Git y Github

Git toma instantáneas del archivo (commits)

git init : Git comienza a seguir nuestro proyecto

- Crea dos áreas donde irá almacenando archivos:
    - **Staging area**: Área temporal (podremos ver qué archivos sigue Git y cuáles no)
    - **Repositorio local**: Se almacenan las instantáneas de Git

Creamos una carpeta : Streamlit

CREAMOS UN ARCHIVO: main.py

Windows:

- Entramos en una carpeta → botón derecho → Git Bash Here
- Trabajamos desde la git bash de Visual Studio Code

Mac:

- Trabajamos desde la git bash de Visual Studio Code

Hacemos el git init. Debería aparecer una carpeta oculta en la carpeta dónde se ha realizado el git init llamada **.git**

(Cmd + Shift + . en Mac para ver las carpetas ocultas)

<u>Directorio de trabajo → Staging Area</u>

git add *main.py* : Añade el archivo main.py al staging área.

git add . : Añade todos los archivos de la carpeta al staging área.

<u>Para ver lo que está siguiendo git</u>

git status

git status -s

<u>Staging Area → Repositorio Local</u>

git commit -m “Primer commit”

La primera vez que haces commit, te puede dar un error y te pide que le des un correo y un username…

git config –global user.email “mi@correo.com”

git config –global user.name “Mi nombre”

Hacemos el commit después de esto y ya sale bien

Si hacemos **git status -s** (o git status --short) vemos los archivos con la siguiente característica:

- Una **A** si el archivo está en el staging area y no hay cambios que “commitear”. Importante guardar el archivo antes de hacer commit (Ctrl+S o Cmd+S)
- Una **M** indica si el archivo está en el staging area y falta commit.
- Una **AM** indica que el contenido del archivo.py en el working directory y en el staging area son diferentes, es decir, que se ha modificado el archivo en local y los cambios no se han guardado en el staging area aún.
- Una **??** si el archivo no está en el staging área.

IMPORTANTE: Hacer git add siempre antes de git commit.

En el caso en que no hagamos git add antes, el commit nos dará un error:

- *Changes not staged for commit.*

¿Cómo ver tus commits?

git log --oneline

Verás que aparece un número antes que el commit:

`4170923` *Es el ID del commit*

```
$ git log -oneline
4170923 (HEAD -> master) Primer commit
```

Hacer add y commit a la vez es posible:

Hacemos un segundo commit

git commit -am “segundo commit”

```
$ git commit -am 'Segundo commit'
[master 2df61af] Segundo commit
 1 file changed, 2 insertions(+)
```

git log --oneline para ver que los commits

```
$ git log --oneline
2df61af (HEAD -> master) Segundo commit
4170923 Primer commit
```

PD: Si se os queda colgado y os pone ":", tocar la letra "q" para salir

¿Cómo restaurar el archivo al primer commit?

git reset --hard 4170923

4170923 es el ID del commit al que quiero volver. (El ID aparece cuando hacemos git log --oneline

Tened cuidado, puesto que si hacéis esto perderéis todo el código creado después del commit al que hemos vuelto.

Si queréis volver a un commit antiguo y sobreescribirlo con lo que tienes

git reset --soft 4170923

GITHUB

Creamos un nuevo repositorio en GitHub → **TallerGit**

...or push an existing repository from the command line

```
git remote add origin https://github.com/Alexcar934/TallerGit.git
```

Lo metemos en la consola

```
(git branch -M main)
```

Esto no lo vamos a usar aún, pero sirve para que la rama principal del proyecto se llame "main"

```
git push -u origin main
```

Lo metemos en la consola y nos devuelve lo siguiente

```
$ git push -u origin main


error: src refspec main does not match any
error: failed to push some refs to
'https://github.com/Alexcar934/TallerGit.git'
```

Me da un error porque la rama principal de mi proyecto en Git se llama master

```
$ git push -u origin master

Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 261 bytes | 261.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Alexcar934/TallerGit.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
```

git push : Cambios de Repositorio Local a GitHub.

git pull : Cambios de GitHub a Repositorio Local.

git fetch : Te muestra los cambios hechos en GitHub, pero no te los hace

Tags: Sirven para guardar versiones del proyecto

¿Cómo especificar una versión de tu proyecto desde la terminal?

git tag 25-04 v1 -m 'Versión 1 del proyecto'

¿Cómo clonar un repositorio de GitHub en una carpeta local?

Vamos al repositorio de GitHub y copiamos el URL →

Vamos a la carpeta donde queremos guardar el repositorio→

Git Bash Here o VSCode. →

git clone url

Creamos una rama

git branch rama 1

Vemos las ramas que tenemos

git branch

Moverse a la rama “rama 1”

git checkout rama 1

Modificamos algo en la rama 1 y lo unimos a la rama master

1. **git checkout master**

El merge se hace desde la rama master.

2. **git merge rama 1**

Ahora modificaremos algo en la rama 1 que sea diferente en la rama master

Ejemplo: Donde en la rama 1 pone print(“soy la rama 1”) en la rama master pone print(“Soy la rama master”)

1. **git checkout master**
2. **git merge rama 1**

Genera un conflicto y visual studio code te da distintas pautas sobre como resolver el conflicto

¿Cómo eliminar una rama?

git branch -D “rama 1”

(D de delete)

SOLO CON VISUAL STUDIO CODE

Creamos nueva carpeta local llamada “Proyecto B”

Entramos al directorio desde VSCode y hacemos:

Archivo → Guardar espacio de trabajo como… → Aceptar

Archivo → Abrir espacio de trabajo → Proyecto B

Vamos a la terminal de VS Code → **git init**

Creamos un nuevo archivo “proyecto.py”

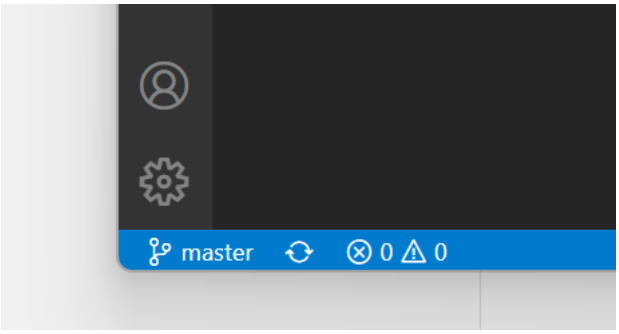
Le añadimos dos líneas de código (dos variables) y creamos el primer commit. Git te pedirá si quieres añadir todos los archivos al staging area antes de hacer el commit (es decir, hacer git add .). Le dais a “Sí”.

Después, donde hacéis el commit, os saldrá “Public Branch”. Si le dáis, os pedirá que os conectéis a github, y desde Visual Studio te creará un repositorio en GitHub (te pedirá si lo quieres público o privado) llamado “Proyecto B”

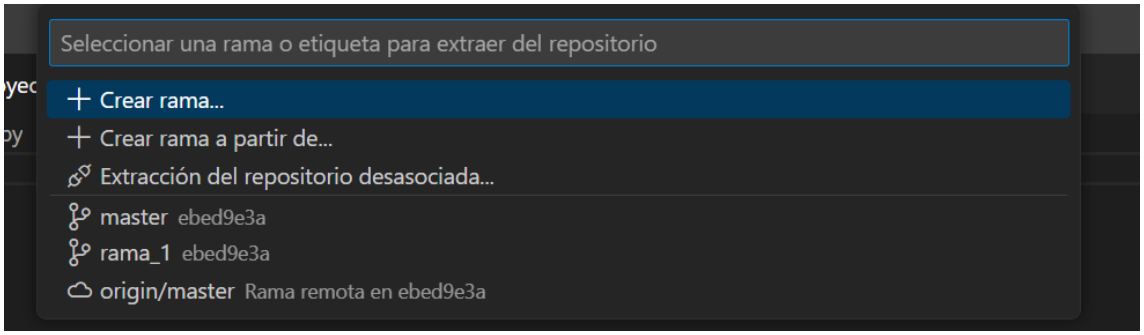
A partir de ahora, cada vez que hagáis un commit, os aparecerá “Sincronizar cambios”, es decir, **git push**

RAMAS:

Abajo a la izquierda en VSCode aparece la rama en la que estamos



Si hacemos click, aparecerá lo siguiente:



Desde aquí podéis:

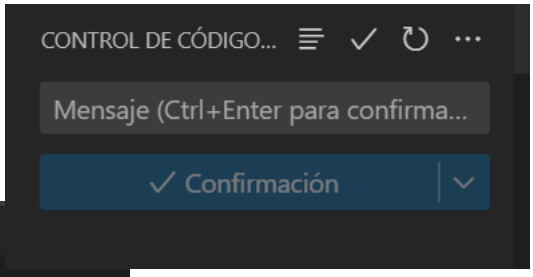
Crear una rama (es decir, **git branch rama_1)**

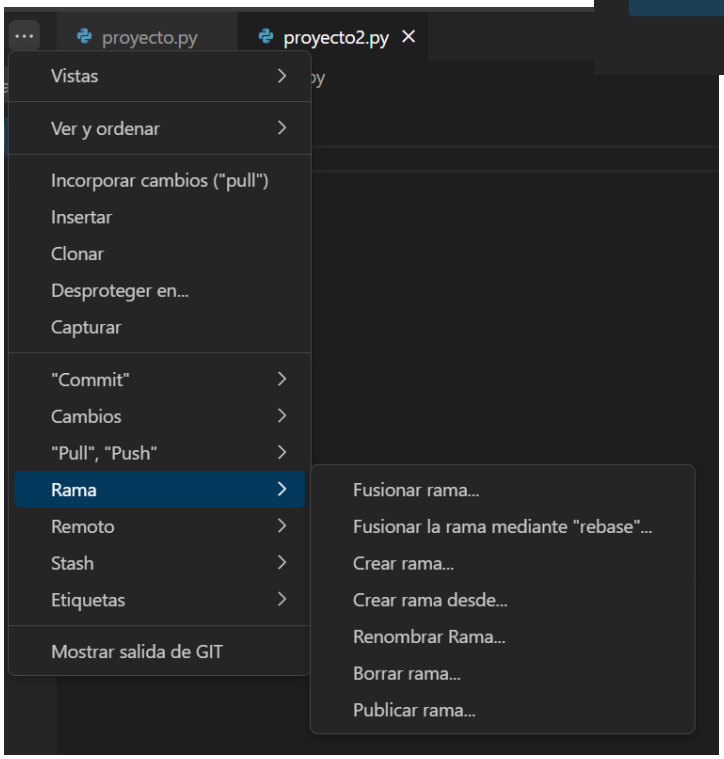
Nota: Cuando creéis una rama y accedáis, donde se hacen los commits os aparecerá “Public Branch” para subir la rama a GitHub

Crear rama a partir de otra rama ya creada

Las ramas que tenéis disponibles aparecen ahí, por lo que podéis acceder a ellas haciendo click. (es decir, el **git checkout …**)

Dentro de los 3 puntitos que se ven en la imagen siguiente a la derecha, tenéis diferentes opciones:





Insertar = git push

Fusionar rama = git merge

Etc. 😊

Echarle un ojo porque es muy útil!

Algo adicional interesante es **git stash**, os dejo una página que lo explica muy bien:

https://www.freecodecamp.org/espanol/news/git-stash-explicado/