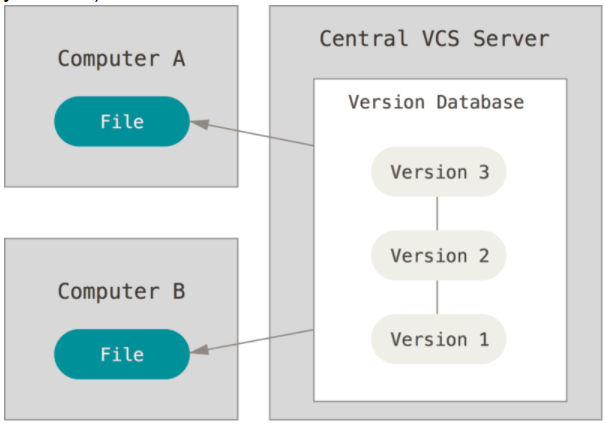
## **Introducción a Git y GitHub**

*(Git es el programa que se ejecuta en su computadora y Github es donde se almacena su trabajo en línea).*

**1 - Distributed Version Control System**: FILES y FOLDS, trabajaremos con un montón de ellos, por lo que este sistema basado en la nube permite trabajar a personas en el mismo servidor (Code Hosting Platform) compartiendo código abierto sin que se “pisen” los unos a los otros → Para ello registra cambios en un archivo o conjunto de archivos a lo largo del tiempo para que pueda recuperar versiones antiguas de estos archivos en cualquier momento posterior.



VENTAJAS:

* Tener **diferentes versiones del mismo archivo** le da un poder masivo sobre su código .
* **Combina código** creado por varios desarrolladores en un archivo.
* Ofrece la posibilidad de **ver quién es el propietario** de cada código.

**2 - Git**: es un **DVCS gratuito y abierto** diseñado para manejar proyectos de programación de computadoras con velocidad y eficiencia.

**Git se ejecuta localmente** . ¿Qué significa? Le permite utilizar el sistema de control de versiones en su computadora portátil. Significa que tendrás **en tu computadora** un Sistema de Control de Versiones, para recuperar todo el código, ver qué cambios has hecho, etc.

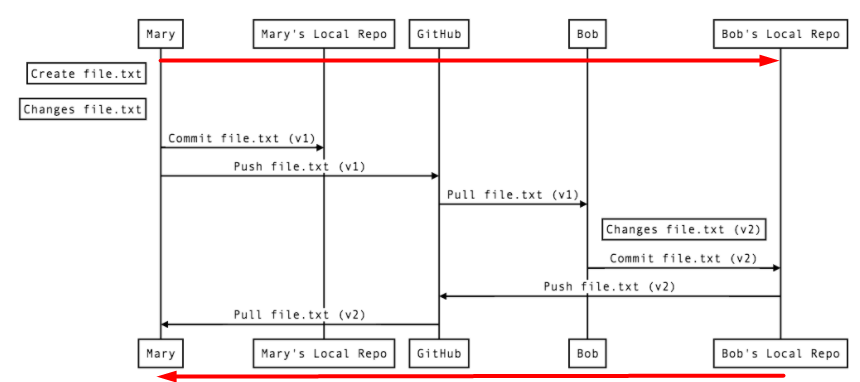
**3 - GitHub**: es una plataforma de alojamiento de código en la nube (**Code Hosting Platform**) para el control de revisiones y la colaboración.

**Git le** permite controlar su código en su *máquina local* , mientras que **GitHub** le permitirá **colaborar** con sus compañeros de equipo *subiendo y extrayendo* el código en el que usted y los demás han estado trabajando.

* **Git →** su sistema de control de versiones a **nivel local** (en su computadora portátil)
* **GitHub →** es el sistema de control de versiones **de su equipo.**

**PUSH & PULL:**

1. **PUSH (subiendo):** significa hacer que el código creado localmente, en su computadora portátil, **esté disponible en línea** (en GitHub) para que los demás puedan usarlo.
2. **PULL (extraer):** significa **llevar** el código de una plataforma en línea (GitHub) **a su computadora portátil** para que pueda reutilizarlo y agregar / eliminar cosas de él.

*EJEMPLO*: Hay dos desarrolladores involucrados aquí, Mary que creará el archivo y lo enviará a GitHub, y Bob que extraerá este archivo, hará algunos cambios y lo enviará de regreso a GitHub para compartir estos cambios con Mary:

* **Commit** = enviar a GIT (una vez que el archivo se agrega y confirma en Git, Git genera la primera versión del archivo)
* **Push** = enviar a GitHub (Al hacer esto, Mary le da a su compañero de equipo, Bob, la oportunidad de obtener este archivo y hacerle algunos cambios.)
* **Pull** = extraer de GitHub (Para hacerlo, Bob tiene que extraer los cambios de GitHub a su computadora).
* **Commit** = enviar a GIT (Automáticamente, aparece una versión del archivo en el proyecto de Bob, también localmente).

→ **Estado actual** En esta etapa, antes de que Bob introduzca los cambios en GitHub, Mary tiene la **versión 1** del archivo, mientras que Bob tiene la **versión 2** del mismo archivo. Después, cuando Bob introduzca los cambios, ambos desarrolladores tendrán ambas versiones.

---------------------------------------------------------------------------------------------

**git config** Git viene con una herramienta llamada git config que le permite obtener y establecer variables de configuración que controlan todos los aspectos de cómo se ve y funciona Git. Estas variables se pueden almacenar en tres lugares diferentes:

$ git config --global user.name "John Doe"

$ git config --global user.email [johndoe@example.com](mailto:johndoe@example.com)

---------------------------------------------------------------------------------------------

[video de uso](https://www.youtube.com/watch?v=kHkQnuYzwoo&ab_channel=EdGoad)

**MINI CONDA:** una versión reducida de conda para poder tener diferentes entornos

**Entorno:** hay que tratar de trabajar en entornos, para si hacemos algo mal o importamos algo “peligroso” que simplemente se cargue el entorno.

Cuando instalamos Conda creamos el entorno “base”, en el cual se implementa python. Lo que vamos a hacer es:

* **crear un entorno para jupyter** → lo vamos a utilizar solo para EJECUTAR jupyter → **jp\_env**
* **crear un entorno para el bootcamp** → lo vamos a utilizar para INSTALAR las **LIBRERÍAS** (numpy, pandas, tensorflow, …)**, trabajar, etc** → **ironhack**

**PASOS PARA CREAR UN ENVIRONMENT NUEVO** (en este caso jp\_env para instalar jupyter)

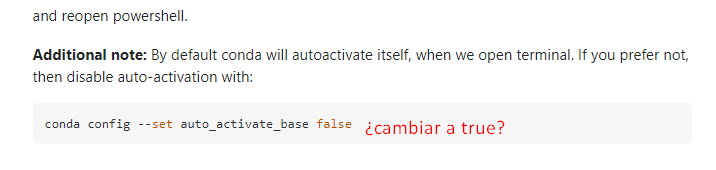
1º (base) PS C:\Users\AlvaroSaez> **conda init powershell** *(paso ya está hecho)*

2º (base) PS C:\Users\AlvaroSaez> **conda create -n jp\_env python=3.7 anaconda**

3º (base) PS C:\Users\AlvaroSaez> **conda activate jp\_env // *conda deactivate***

4º (jp\_env) PS C:\Users\AlvaroSaez> **conda install ipykernel // pandas // nodejs**

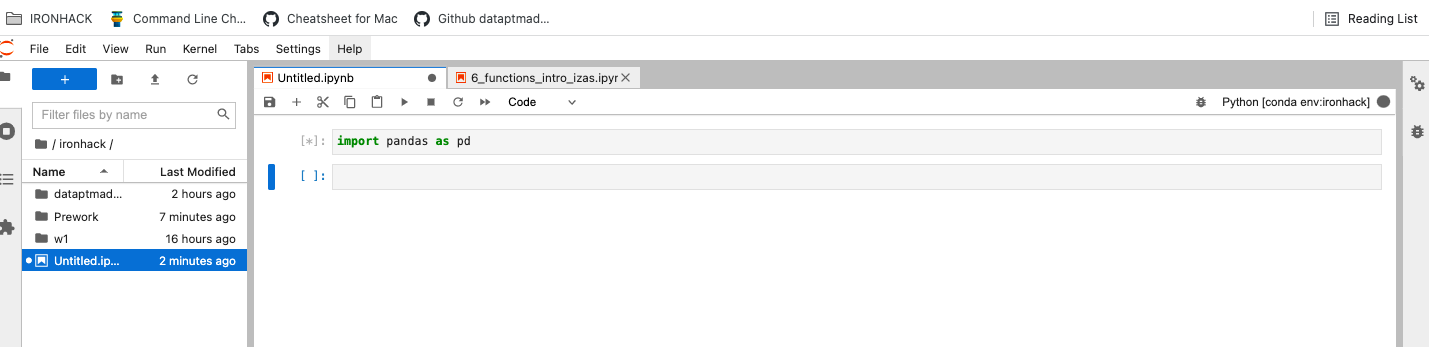
5º (jp\_env) PS C:\Users\AlvaroSaez> **jupyter-lab**



\* Si queremos [**eliminar un entorno**](https://docs.conda.io/projects/conda/en/latest/commands/remove.html)**,** tenemos que **salir de él**, y aplicar el siguiente comando: (base) PS C:\Users\AlvaroSaez> **conda remove -n<env name>**

**INSTALAR LIBRERÍAS EN NUESTRO ENTORNO DE IRONHACK:**

* Abrimos una nueva ventana de terminal
* “**conda activate ironhack**” me voy a mi entorno de ironhack
* “**conda install pandas”** y confirmamos con un “**y**” y queda instalado en el entorno de ironhack
* Y ahora comprobamos que en el entorno de ironhack al ejecutar “**Import pandas as pd**” no nos da fallo
* “**conda list”** permite ver las librerías instaladas en ese entorno



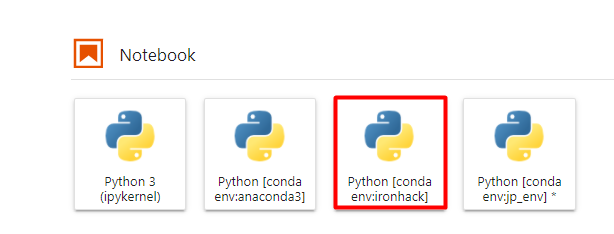
PASOS A PARTIR DE AHORA (una vez creado el env e instalado jupyter y kernel):

1º abrir **powershell**

2º activar el entorno de **jp\_env → conda activate jp\_env **

**3º** abrir **jupiter → jupyter-lab** or  **jupyter notebook **

**4º** comando para continuar utilizando powershell **→**

**5º** trabajaremos en el **entorno de ironhack **

**GIT:**

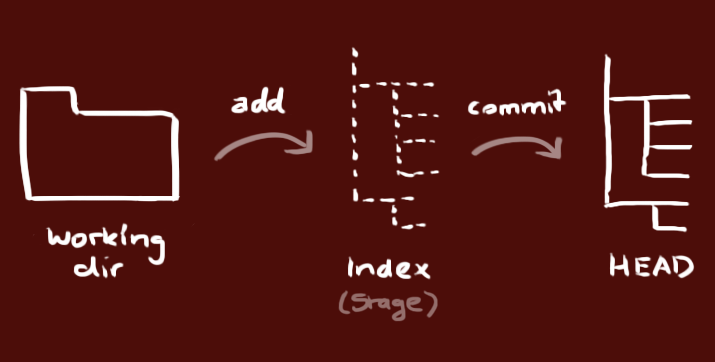
Como hemos dicho, Git es un **DVCS gratuito** y abierto diseñado para manejar proyectos de programación de computadoras con velocidad y eficiencia. Está diseñado como un **directed acyclic graph**.

TERMINOLOGÍA**:**

* **repository | repo**: ubicación de almacenamiento para paquetes de software
* **Tracked**: cualquier archivo que se haya puesto bajo la "vigilancia" de git a través del comando **git add**
* **Untracked**: cualquier archivo que git aún no sepa qué hacer al respecto (es importante para no almacenar de más)
* **commit**: snapshot del estado actual de un repositorio
* **branch**: separación de un commit en dos líneas (two children)
* **merge**: unión de dos commits en uno solo
* **pull request**: request for a merge of some changes to the repo's owner
* **pull**: extraer de GitHub a Git actualizando el repo local
* **push**: enviar a GitHub actualizando el repo remoto

Tiene las siguientes UBICACIONES:

* **Working space/directory**: (working tree). Contiene todos los archivos para un único “commit”, es donde ocurre su trabajo
* **index**: realiza un seguimiento de los archivos que se incluirán en el próximo “commit”. Como un entorno de staging.
* **HEAD**: Contiene el último “commit” hecho por nosotros .
* **local repository**: contiene todos los “commit” del proyecto. Se encuentran en un directorio **.git** oculto en su proyecto
* **remote repository**: (como GitHub), contiene los “commit” del proyecto pero en una ubicación remota.

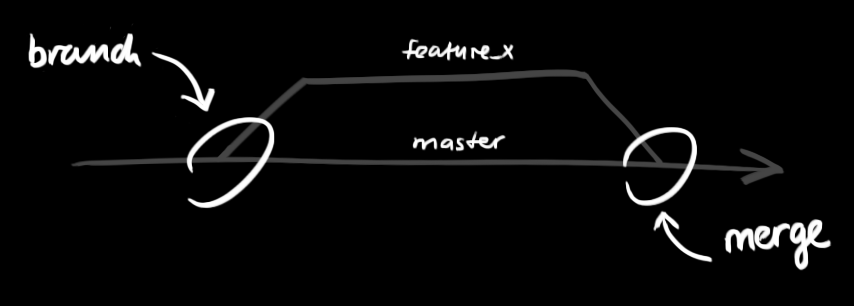


→ **Para añadirlo a repositorio LOCAL:** Trabajamos en un directorio, cuando hacemos cambios en alguno de los archivos tenemos que hacer un **git add <file name>** (se almacenará en staging), para finalmente añadirlo en nuestro head tendremos que hacer un **git commit -m <”mensaje”>** (todavía no estará en nuestro repositorio remoto, sino local)

→ **Para enviarlo a repositorio REMOTO**:

* Si ya tenemos clonado un repositorio haremos lo siguiente **git push origin <branch name>**. Si
* Si no lo hemos clonado: **git remote add origin <server>**

RAMAS: Se utilizan para **aislar cambios** sin que afecten al proyecto principal. Por lo que debemos de **crear una rama nueva** (**git checkout - b <branch name>**), trabajar en ella y al final hacer **un merge sobre la rama principal** (**git merge <nombre de la rama>**).



\* Una rama no está disponible para otros a menos que envíe la rama a su repositorio remoto **git push origin <branch name>** RECORDAR, que previamente los archivos deben de ser añadidos y hacer el commit → **git add <filename>** y **git commit -m “<mensaje>”**.

**ORDEN**:

* **git init** o **git clone** (para crear o clonar un repositorio y tener la carpeta oculta .git)

→ normalmente los haremos dentro de la carpeta “ironhack”: **git clone + <repository link>**

* **git status** (para ver si realmente nuestra carpeta es un repositorio)
* **git checkout - b <branch name>** (para cada assignment de clase crearemos una nueva rama)
  + git checkout  **<branch name>** (para entrar a una rama sin querer crearla)
* *// creamos archivos, trabajamos en ellos*
* **git add <filename>** (para añadir el nuevo archivo o los cambios en el archivo a la nueva rama *- se añade a staging -*)

\* **git add .** → añade **todos** los archivos (es buena practica hacerla yendo a la base de la ruta)

\* **git restore** **<filename>** (si queremos descartar este cambio)

* **git commit -m “<mensaje>”**
  + **git commit -m "lab-started"** (mientras trabajamos)
  + **git commit -m "lab-finished"** (cuando acabamos de trabajar)
* **git checkout main** (o git switch main) [antes de hacer el merge de la rama nueva a la main, deberemos de entrar en la main]
* **git merge <nombre de la nueva rama>** (estaremos copiando todo a la main)
* **git push +** [**“remote link” (con el token**](https://docs.google.com/document/d/1FRFjLHP06Q-SMj-qcDKxeFNR55OWlmNHy2nD6qWBXCI/edit)) (si ponemos el remote link hacemos un push de TODO) o **git push origin <branch name** (intentar que sea la main)**>** (si el repositorio ya está clonado)

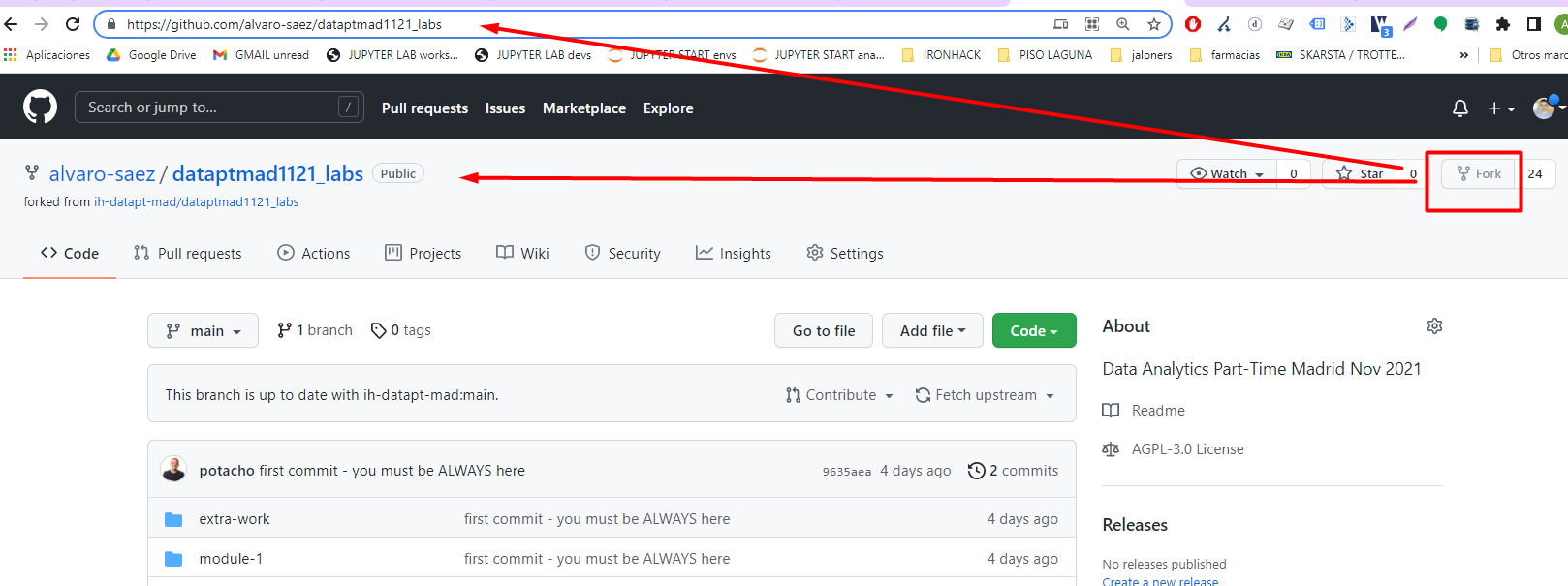
***\* Es CONVENIENTE ANTES DE HACER EL PUSH hacer un git pull o git fetch →* git pull origin main***(por si acaso otros han hecho algún cambio en el repositorio remoto. Un git pull hace una “fusión” con nuestro main, en cambio un fetch te copia el repositorio remoto en un nuevo branch local, por lo que luego tendremos que hacer un merge). También podemos* **git pull +** [**“remote link” (con el token**](https://docs.google.com/document/d/1FRFjLHP06Q-SMj-qcDKxeFNR55OWlmNHy2nD6qWBXCI/edit)).

* **pull request dentro de la interfaz de git hub →** “create pull request”

IMPORTANTE: cada vez que hagamos algo nuevo, crearemos una RAMA nueva de nuestro MAIN, después haremos un GIT ADD y un GIT COMMIT, iremos a la rama MAIN y haremos un merge de la nueva a la main

**LAB | Resolving Conflicts in Git**

1. **fork** the remote repository



1. **clone it** as a **local repository**

→ git clone https://github.com/alvaro-saez/dataptmad1121\_labs

1. **create a branch** for this lab called resolving-git-conflicts

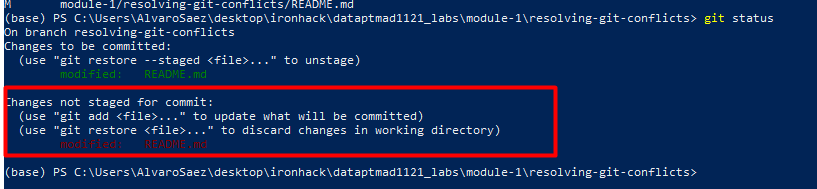
→ git checkout -b resolving-git-conflicts

1. see if it is a repository:

→ git status

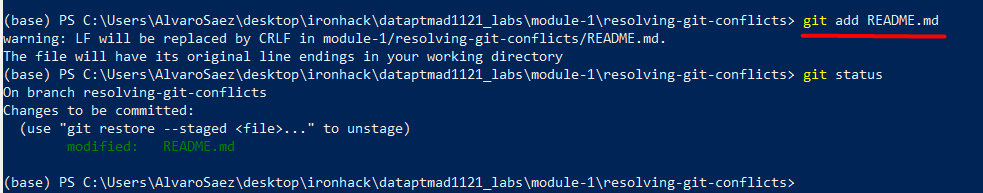
1. open the file README.md and make a change (save it):
2. check the changes in the repository:

→ git status

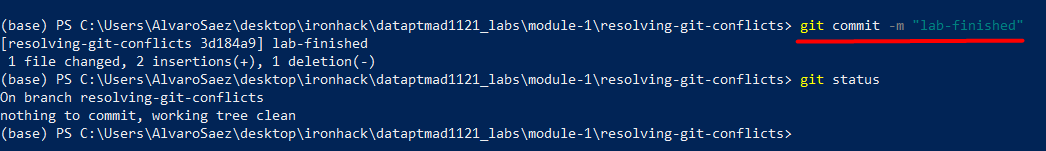


1. **add this change** to staging

→ git add README.md (in its path)



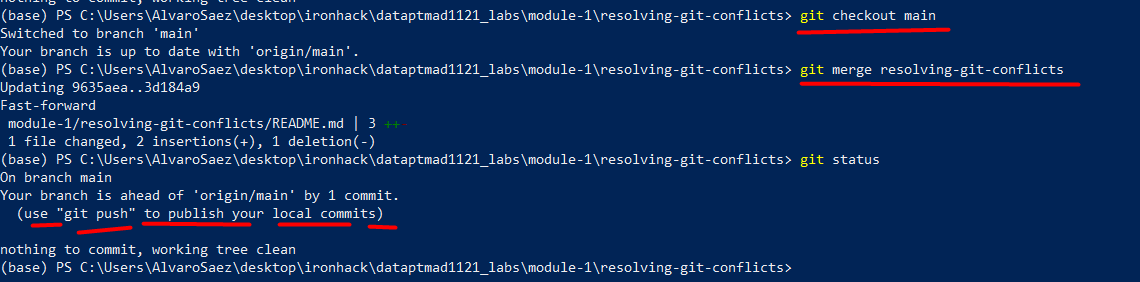
1. **commit the change**

→ git commit -m "lab-finished" 

1. **go to** the **main branch** and **merge the changes** from the resolving-git-conflicts branch

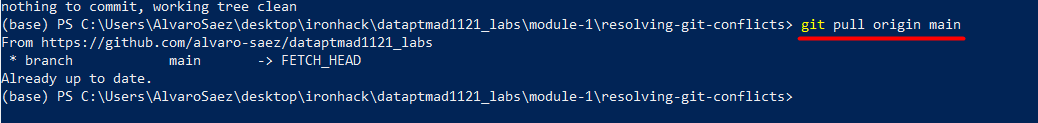
→ git checkout main

→ git merge resolving-git-conflicts



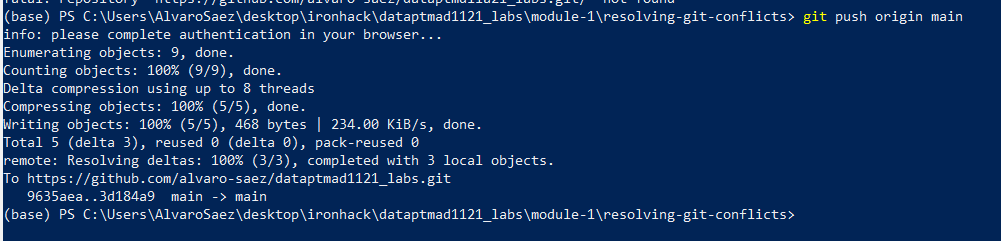
1. we should make a push to send to the remote repository all changes, buf before that, we are going to **make a pull to see if there is any change**.

→ git pull origin main



1. Make the push to the remote repository

→ git push origin main



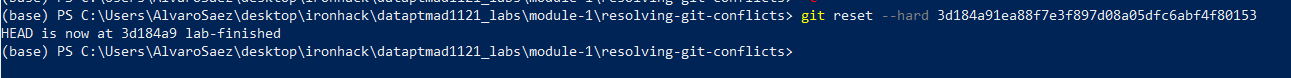
**\* EXTRA STEP:** CREATE CONFLICTING CHANGES:

1º I have added a markdown cell in README.ms file, made a git add and a git commit. Using the git log I see:

* The last commit ID: “05eb22e7b9077d4a1252fc8f9beb6e0ce99e5c2e”
* The prior commit ID: 3d184a91ea88f7e3f897d08a05dfc6abf4f80153

2º Revert to the commit before:

→ git reset --hard 3d184a91ea88f7e3f897d08a05dfc6abf4f80153



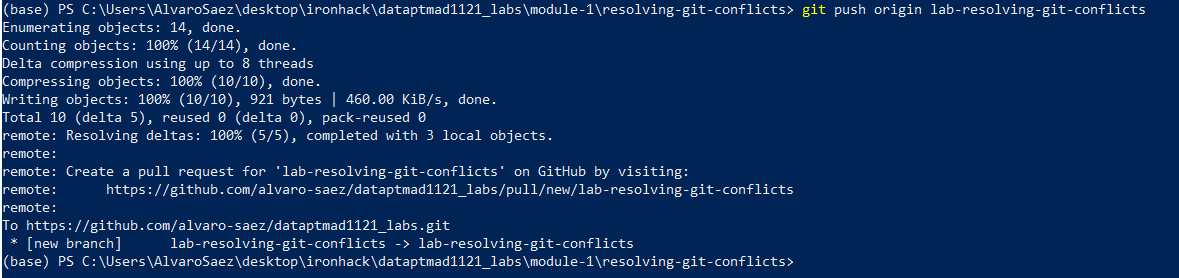
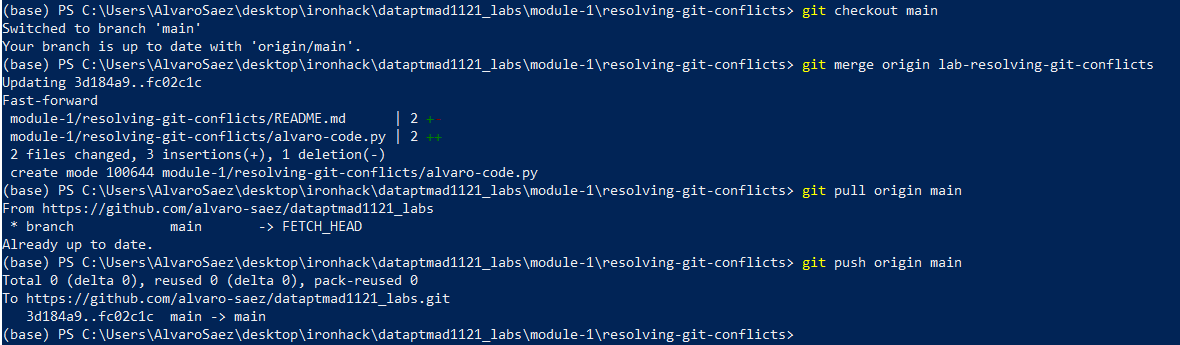
3º Create a new branch called lab-resolving-git-conflicts

→ git checkout -b lab-resolving-git-conflicts

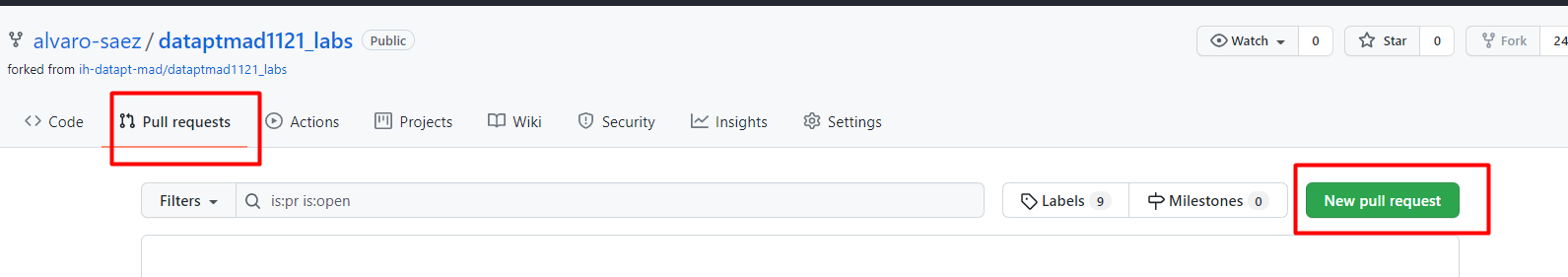
4º git add README.md and git commit -m “lab-finished”

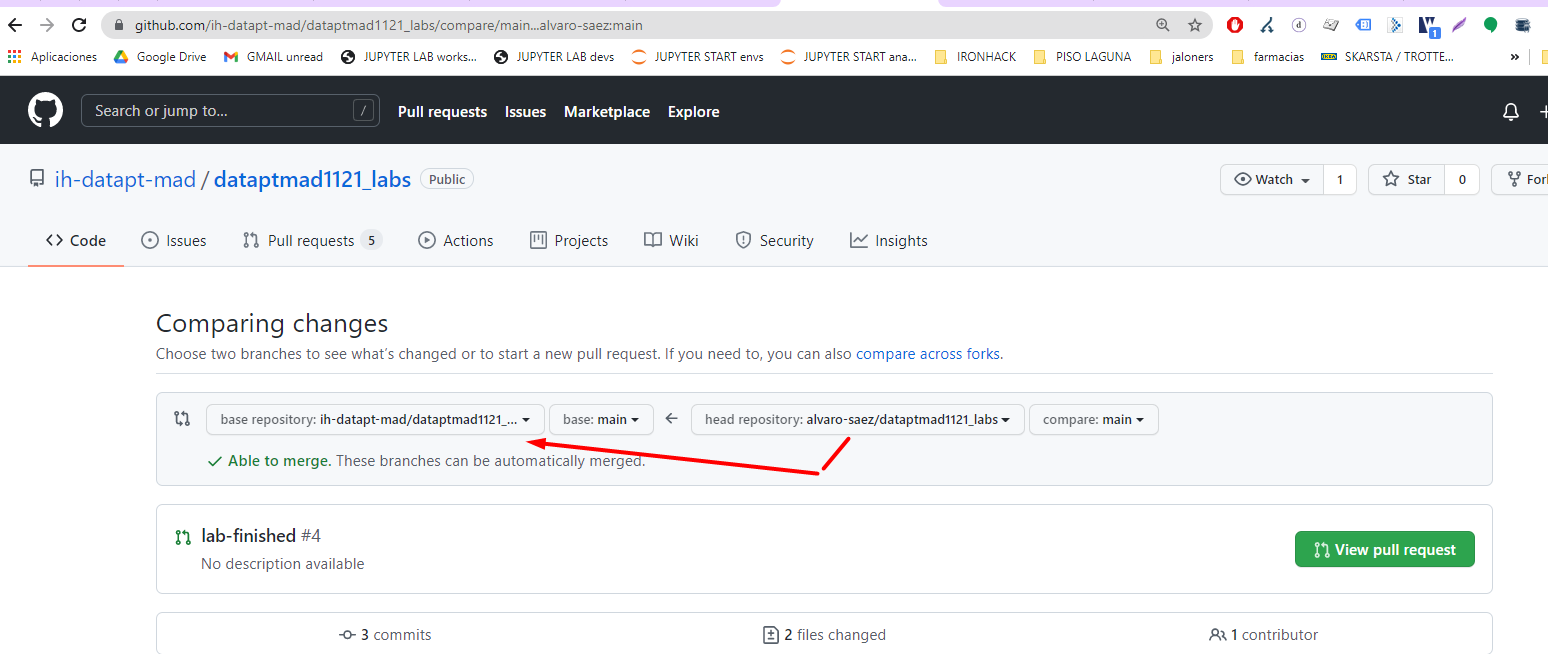
5º git pull to import any possible change and make a push

→ git push origin lab-resolving-git-conflicts

\*he hecho un push de lab-resolving.git-conflicts, pero una buena práctica era salir de esa rama, ir a la main, hacer un merge y ya después hacer el push

6º **make a pull in the GitHub U**I: para enviarlo desde nuestro repositorio remoto al repositorio remoto del bootcamp





RESUMEN DE COMANDOS**:**

* **git init:** Para crear un REPOSITORIO convierte una carpeta normal y corriente en un repositorio (creando así la carpeta **.git**)
* **git status**: para ver si una carpeta es un repositorio
* **git log**: muestra el historial de los commits
* **git ignore:** decimos a git que ignore la acción sobre el archivo, o que ignore los archivos. Con esto lo ignora cuando hagamos un push y así no llegamos al límite de GIT y no tendremos que pagar.
* //BRANCHS
* **git branch <nombre de la rama>**: crea nuevas **ramas**
* **git checkout <branch name>:** para entrar en la rama creada
* **git checkout - b** **<branch name>** te crea y te mete en la rama nueva directamente
* **git branch -D <nombre de la rama>:** eliminamos la rama. Para eliminar la rama tendremos que estar fuera de ella (es decir en otra rama)
* **git merge****<nombre de la rama>:** une la rama con la rama activa actual
* *para los siguientes necesitamos un token*
* **git clone + link**:
  + para clonar un repositorio remoto en local. → **git clone** [https://](https://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@github.com//alvaro-saez/dataptmd11a21_labs.git)ghp\_HMG6tpURs8uR0aTzCuUTm9x0IBIbt01vpNuC[@github.com/alvaro-saez/dataptmd11a21\_labs.git](https://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@github.com//alvaro-saez/dataptmd11a21_labs.git)
  + para clonar un repositorio local → **git clone** /path/to/repository
* **git push <remote> <branch>** → **git push** [https://](https://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@github.com//alvaro-saez/dataptmd11a21_labs.git)ghp\_HMG6tpURs8uR0aTzCuUTm9x0IBIbt01vpNuC[@github.com/alvaro-saez/dataptmd11a21\_labs.git](https://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@github.com//alvaro-saez/dataptmd11a21_labs.git)
* **git pull <remote> <branch> → git pull** [https://](https://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@github.com//alvaro-saez/dataptmd11a21_labs.git)ghp\_HMG6tpURs8uR0aTzCuUTm9x0IBIbt01vpNuC[@github.com/alvaro-saez/dataptmd11a21\_labs.git](https://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@github.com//alvaro-saez/dataptmd11a21_labs.git)
* **git fetch + link →** Para ver los cambios que han hecho otros en el repositorio remoto y ver si tenemos que fusionar estos cambios con los nuestros en local
* **git add <file name>**: añadir como en “staging” los cambios que hemos estado haciendo en el log de .git (nuestra carpeta oculta que tenemos creada por defecto en nuestro repositorio)
* **git commit**: hace un **push** del fichero y **genera** un identificador único (hash) de ese punto de tiempo (**tiempo de referencia**) para tener un registro de todos los cambios y poder volver atrás.
  + **git commit -m “<mensaje>”**el último commit (una vez añadido los nuevos archivos -add-) hay que poner **git commit -m “lab finished”**
  + **git commit -a :** hace un commit para todos los archivos

[TUTORIAL](https://agripongit.vincenttunru.com/)**.**

[TUTORIAL 2:](https://rogerdudler.github.io/git-guide/index.html)

**Comandos de terminal** → [en el siguiente doc](https://docs.google.com/document/d/1n54uYaq_ARZiaaGXpWk2A3Md5fY5hhGP1nzQjxXZhSU/edit#heading=h.h0nhcg35x03r)