**GitHub**

Primer de tot hem de crear un compte a GitHub. Ho farem amb el compte de correu de l’institut.

Un cop hem fet tots els passos de creació del compte podrem iniciar sessió i crear un repositori.

Tot i que hi ha a la pantalla un accés directe, clicarem a la icona del nostre perfil per veure totes les opcions que hi ha.

Les més importants son:

*Your profile (El teu perfil): Accés al teu perfil públic, on altres usuaris poden veure els teus repositoris, contribucions i informació personal.*

*Your repositories (Els teus repositoris): Mostra una llista de tots els repositoris que has creat o als quals tens accés.*

*Your projects: Llista els projectes que has creat o als quals estàs col·laborant.*

*Your stars: Mostra els repositoris als quals has donat una "estrella", indicant que els trobes útils o interessants.*

*Your gists: Accés als teus gists, que són petits fragments de codi o notes compartides públicament o en privat.*

*Your organizations: Si formes part d'alguna organització, aquesta opció et permet accedir-hi directament.*

*Settings (Configuració): Aquí pots gestionar la configuració del teu compte, incloent preferències de seguretat, configuració de notificacions, autenticació de dos factors, claus SSH, etc.*

*Feature preview: Pots activar funcionalitats experimentals o en fase de prova.*

Ara clicarem a “Your repositories”:



Aquí crearem el repositori nou omplint alguns dels apartats:



Ara anem al nostre sistema local (Kali) i creem una carpeta on clonarem el repositori remot de github.

Si provem de clonar el repositori directament fent per exemple **git clone <url-del-repositori>**

i posant les nostres credencials ens dona un error:



Encara que per alguns usos es pot generar un token des del propi github, el que farem ara és generar un parell de claus.



Després d’això copiarem la nostra clau pública i l’afegirem a github:



Ara clonarem el repositori a partir de la seva url:



**git clone** **git@github.com:agarrigMontilivi/pythonFuncions.git**

ara ens permet fer la clonació tot i que ens avisa de que hem clonat un repositori buit.

Ara obrirem la carpeta del nostre repositori dins de VS Code i crearem l’arxiu Python del qual volem fer seguiment.



Dins de la carpeta crearem un arxiu que es dirà operacions:



El VS code ens permet fer pràcticament qualsevol operació amb git. Però de moment no farem servir aquestes capacitats.



Si encara no ho hem fet, haurem de fer la configuració inicial de git:



Primer de tot afegim el fitxer al seguiment de git:

**git add operacions.py**

I després fem el commit:

**git commit -m "Afegit fitxer operacions.py que demana dos números i en fa la suma"**

I finalment enviem els canvis al repositori remot amb:

**git push origin main**

*-* ***origin****: És l'àlies que Git utilitza per referir-se al repositori remot d'on es va clonar un projecte o amb el qual està vinculat. En la majoria dels casos, es refereix al repositori a GitHub, GitLab o un altre servei de control de versions remot. Es crea automàticament quan clones un repositori, però també el pots configurar manualment.*

*-* ***main****: És el nom de la branca principal en molts repositoris Git, on normalment es mantenen les versions estables i definitives del projecte. Abans es deia `master`, però moltes plataformes com GitHub van canviar el nom de la branca per defecte a `main` per ser més inclusius.*

*Així, quan executes comandes com `git push origin main`, estàs enviant els canvis de la branca principal (`main`) al repositori remot (`origin`).*

Si ara anem al repositori remot veurem que s’ha pujat l’arxiu:



Ara farem un canvi remotament a aquest fitxer i veurem com el recuperem en local.

Per recuperar localment els canvis que hem fet executem:

**git pull origin main**

cal anar amb compte perquè si tenim canvis fets a l’arxiu localment i no estan confirmats, ens donarà un error. Git està evitant sobrescriure aquests canvis quan intentes fer un **git pull**.

A més, si el contingut local i remot no coincideix, ens donarà un error perquè git no sap com tractar les divergències i ens demana que li diguem com fer-ho.

Les opcions son:

La millor opció entre `merge` i `rebase` depèn de les teves preferències i el flux de treball de l'equip, ja que cada enfocament té els seus avantatges i inconvenients. A continuació et detallo els punts més importants de cadascun per ajudar-te a prendre una decisió:

 *1. Merge (fusió)*

*- Com funciona: Combina els canvis locals i remots en un nou commit de fusió, preservant tot l'historial.*

*- Avantatges:*

 *- Historial complet: Manté tots els commits originals, tant locals com remots, sense reescriure l'historial.*

 *- Resolució senzilla: És més fàcil d'entendre, ja que mostra clarament quan es van fusionar les branques.*

 *- Menys risc de conflictes inesperats: Ja que no es modifica l'ordre dels commits existents.*

*- Inconvenients:*

 *- Historial més desordenat: Es poden crear molts commits de fusió que poden fer que l'historial sigui més difícil de seguir.*

 *2. Rebase*

*- Com funciona: Situa els commits locals a sobre dels canvis remots, creant un historial lineal sense commits de fusió.*

*- Avantatges:*

 *- Historial més net: L'historial es veu lineal i net, sense commits de fusió.*

 *- Més fàcil de seguir: Si treballes en projectes grans o amb molts desenvolupadors, l'ús de `rebase` pot fer que sigui més fàcil veure el camí lògic dels canvis.*

*- Inconvenients:*

 *- Reescriu l'historial: Reorganitza els commits locals, cosa que pot ser problemàtica si ja s'han compartit amb altres desenvolupadors o al remot.*

 *- Conflictes més complexos: Si hi ha conflictes, poden ser més difícils de resoldre amb `rebase`, perquè el procés reescriu els commits.*

 *Quina és la millor opció?*

*- Recomanació per a equips grans o per a fluxos de treball on es prioritza la netedat de l'historial: `rebase` és més recomanable perquè permet mantenir un historial lineal i net.*

*- Recomanació per a usuaris que volen seguretat i un procés més senzill: `merge` és l'opció preferida. Manté tot l'historial original i és menys arriscat.*

Per dir-li a Git que faci un merge (fusió) dels canvis del repositori remot amb els teus canvis locals quan fas un `git pull`, pots utilitzar la següent comanda:

**git pull --no-rebase origin main**

(Si volguessim l’altra opció, faríem git pull --rebase origin main )

Aquesta comanda indica a Git que faci la fusió normal (merge) sense reescriure l'historial com fa `rebase`.

Es pot configurar Git per a què utilitzi el merge per defecte cada vegada que facis un `git pull`:

 **git config pull.rebase false**

 Això aplica només per al repositori actual. Si vols que aquesta configuració s'apliqui a tots els teus repositoris, pots afegir l'opció `--global`:

 **git config --global pull.rebase false**

D'aquesta manera, Git sempre utilitzarà la fusió (`merge`) quan recuperis els canvis amb `git pull`.

**Conflictes Local / Remot**

Quan Git no pot fusionar automàticament els canvis locals amb els canvis que han arribat des del repositori remot, perquè les mateixes línies del fitxer han estat modificades tant localment com remotament es produeix un conflicte.

 Com resoldre el conflicte:

1. Obrir el fitxer conflictiu (`operacions.py`):

 - Obre el fitxer en el teu editor de text (per exemple, VS Code o qualsevol altre editor).

 - Veureu una marca que indica on es troba el conflicte, que tindrà una estructura semblant a aquesta:

 **<<<<<<< HEAD**

 **Canvis locals (que tens al teu ordinador)**

 **=======**

 **Canvis remots (que venen del repositori remot)**

 **>>>>>>> origin/main**

2. Resoldre el conflicte manualment:

 - Tria quina part dels canvis vols conservar: els locals, els remots, o una combinació de tots dos.

 - Elimina les línies `<<<<<<< HEAD`, `=======`, i `>>>>>>> origin/main` un cop hagis resolt el conflicte.

3. Afegir el fitxer resolt a l'àrea d'staging:

 Un cop resolt el conflicte, has d'afegir el fitxer amb:

 **git add operacions.py**

4. Fer un commit per confirmar la fusió:

 Finalment, has de fer un commit per confirmar que has resolt el conflicte:

 **git commit -m "Resolució de conflicte en operacions.py"**

5. Pujar els canvis al repositori remot:

 Un cop fet el commit, pots pujar els canvis resolts al repositori remot amb:

 **git push origin main**