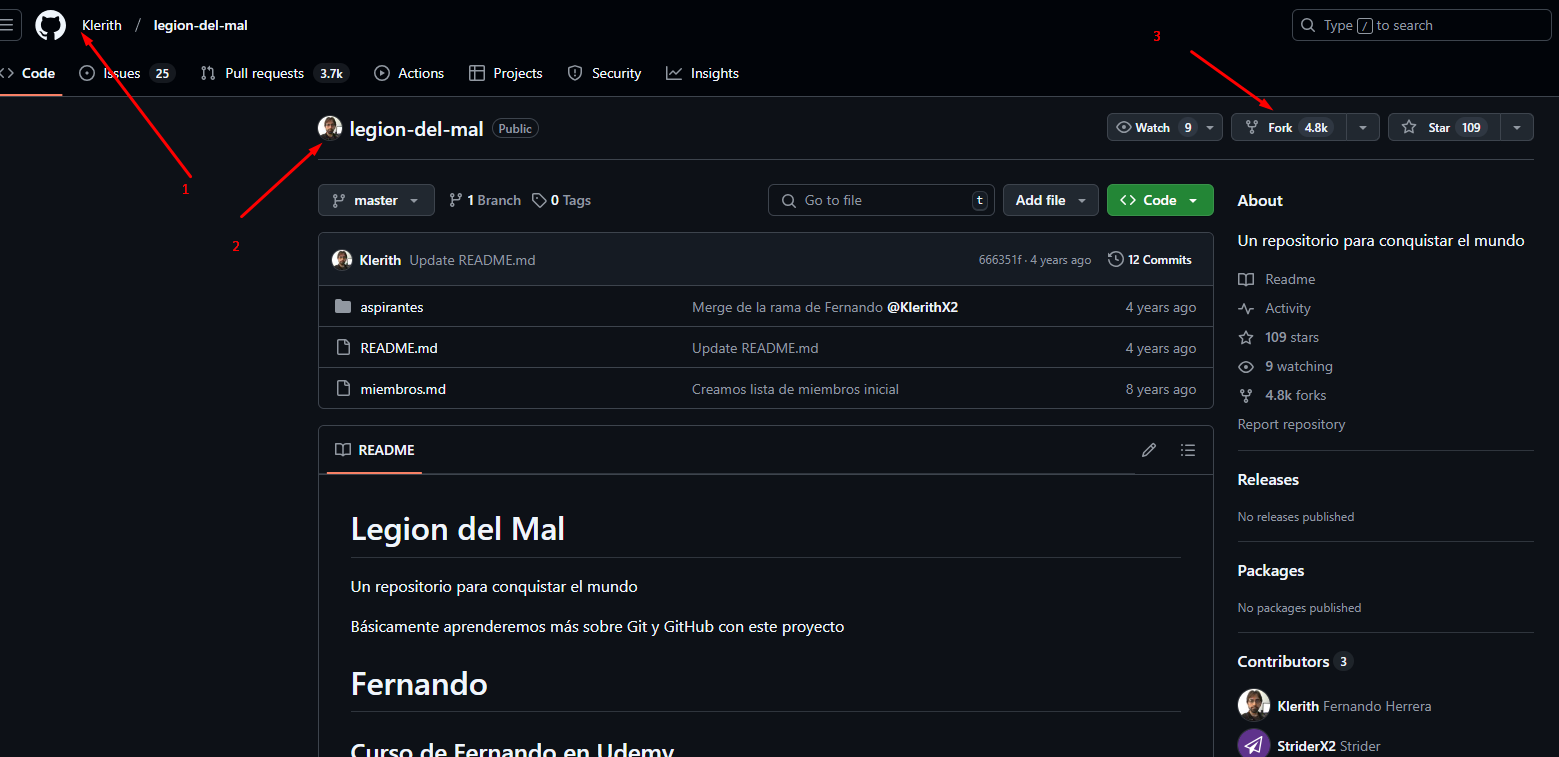
**Documentación pull request**

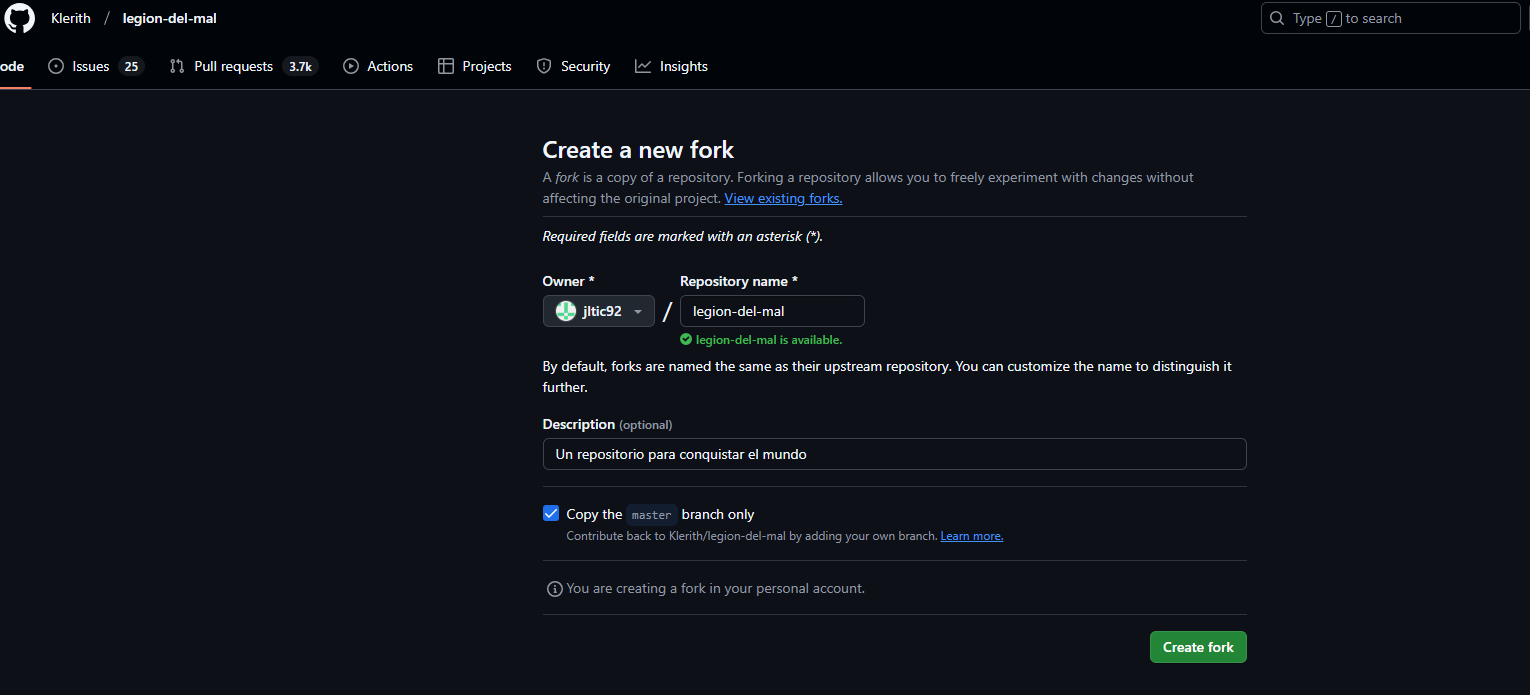
<https://blog.mergify.com/understanding-the-github-pull-request-workflow/>

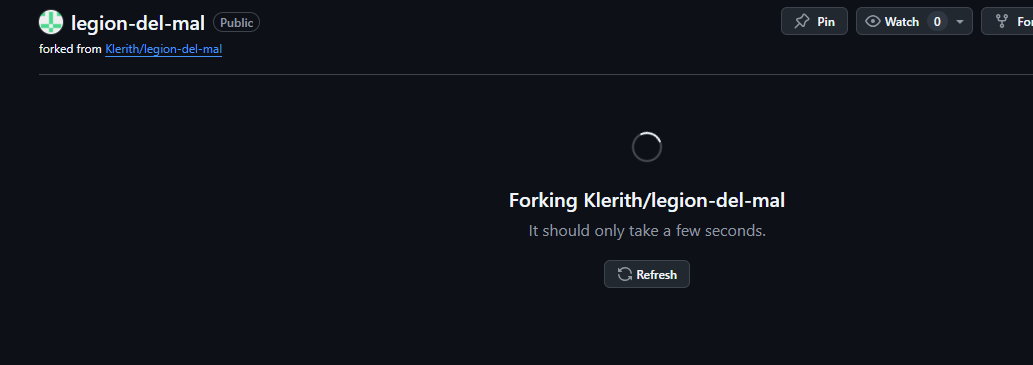
**Git fetch =>** obtiene los commits faltantes para completar la historia.

**FORK**

Cuando no tenemos acceso a un repositorio podemos hacer un fork y eso es copiar el repositorio a nuestra cuenta de GIT HUB donde tenemos los permisos completos.



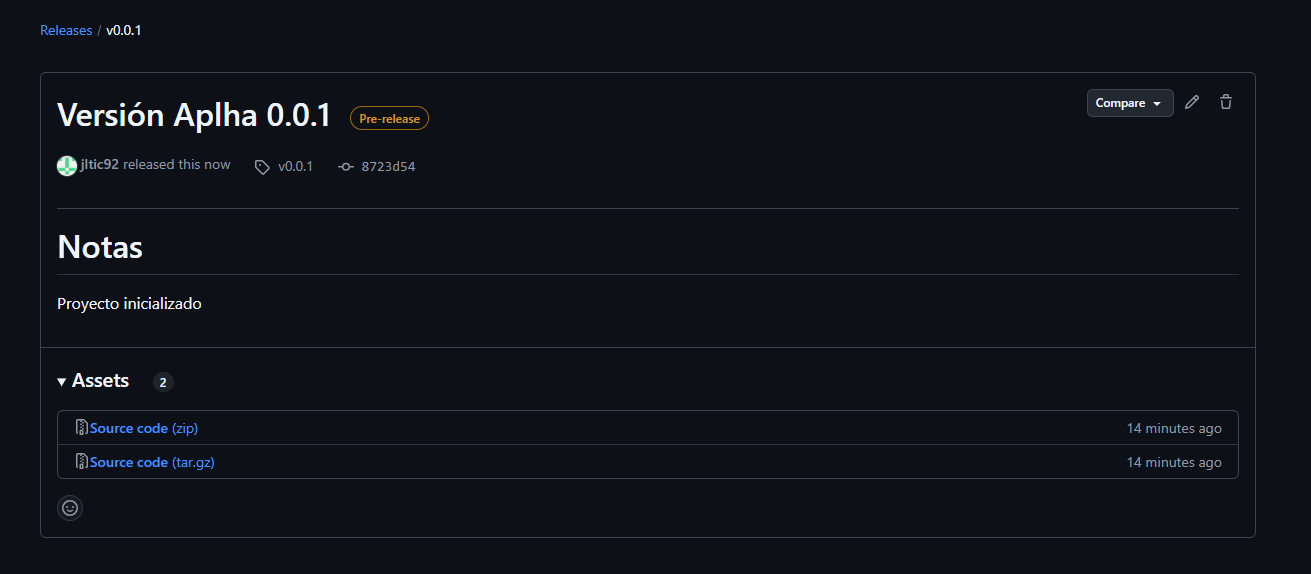




Cuando se trabaja con forks se tiene que usar upstream para subir los cambios y obtener los de otros que también hicieron fork.

Cuando de un repositorio remoto del proyecto se hacen varias ramas por cada desarrollador, a cada uno se le conoce como **feature Branch**, en cada una de ellas se estarán trabajando nuevas características que después serán incluidas al master.

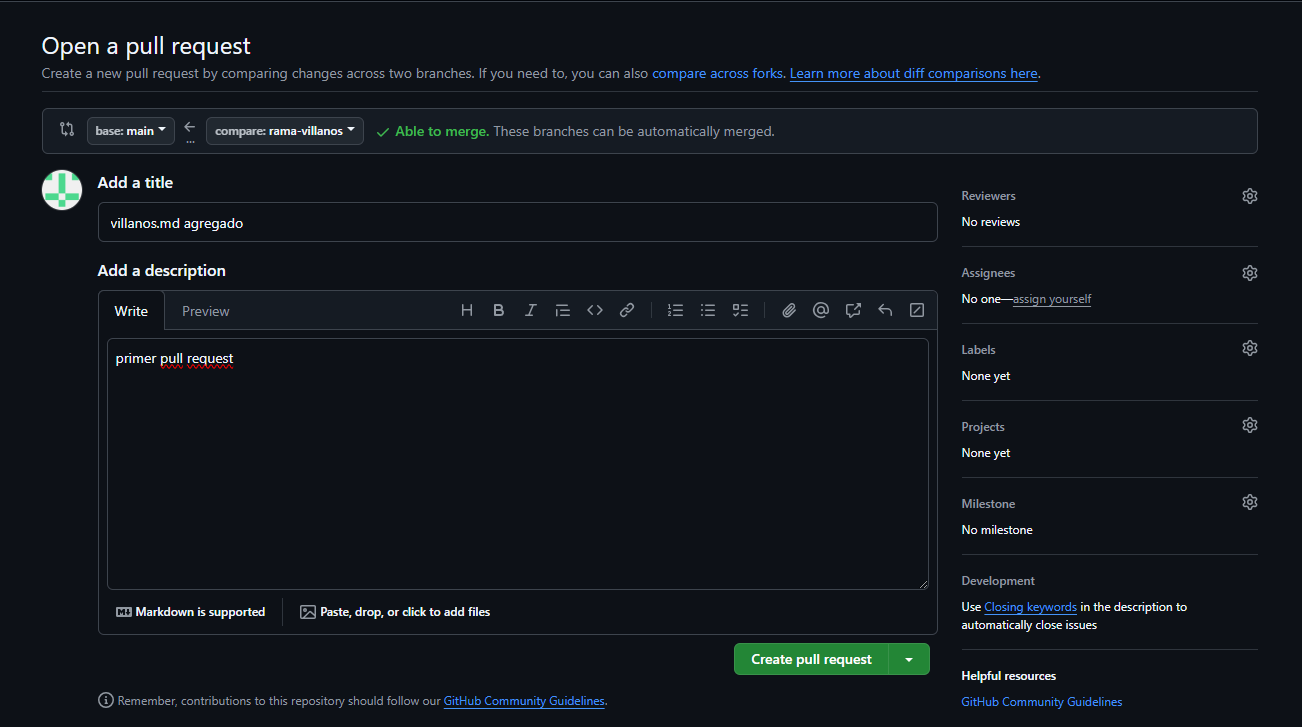




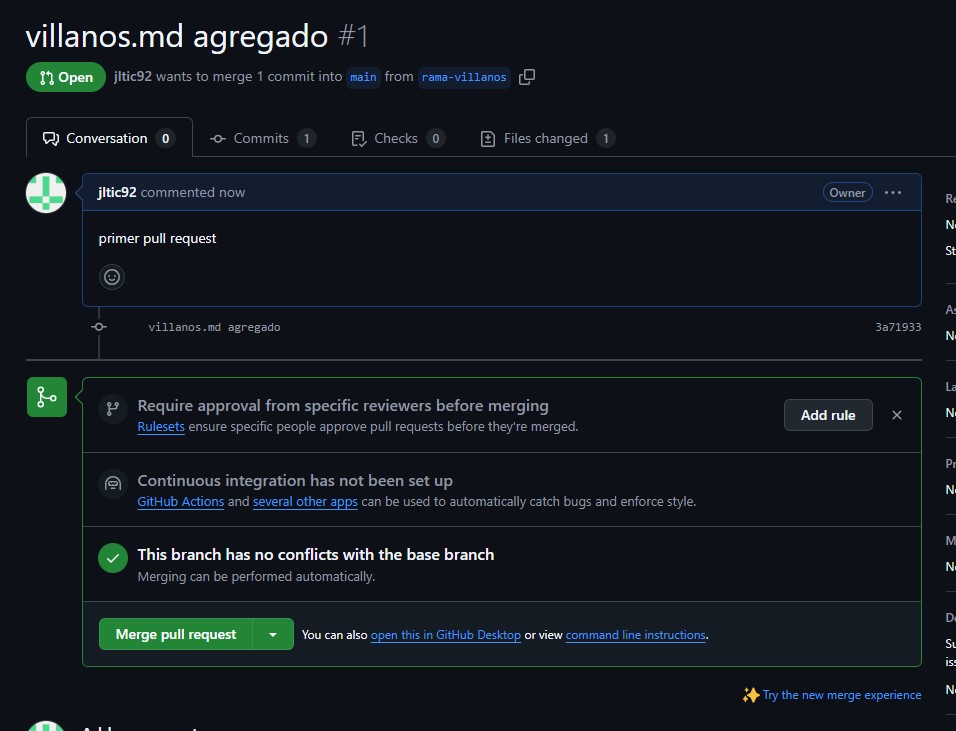
**git checkout -b rama-villanos** => Cuando se crea una rama

**git push --set-upstream origin rama-villanos** => Subir al repositorio remoto la nueva rama con el commit elaborado previamente

Solicitará 1 pull request indicando la rama a la que queremos incorporar los cambios

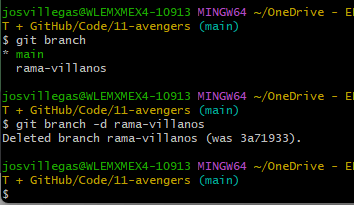


Valida si los cambios se pueden combinar correctamente



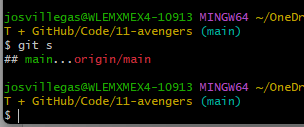
Cuando se generó el pull request pero todavía no se aprueba el merge o el pull, cada commit que se haga para la misma rama-villanos se agregará al pull request y también será autorizado en caso de aplicar.

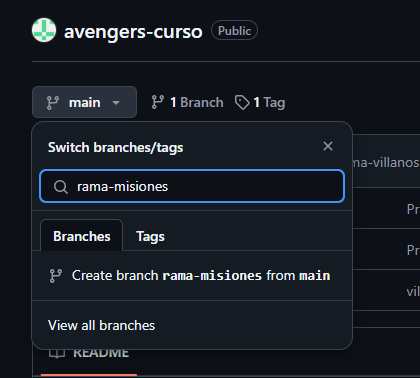
**git Branch -d rama-villanos** => Se puede borrar la rama después de hacer el merge y, si lo hacemos, tenemos que eliminar la rama de forma local



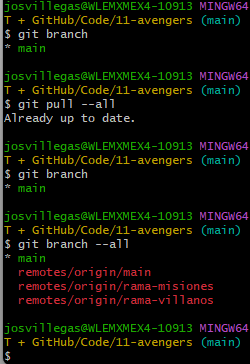
git status -sb => muestra la rama en la que estoy, cómo se llama la rama origen y si tenemos archivos pendientes de commit

git config --global -e => Para modificar un alias (dentro de presiona a para insertar)

Esc => :wq!

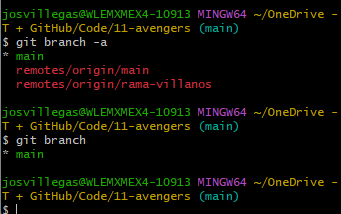
Para crear una nueva rama podemos hacerlo desde GitHub, dando clic en la búsqueda, colocar el nombre que queremos y GitHub mostrará la opción para crear la rama desde main

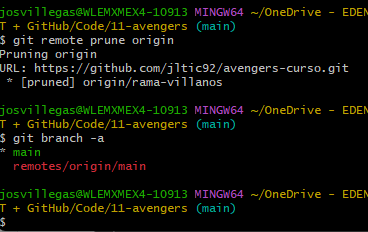


Si al momento de hacer un git pull no se trae la información de una rama creada, se tiene que hacer un git pull –all, en caso de que al hacer git branch no aparezca la nueva rama, tengo que ejecutar el comando git branch –all

Para entrar a la rama se coloca **git checkout rama-misiones**

**git push origin :rama-misiones** => borrar rama desde git bash incluida remoto

Cuando se elimina una rama en git hub y quieres ejecutar el comando anterior se da un error de que la rama ya no existe en el remoto pero sigue apareciendo si ejecutas el comando git branch -a para eliminarla se ejecuta el comando



**git remote prune origin**

Los tags ayudan mucho para mantener las versiones compilados, al tener un problema de que se borró una rama, al regresar al tag se puede subir nuevamente la rama con git checkout -b [nombre de rama], hacemos git push y ya se sube al remoto.

Git reflog => muestra los commits que ha tenido la rama

**Sección 9 milestone**

Los milestone son las agrupaciones de issues de un proyecto. Una documentación es la siguiente:

<https://docs.github.com/es/issues/tracking-your-work-with-issues/about-issues>

Un issue son las solicitudes de cambios por medio de Git Hub

Colaboradores son aquellas cuentas que pueden ver, editar y eliminar cosas del proyecto ya que tienen permisos elevados.

Los issue se pueden cerrar directo desde Git Hub haciendo referencia al issue y dando clic en cerrar, o bien, se puede cerrar desde un commit usando el ID del issue, por ejemplo:

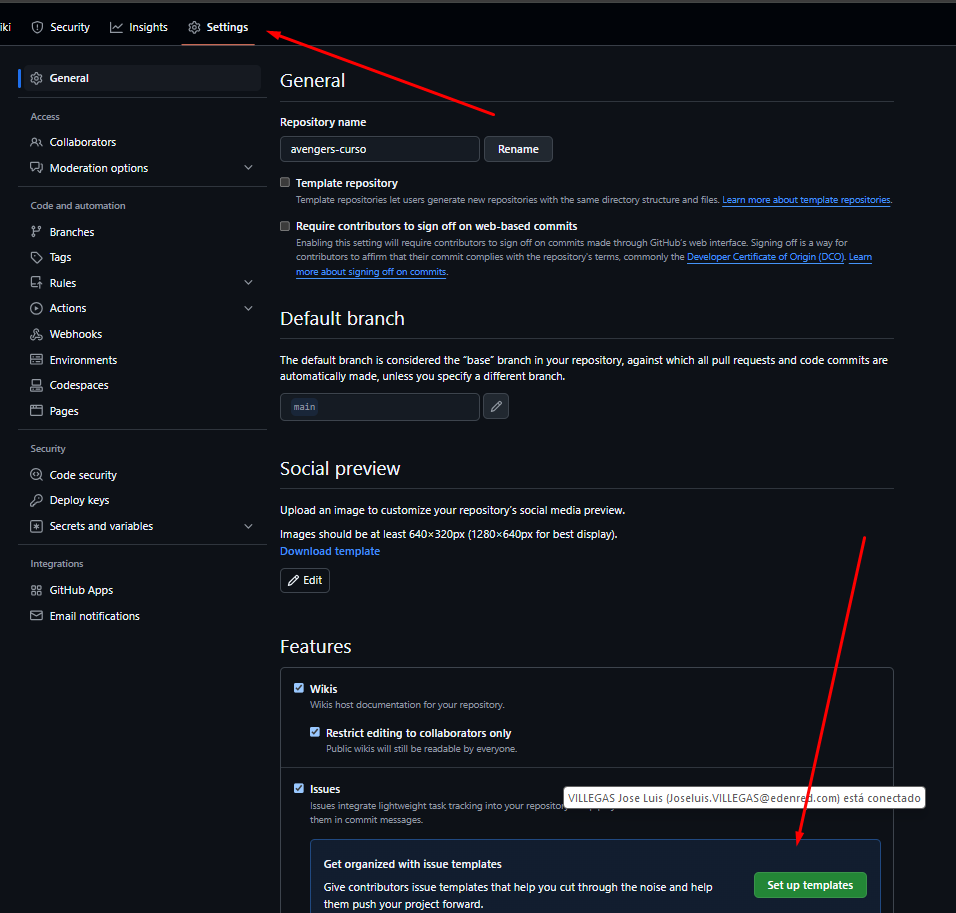
git commit -am “Fixed #5: se arregló”

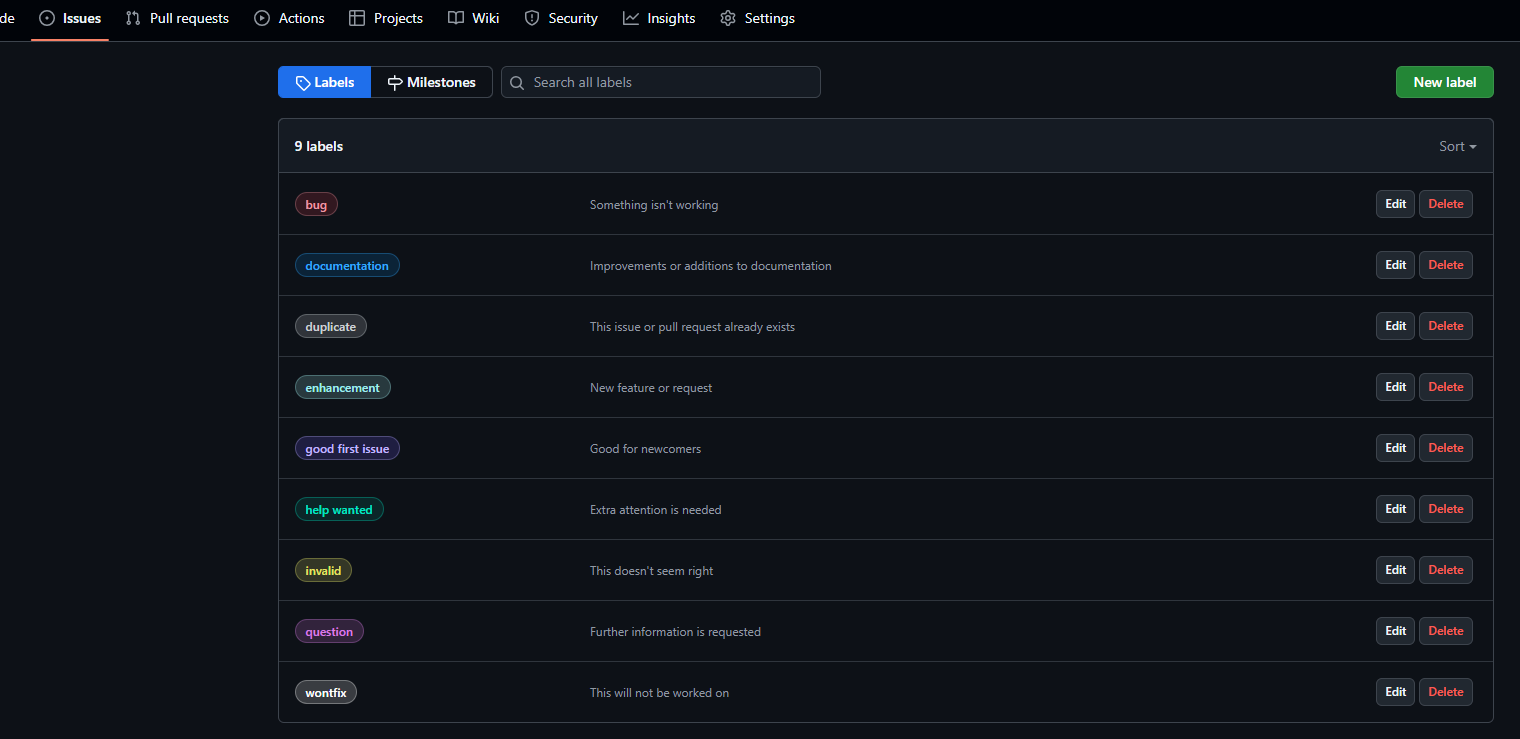
Git hub lo que hará será considera el ID del issue para cerrarlo por que se atendió la actividad.

Se pueden establecer templates para issues para que pueda parecer como un reporte.

Las etiquetas sirven para categorizar cada issue como bug, feature, entre otros.

En la pestaña de settings se pueden agregar colaboradores





**Sección 10**

Wiki: Es un espacio que sirve para describir el proyecto

Se pueden crear proyectos para seguimiento de las actividades

**Sección 11 organizaciones y equipos**



Administra N cantidad de repositorios que todos los miembros de la organización tendrán acceso. Todo que suban en la organización se va a compartir.

Si te añaden a una organización, tus repositorios privados no se añaden, a menos que tu los migres a la organización.

Se pueden crear equipos de trabajo que pueden verse como mini organizaciones, entre cada equipo se pueden compartir repositorios o dar seguimiento.