**GIT**

<https://git-scm.com/>

Repositorio distribuido: cada persona tiene una copia en su local + la rama del servidor central

La configuración global de git se encuentra en C:\Users\IdUser\.gitconfig

# Comandos básicos:

* git –version
* git help **sale todo si pone git help init solo la info del init (misma info que en la web). Para salir q**
* git init **inicializa un repositorio LOCAL git (se crea la carpeta oculta .git de tu proyecto). Hay que usar** git add **para definir qué archivo hago trazabilidad**
* git –help config
* git config –global [user.name](http://user.name) “ana” **(si no existia .gitconfig global en la carpeta de usuario te lo crea)**
* git config –global -e **lo editas en la consola, click en a, editas luego esc + : + wq! para escribir y salir o q! para salir solo**
* git status **te dice el estado del repositorio y en que rama estas**
* git status –short **me da un resumen del status**

## git log

* + git log –oneline **me da un resumen de los commit**
  + **Para ver el log de commit con fecha y autor resumido:**

git log --graph --decorate --all --pretty=format:"%C(yellow)%h %Cgreen%ad %Cblue%an%Creset %s %C(red)%d"

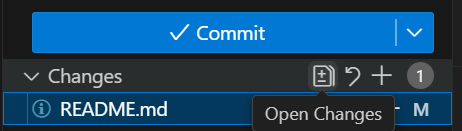
* git add . **añade todos los archivos para que se trace su estado desde el principio y en cada commit. Si no lo alades se ve untracked:**

**sino A**

* **para quitarlo** git reset “nombre del archivo”
* git commit -m “comentario del commit” **(Pull request inicial de la rama master/main)**
* git checkout – . **para recuperar todo el repositorio de acuerdo al último commit con los archivo con seguimiento/trazados en el git status**
* git branch **para saber en qué rama estas**
* git branch -m master main **cambia el nombre de master a main**
* git config –global init.defaultBranch main **para que cada vez que se cree un repositorio la rama master siempre se llame main**
* git commit -am “readme actualizado” **si el archivo está trazado este comando hace add y commit (siempre hay que hacer add ante cualquier modificación)**
* git log **para ver el log del repositorio (los commits etc)**
* **archivos modificados para comitear**

**D eliminado, A añadido a ser trazado, N new**

* **sobre todos los changes puedes hacer open changes para comparar los cambios o abrir los archivos desde la vista source control**

**la flecha es restaurar antes del cambio, + añadirlo al status=trazarlo (commit) puede ir a nivel de carpeta tb. Si quieres hacerlo por consola es** git diff

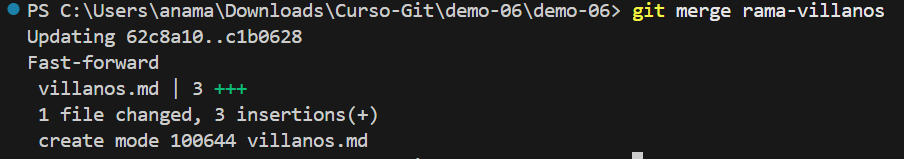
* **sobre cada archivo tienes open file y te resalta las líneas cambiadas (sin comparar con el original)**

****

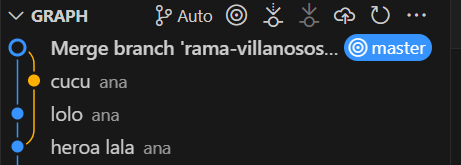
* git commit js/\*.js -m “comentario del commit” **commits por archivos OJO tienen que haber sido añadidos primero para trazarlos y, si la carpetas estan vacias no les hace seguimiento**
* **git commit –amend -m “Instalaciones actualizadas”** para corregir el mensaje del último commit

# Gestión de ramas en repositorio LOCAL:

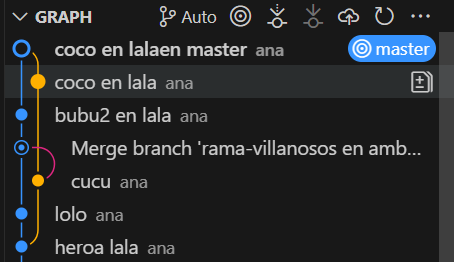
* git branch nombreRama
* git checkout “nombre de la rama a donde quiero ir”
* **para crear rama y cambiar directamente a ella en local:** git checkout -b rama-villanos
* git add . #para añadir a la rama local trazabilidad de todos los archivos en su pila de status
* Merge entre ramas:
  + git merge rama-villanos **estando en la master**
  + Hay 3 tipos
    - fast-forward: cuando git detecta el cambio de la rama que va por delante de la rama en la que estás y lo mergea solo



* + - Unión automática: 2 ramas avanzan y se mergean pero en cosas diferentes asi que git puede hacerlo automáticamente



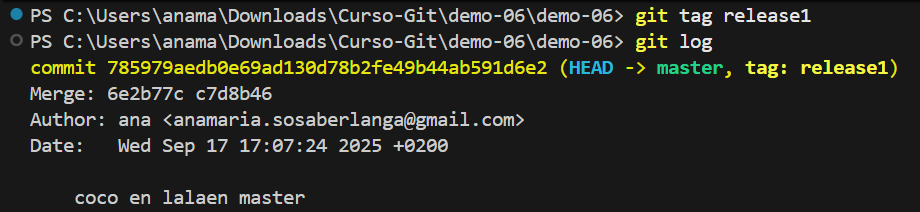
* + - Conflictos



* + git branch -d rama-villanos **para eliminar la rama en local que ya no usas si pones -f al final lo fuerzas así git no te pregunta**

# Tags para releases y hotchanges

Los tags sirven para crear versiones/releases, subir hotchanges directos producción y/o rescatar ramas de versiones estables, haciendo referencia a un commit y por tanto a un estado concreto del código: git tag nombreTag o **git tag -a nombreTag** (esta guarda más información→ mejor para releases)



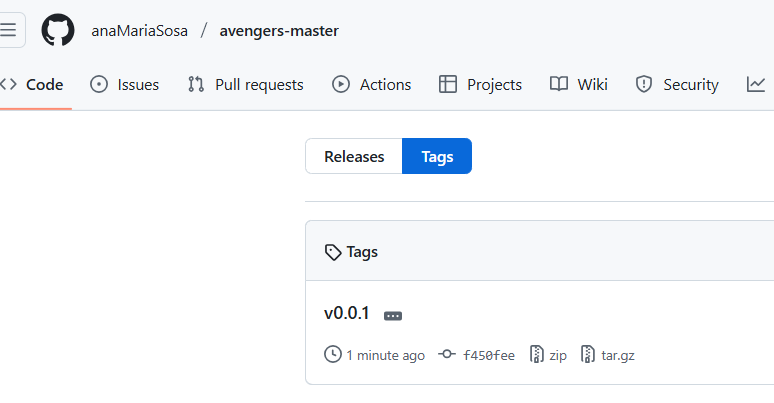
con git tag te saca el histórico de tags y con git tag -a release0.5 codgiHashdelcommit -m “explicacion” para definir release en un commi que no es el último.

Ejemplo:

git tag -a v0.0.1 -m “primera version”

git push –tags #para subir los cambios correspondientes al tag

Si vas a github aparece un tag con el detalle del autor y demás



Para hacerlo release candidate cambia a release, rellena la info, si deseas que sea release candidate haz check en pre-release sino no y publicala

* Rebase: tengo 2 ramas, una es la master, quiero mergear en la secundaria con diferentes cambios que no se solapan→ rebase tal que:
  + git checkout ramasecundaria
  + git rebase master

Esto me actualiza mi rama con la master y luego realiza los últimos commits de mi rama (hay más casuísticas y tipos de rebase pero es mejor hacer merge de conflictos o cherry picks)

Observación: si has borrado la rama, el tag y demás con el histórico y cherry pick puedes recuperar la rams pérdidas

# **Repositorio en remoto → Github**

* Git control de versiones local que se integra con PaaS como Github (de Microsoft) para tener repositorios en remoto/cloud. Git NO maneja el acceso al repositorio, lo hace github
* Github usa el lenguaje de etiquetado markdown (.md)
* Repositorio tiene n proyectos
* insights KPIs

<https://github.com/>

Otros ejemplo como github: **bitbucket** (integrado con jira)

* Cuando queremos agregar un origen remoto (que el repositorio no este en local en la carpeta en la que hacías git init): git remote add nombreDelRepoRemoto(suele ser origin) rutaURL(del github). Para saber cuando repositorios remotos tenemos: git remote -v
* Cuando queremos subir por primera vez nuestro código al repositorio remoto: git push -u origin master Con la “-u” definimos que por defecto cada push que se haga se hará desde la rama master hacia el repositorio remoto. Me pedirá credenciales siempre. Para cachearlas y que no me las pida más seguir: <https://docs.github.com/es/get-started/git-basics/caching-your-github-credentials-in-git#platform-windows>
* cuando creas un repositorio desde github te aparecen dos secciones de código, una es cuando no tienes código todavía y la otra era para enlazar/subir el código existente a tu repositorio remoto en github. Ejemplo de la segunda opción:

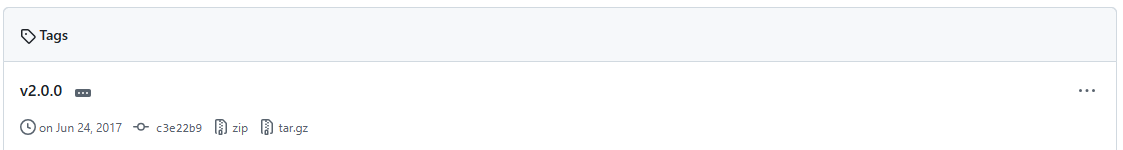
**git remote add origin https://github.com/anaMariaSosa/liga-justicia.git**

**git branch -M main**

**git push -u origin main**

Una vez ejecutas esto en tu repositorio remoto vacío aparece el código existente en tu local y a partir de ahí todos los push que hagas se subirán a tu repositorio remoto origen main

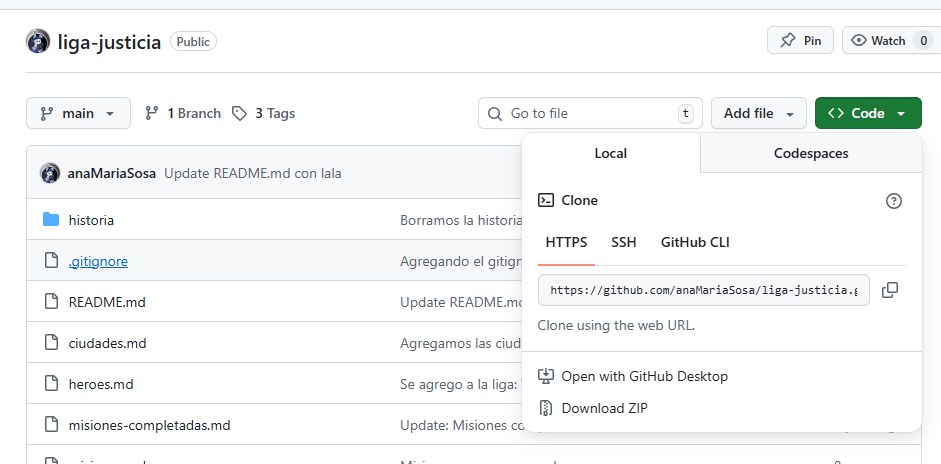
* **git push –tags** para subir a tu repo remoto los tags(versiones/release) que tuvieras. Una vez realizado en github en tu repo en las tags veras por ejemplo:



OJO el .zip te descarga el código en ese punto en el que creaste el tag pero NO es un repositorio git (no tiene la carpeta oculta .git), tendrias que inicializarlo, para tener el .git está el clone

OJO: crear releases con tags, puedes incluso definir pre-release tb

* **git pull** traerte todo desde el repositorio remoto (por defecto origin)
* **git clone rutaCopiadadeGitHUb**



OJO se clona EN TU CARPETA DE REPOSITORIO LOCAL lo que está siendo trazado (los add .) lo que no se han añadido o esté en ignore no se incluye. Ver más detalle en clone mas adelante

# Gestión de commit y diferencias entre ramas

* si haces un cambio desde el visualStudioCode, en tu código del repositorio y, haces un commit (git commit -am “mensaje para el PR”) se sube en tu local pero no al repositorio remoto (el github) para subir al repositorio remoto hacemos **git push** (por defecto a origin)





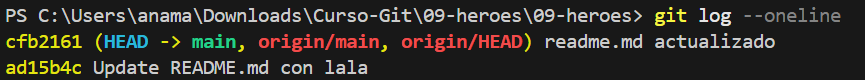
HEAD=la rama en la que estás en local

origin/HEAD = referencia a la rama “copia” de la que estás en el repo remoto (puede no existir aún como tal, aun así el origin/HEAD se considera asumiendo que lo vas a crear en algún momento)

origin/main = referencia a la rama “copia” de main en el repositorio remoto

main=main local

después del push

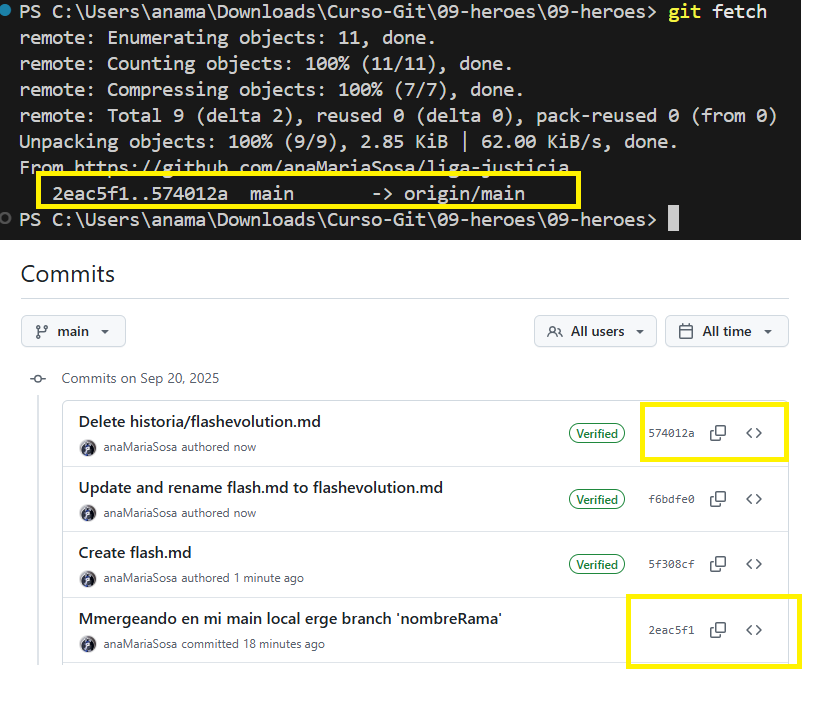


NUNCA hagas push antes de pull en tu rama secundaria

OJO: PR= pull request es más de github que de git ya que no solo es subir los cambios también se analiza que pasa ante merge entre ramas.

* squash de pr = se mergean el ultimo pr y este como si no existiera el último realizado control de issues
* **git fetch** #antes de hacer un pull a veces te interesa ver qué es lo que NO tienes, para ello está el fetch que te baja commits, etiquetas y ramas para que verifiques lo que hay en main antes de actualizarte con pull

Ejemplo hice cambio en mi repo remoto que en local no tengo, cuando hago en VSCode git fetch me lo indica:



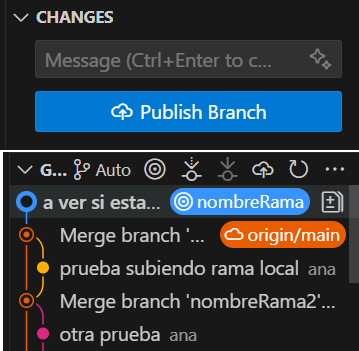
si haces git pull se mergea mi local con lo que me traigo del repo remoto

# Gestión de ramas entre repositorio LOCAL y REMOTO:

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

* + Recuerda:
    - **git branch nombreRama** #crear rama en local basada en la de por defecto que es main
    - git checkout “nombre de la rama a donde quiero ir” (tambien puedes usar git switch)
    - para crear y cambiar de rama a la vez git branch -b nombreRama o **git switch -c nombreRama**
    - **git branch** para verificar en qué rama estas
    - **haz cambios**
    - **git add .** # una vez creada la rama añadele los archivos a trazar en el git status para que pueda hacer commit y seguimiento
    - **commit**, push y pull SIEMPRE se hacen en la rama en la que estés.
      * **git commit -m “mensaje** ”#desde la rama 2
      * **git log –oneline** # veras que rama2 esta 1 commit por delante de main local y main repo remoto. Si haces el switch a la main y hace el git log, al estar en main te aparece que estás alineado main local y main repo remoto

Crear tu rama en el repositorio remoto **git push -u origin nombreRama** (con el VSCode después del commit te indica que publiques la rama)



Una vez creada origin/HEAD, sigue los pasos:

# Estás en rama2

#veo el estado entre local y origin= al repositorio remoto (github) sin mergear

#si no devuelve nada ambas ramas, en el ejemplo main y nombreRama estan alineadas

**git fetch origin**

#hago el merge del origin/main a mi rama local main, si hay alguna diferencia, sino te dire Up to date

**git merge origin/main**

# Resuelve conflictos si aparecen, luego:

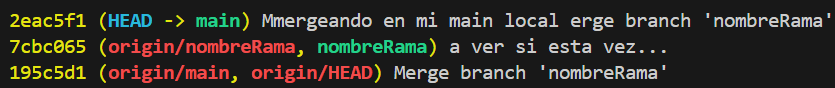
**git push**

**git switch main**

**git pull**  # trae cambios del main remoto al main local por si acaso

**git merge --no-ff rama2**  # crea un merge commit con la main local

#en este momento, estoy en main local, en remoto nombreRama está alineada me falta main remoto



# Si hay conflictos resuelvelos y **git commit**  #te dira que tu main local ahora está por #delante de tu main remoto

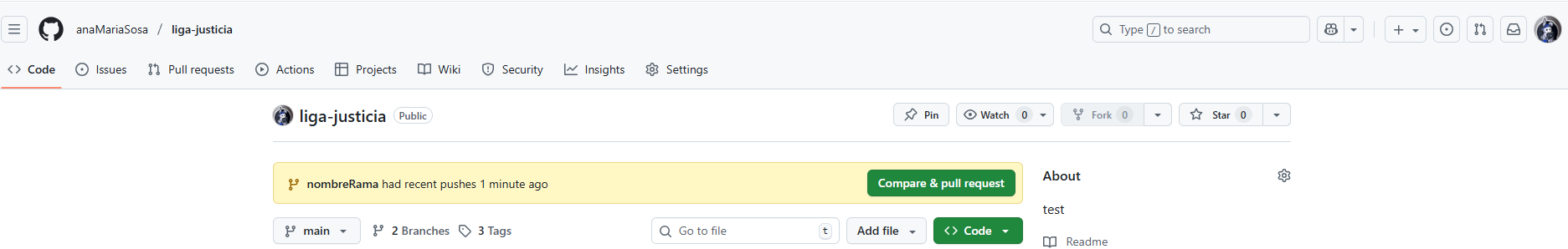
# en este punto solo falta actualizar main del repo remoto

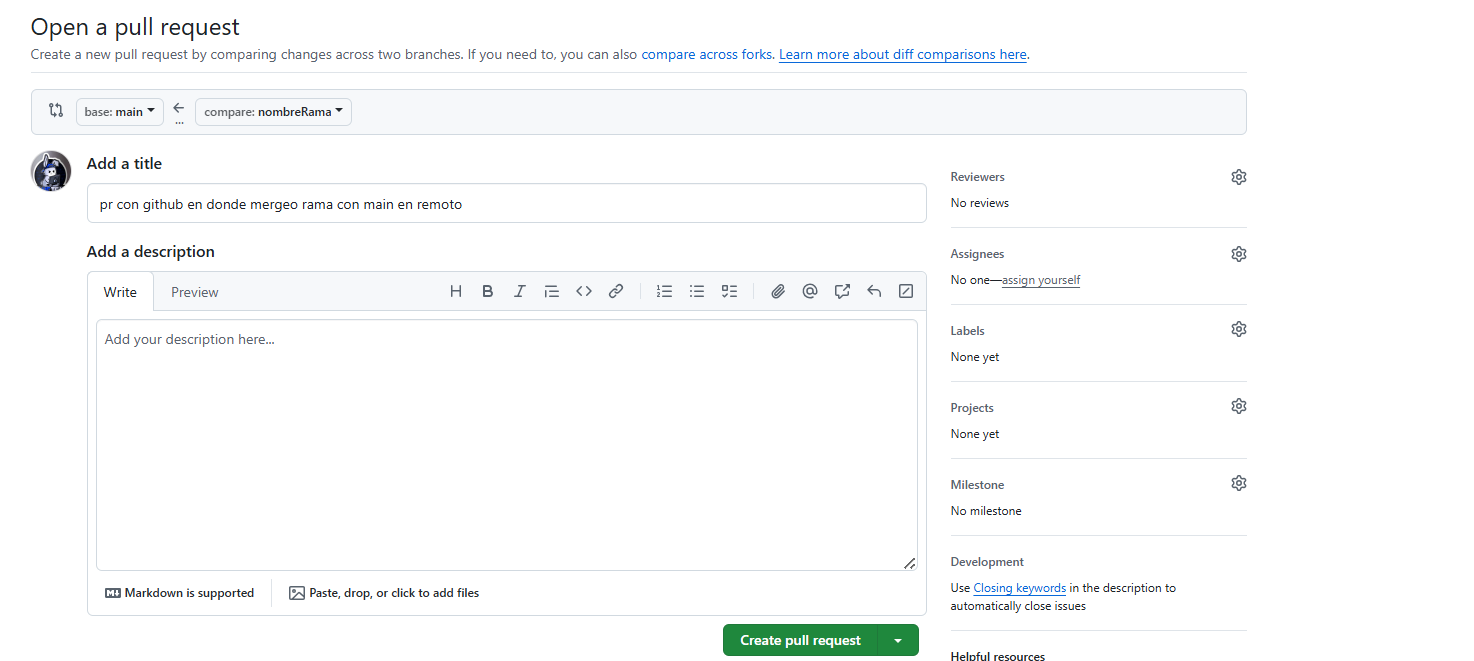
**git push** #subes a main repo remoto

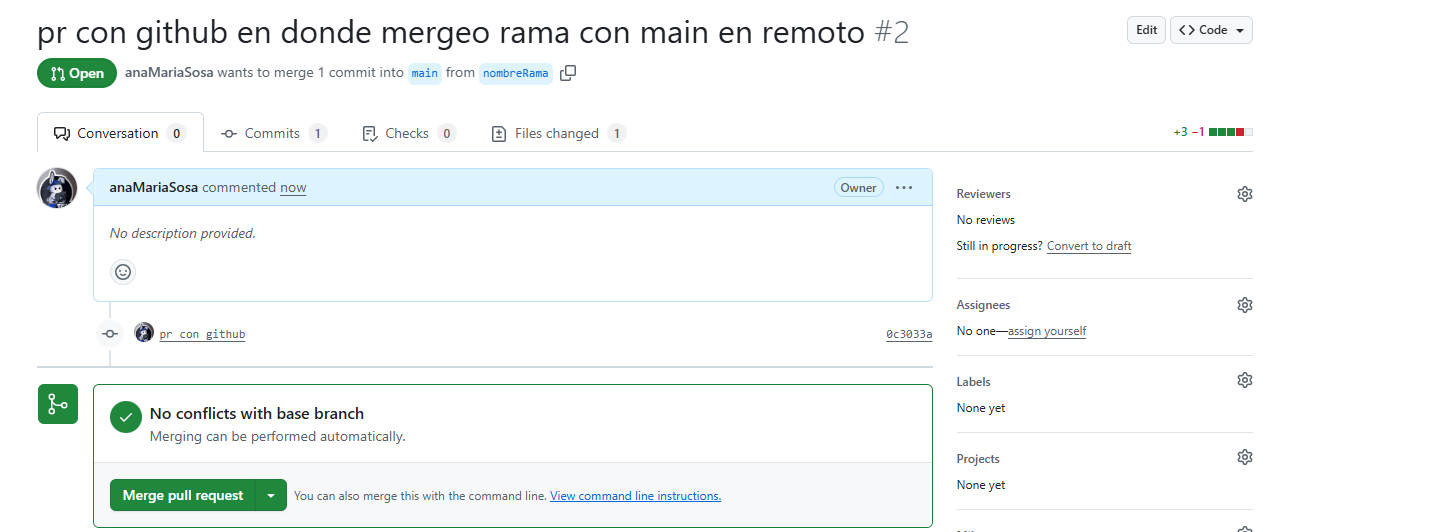
**git branch -d rama2** # borra la rama local

**git push origin --delete rama2**  # borra la rama remota

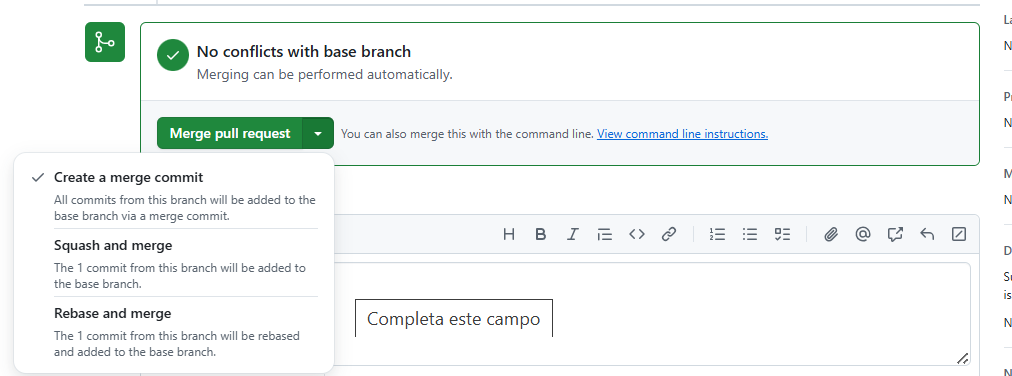
OJO: esto solo es posible si no tienes branch protection en la rama main. Si lo tienes hay que mergear con un PR desde github: repite los pasos anteriores hasta el momento de crear tu rama en github (**git push -u origin nombreRama**). Ahora el merge lo hacemos desde github con pullRequest(PR)

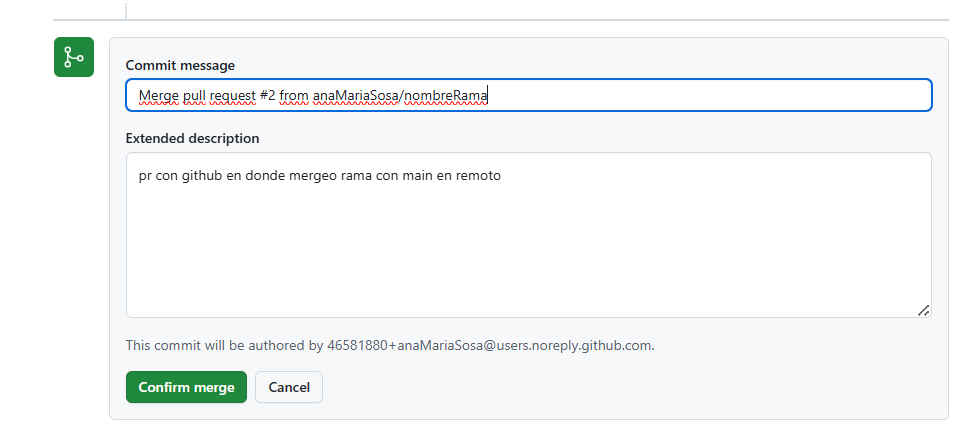




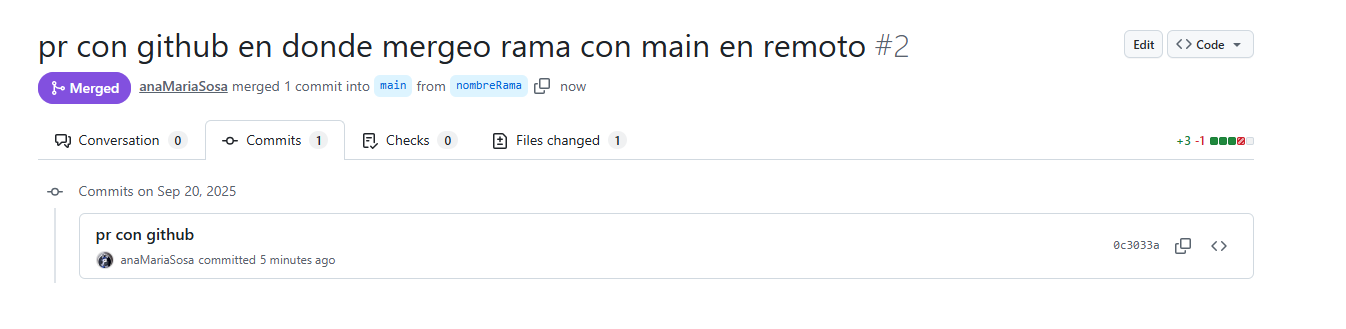


si hay check es que hay conflictos, los resuelves a mano en la pestaña de checks





te vas a pullrequest y ves el histórico y los cambios concretos si quieres en ficles changes:

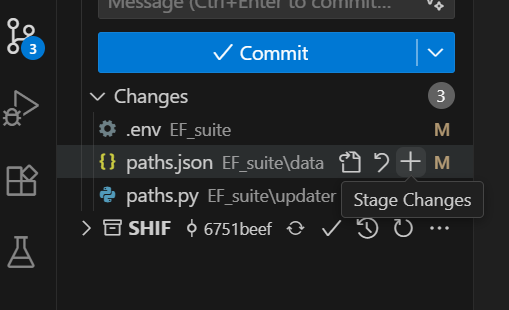


en este punto main y nonbreRama en remoto están merengadas y alineadas, al igual que nombrerama local, solo falta main local, para ello:

* git switch main
* git pull origin main #esto hace fetch y merge a la vez. OJO: si quisiera bajarme a mi main local cualquier otra rama de origin sería git pull origin nombreRama. Si tienes por defecto main en tu gitconfig gracias a **git push -u origin main** con hacer git pull vale

## OJOO--Stage Changes para commit

OJO: recuerda seleccionar lso archivos que quieres commitear:



## Crear Rama en local a partir de otra rama en remoto

Una vez clonado tu repo

Git fetch origin

git switch otra-rama-origin

git pull

git switch -c mi-nueva-rama

en una sola línea: git switch -c mi-nueva-rama origin/otra-rama-origen

bajarte cambios de remoto a local de una rama explicita: git pull origin otra-rama-origen

## Revisando ramas y código de los demás

git pull –all

git branch #solo salen las mías aunque me haya hecho el pull de todas las del repo remoto

git branch -a #veo las ramas de los demás que tengo con el pull all

#si hago switch a una rama que no es mia me la actualiza como que si que es mia y ya el git #branch me la muestra

y ya gestionas los PR desde github

# Revert, reset y cherry picks:

* Cherry picks: **para seleccionar un commit concreto a mergear:** git cherry-pick hashdelcommit. En github no hay boton como tal habría que usar github desktop
* Reset: suele usarse solo en local
  + consola: ejemplo volver al estado en remoto descartando cambios locales:

**git fetch origin**

**git reset --hard origin/main**

* + github: no hay opción como tal puedes crear una PR revirtiendo los cambios (un revert)
* Revert
  + Consola:

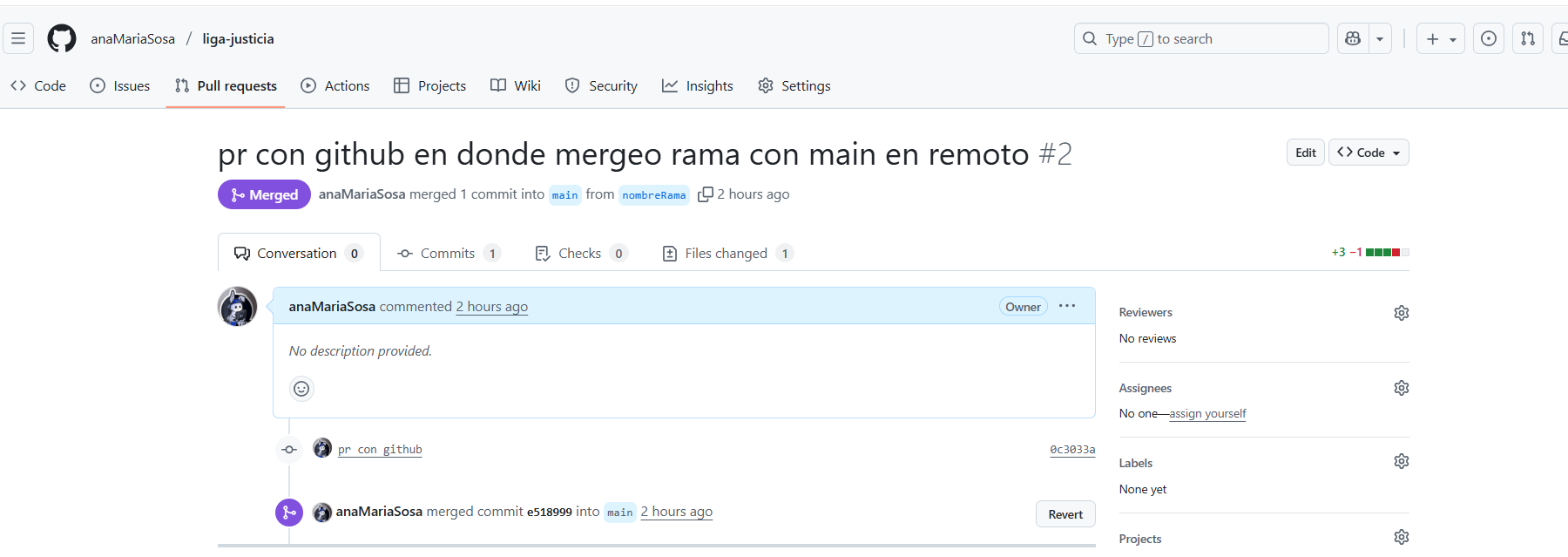
# revertir varios (uno a uno o como rango con -n para acumular y luego confirmar)

git revert -n abc1234 def5678

git commit -m "Revert: xyz"

git push

* + Github: hay botón revert para crear una PR



# Fork y Clone:

* Clone cuando somos “dueños/colaborador del repositorio. git clone y git pull/push para bajarte y subir cambios
* Cuando no eres dueño/colaborador del repositorio se suele usar fork ya que si haces clone te lo bajas pero tus cambios no los vas a poder subir. El fork te clona el repositorio en tu cuenta de github, manteniendo la referencia a su repositorio original, una vez clonado en nuestra cuenta podemos hacer todas las operaciones. Para que los cambios que hacemos en nuestro fork los vean los del repositorio original, hay que manejar PR en github. Observación: En github cuando NO eres colaborador, la opción de setting no aparece por ejemplo
  + Desde gitHub puedes ir a cualquier repo público y hacerte un fork a tu repo remoto
  + Luego con clone te lo creas en tu local

## Bajarme actualización del repositorio original

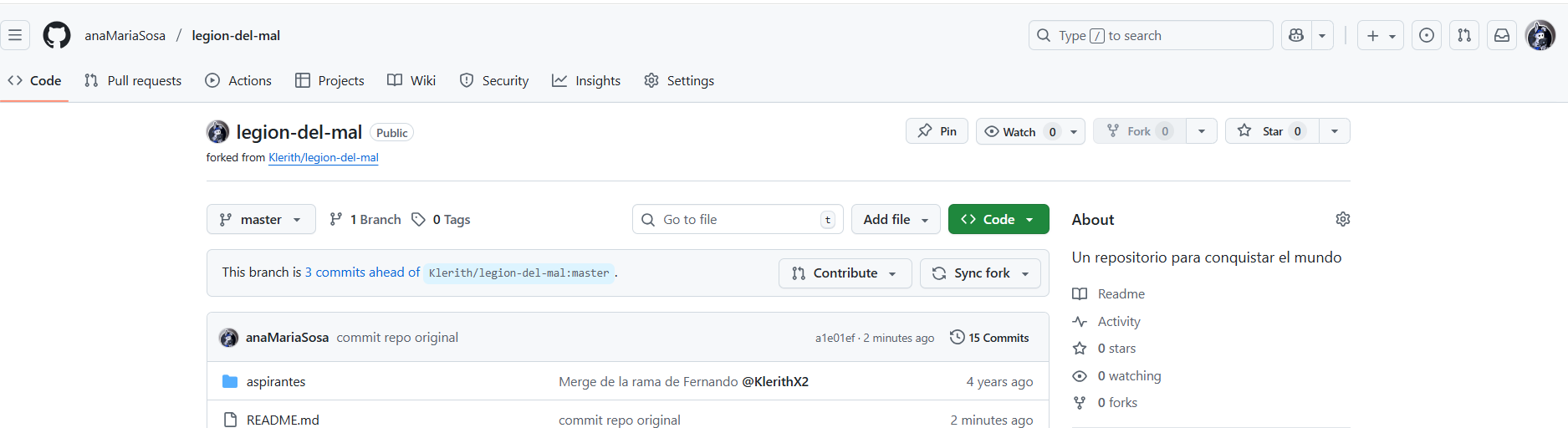
Hay que hacer un pull del repositorio original, para ello haz click en sync fork en donde sale comparar o mergear, luego git pull en tu local y ya está todo

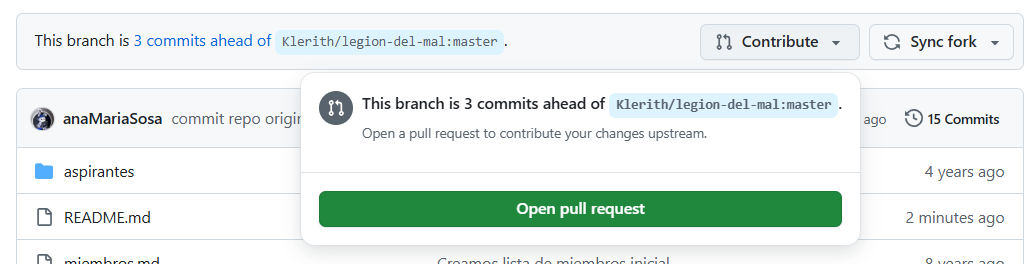
Para hacer esto por consola:

* **git remote add upstream urldelrepoOriginal** # se le suele llamar upstream peor puedes ponerle lo que quieras
* **git pull upstream** #y sigue las instrucciones que aparecen en pantalla git rebase etc..

## Subir commit al repositorio original:

hago commit en mi repo forkeado local, lo subo a mi remoto y me voy al github:





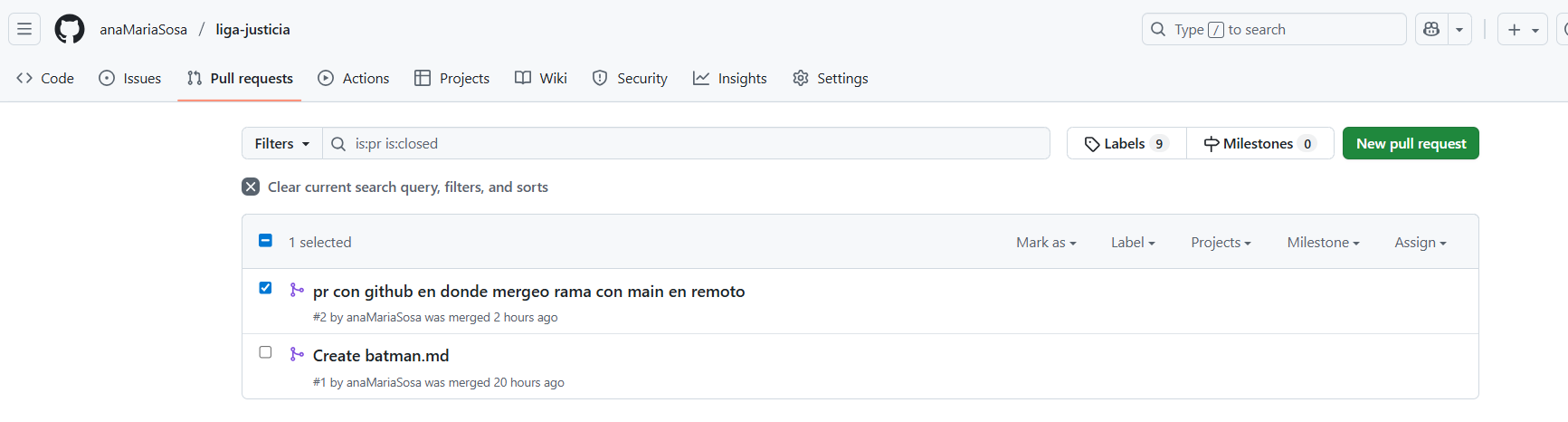
En el open pull request está todo el detalle, le das a create PR con un comentario descriptivo. Los colaboradores verían:

* va a pull request
* haces click en el PR que te interesa, ves el detalle de cada commit del PR, haces click en merge o cancelarlo PERO OJO mejor ir a files change primero para aprobar, rechazar o revisar, una vez submites la revision/aprobacion etc, vuelves eliges mergear, añades comentario y haces click en el botón seleccionados

# **Módulos, Variables de entorno y Entornos (Environments)**

modules y env en el .gitignore para crear varios archivos de configuración por entorno por repositorio etc sin alterar el global (el de mi pc)

# **Issues, Labels y MileStones**



Puedes crear issues y asignarles PR por si pasa a alguien mas que este trabajando en el repo

Usando el #y el número de issue puedes referenciar tu PR a la issue o poniendo el hash del commit en los comentarios o en el mensaje del commit “Fixes #5: lo que quieras” github lo reconoce! Las labels nos ayudan a gestionar las issues

Milestones: para los hitos, al igual que los tags pueden usarse para releases como agrupaciones de issues (asi planificas en avance y luego con los PRs lo vas resolviendo)

# **OJO--Comprobar estado de mi rama local vs remota**

# **Python-Jupyter-Modules and Dashboards**

Python

pip config -v list #muestra las rutas validas donde puede estar el pip.ini

py -m pip config debug #para saber donde esta mi pip.ini de acuerdo a las rutas validas

jupyter

lanzarlo en consola:

cd C:\Program Files\Python311\Scripts #ir a la carpeta (si no tienes permisos de escritura #mueve el ejecutable a una que lo tengas

jupyter-lab.exe #verificar que existe y ejecutar (poner tal cual)

## **SubModules**

* submodulos ver video <https://www.youtube.com/watch?v=gSlXo2iLBro>
* Get-ChildItem -Recurse -Depth 2 | Tree l #para ver submodulos 2 por ejemplo, es un comando de SO no de git
* Gis submodule add urldelsubmodulo #crea el .gitmodules y clona y añade el submodulo a tu repo
* En tu rama (main o la que sea) verificar a que commit esta apuntando tu gitlink (referencia al submodulo):
  + Git rev-parser –show-toplevel #para asegurarte que estas en la capreta padre correcta del repo local
  + Git submodule status # te saca el ultimo commit (código sha largo) y la ruta del submodulo
  + Git ls-tree HEAD rutaDelSubmodulo #saca lo mismo que git submodule status
  + Git ls-tree abbrev=7 HEAD rutaDelSubmodulo #saca el commit corto que puedes comparar con el github sin copiarte el sha completo

Imagen que contiene Aplicación

El contenido generado por IA puede ser incorrecto.

### Actualizar mi gitlink:

* + Tienes que tener el repo del submodulo clonado en tu local
    - Actualiza tu repo local del submodulo:
      * cd ruta/del/submodulo
      * git fetch
      * # si estás en detached HEAD:
      * git checkout main || git checkout -b main origin/main
      * git pull
    - actualiza el getlink sin hacer add del repo del submodulo:
      * git ls-remote urlRepoModuleRemoto main #esto te devuelve el sha largo del ultimo commit de main (que también puede ir a burcarlo en el github en commit copiar full sha)
      * git update-index –cacheinfo 160000 fullSHAdelSubmodulo rutaLocalSubmodulo #la carpeta vacia!
      * Para saber la ruta de tu submodulo: git config -f .gitmodules –get-regexp ‘submodule\..\*\path’
      * git commit -m “lo que quieras”
      * git ls-tree –abbrev=7 HEAD rutaLocalSubmodulo #verificas que en local ya apuntas al getlink que querias
      * git submodule update --init --checkout -- EF\_suite/SHIFT #actuliza tb la copia asi git submodule status y git ls-tree HEAD 'EF\_suite/SHIFT' estarán alineados (gitlink y repo local)
      * **haces push de la rama si deseas actualizar también tu rama en remoto**
    - registra el nuevo gitlink bajando el repo del submodulo en tu repositorio padre
      * git add ruta/del/submodulo
      * git commit -m "Bump submodule (ruta/del/submodulo) to <nuevo-SHA>"
      * git push -u origin mi-nueva-rama

Dashboards: <https://www.youtube.com/watch?v=7yAw1nPareM>

Resumen:

* clonar repo
* Bajar rama
* Actualizar rama y submodulo
* Commit local
* Subir al remoto

VSCode:

* Control + shift + P 🡪 te abre la línea de comandos para reload window por ejemplo
* En una clase Control + shift + O 🡪te muestra la lista de variale sy metodos que posee
* Sobre un método F12 🡪 navega a su definicio
* Sobre un método Shift + F12 🡪 todas las referencias al método
* Sobre un método Control + espacio🡪 te muestra los parámetros que espera
* Control + shift + F 🡪 navega al menú de buscar (la lupa a la izquierda) buscar texto en el repositorio
* Control + P 🡪 va a un fichero