

Utilisation de GitHub en équipe

E.ROMETTE & A.FALIGOT-GIRARDELLI

07/05/20

Contents

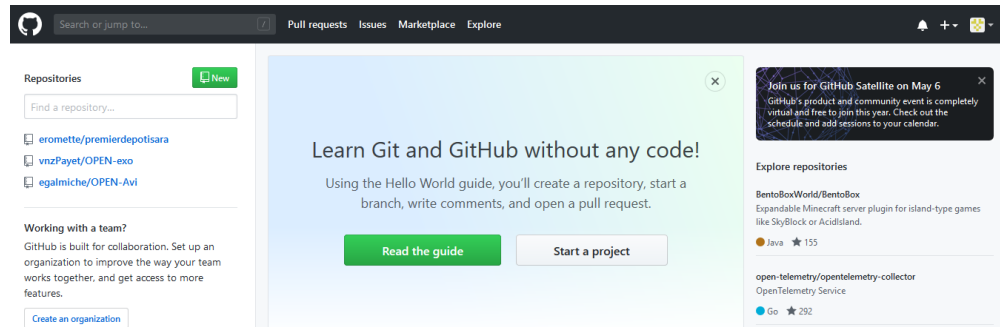
1	<i>Initialisation du repository d'équipe</i>	2
2	<i>Ajouter les membres de l'équipe</i>	4
3	<i>Gestion de fichiers par la console</i>	4
4	<i>Gestion de fichiers par GitHub Desktop</i>	5
4.1	<i>Explications :</i>	5
4.2	<i>Lancement du logiciel :</i>	5
4.3	<i>Cloner un répertoire :</i>	6
4.4	<i>Commit de fichiers :</i>	7
4.5	<i>Historique des commits :</i>	9
4.6	<i>Le principe de "branch" :</i>	9
4.7	<i>Effectuer un push/pull :</i>	10
4.7.1	Push :	10
4.7.2	Pull :	10
5	<i>Références</i>	11



Résumé : Dans cette synthèse nous allons étudier la gestion de GitHub pour un travail d'équipe.

1 Initialisation du repository d'équipe

Une fois votre compte GitHub ouvert, allez au Menu principal du site où vous pouvez avoir une vision sur tous les Répertoires que vous avez créé ainsi que ceux que vous avez cloné.




Pour créer un nouveau Répertoire, cliquez sur le bouton “New” en vert. Cela vous amènera sur la page suivante :

Create a new repository


A repository contains all project files, including the revision history. Already have a project repository elsewhere?
[Import a repository.](#)


Owner Repository name *

 eromette /

Great repository names are short and memorable. Need inspiration? How about [redesigned-goggles?](#)

Description (optional)

☒  **Public**
Anyone can see this repository. You choose who can commit.

☐  **Private**
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer.

Add .gitignore: **None** | Add a license: **None** 

“**Owner**” indique le propriétaire du répertoire, le compte qui a tous les droits : l’administrateur.

“**Repository name**” correspond à l’emplacement où vous pouvez choisir le nom du répertoire. Il est conseillé de l’avoir en 1 seul mot (peut être composé de plusieurs mots mais attachés avec des caractères spéciaux comme `&` - `_`).

“**Description**” correspond à la description qui sera faite en haut de votre répertoire.

“**Public/Private**” permet de choisir si le répertoire est visible de tous ceux qui visitent le profil du créateur ou non. En cas de “Private”, il faut une autorisation d’accès.

“**README.Md**” correspond à un fichier en Markdown donnant les informations principales du répertoire. Il est souvent utilisé pour donner les consignes, des indications ou les notes de mises à jour.

“**Add .gitignore**” permet de créer un fichier `.gitignore` qui peut être édité textuellement. Ainsi, si le dossier contient :

'*.txt' -> le fait de mettre un astérisque permet d’ignorer lors du push tous les fichiers se terminant par l’extension `.txt`.

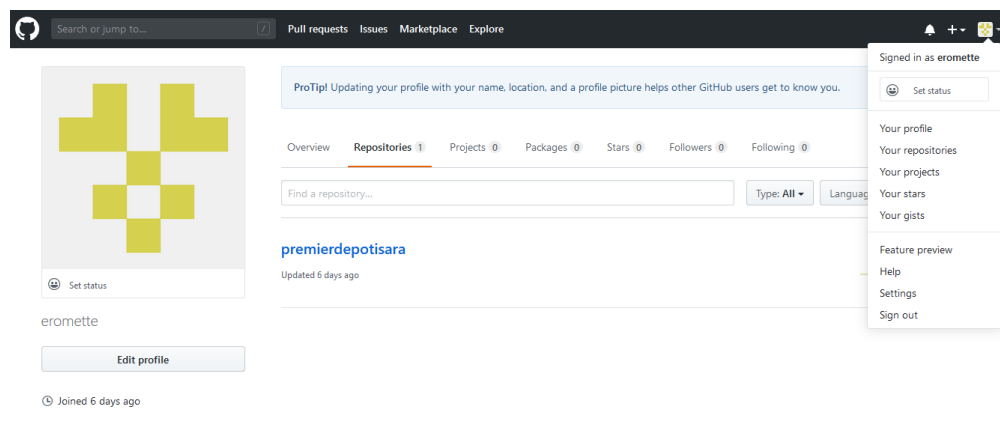
'Projet1.docx' -> cette ligne permettra de ne pas synchroniser avec le répertoire en ligne le fichier appelé `Projet1.docx`.

Par contre, `*.docx` fera que lors du push, tous les fichiers `.docx` seront ignorés.

Cela est pratique lors de la création en commun de dossiers contenant des fichiers R car celui-ci crée automatiquement des fichiers `.Rhistory` et ainsi, écrire dans le `.gitignore` `.Rhistory` ignorera automatiquement tous les fichiers `.Rhistory`.

“**Add a license**” permet d’ajouter une license au répertoire, c’est à dire indiquer aux personnes, qui téléchargent/utilisent le répertoire et ce qu’il contient, qu’ils doivent respecter les conditions de la License (CC BY, ...).

Une fois votre répertoire créé, celui-ci est disponible dans votre onglet *Repositories* et vous pouvez le gérer à votre guise.



2 Ajouter les membres de l'équipe

Pour ajouter des collaborateurs, il faut aller dans notre dépôt/repository. Une fois dedans :

1. Aller dans *Settings*
2. Puis dans *Manage Access*
3. Cliquer sur *Invite a collaborator*
4. Entrer le nom de notre collaborateur
5. Lui envoyer une invitation par mail

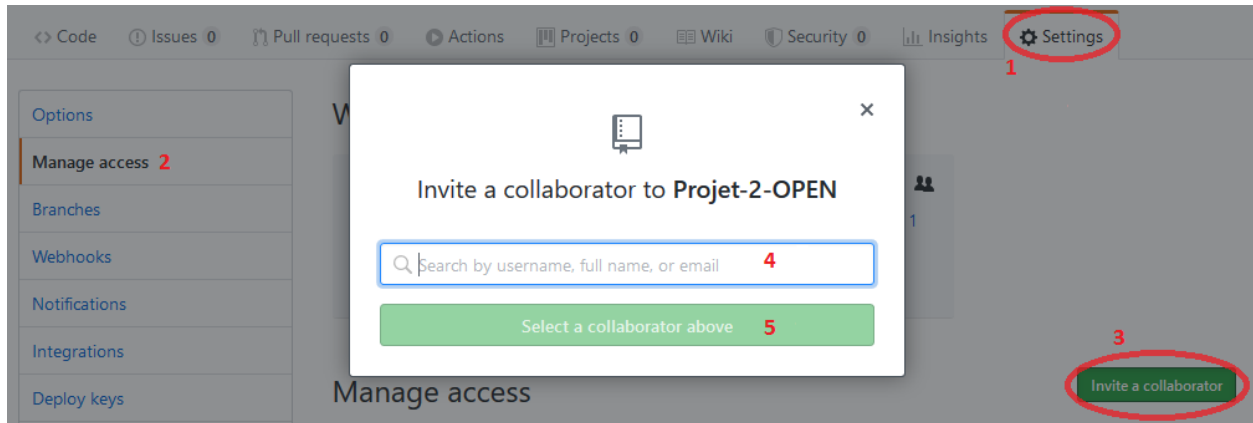


Figure 1: Ajout d'un collaborateur

Une fois que la personne aura accepté l'invitation, elle pourra utiliser le dépôt librement.

3 Gestion de fichiers par la console

Pour gérer les fichiers présents dans son dépôt, on utilise l'invite de commande et on commence par créer une copie locale de notre dépôt. Pour cela on ouvre l'invite de commandes de notre ordinateur et on accède au lieu où l'on veut stocker notre dépôt par la commande `cd` :

Par exemple : pour accéder au fichier Groupe2 dans ses documents, on ouvre son invite de commande et on tape `cd Documents` puis entrer et `cd Groupe2` et entrer

Une fois dans le dossier souhaité, on peut cloner le dépôt. Pour cela on utilise :

- **git clone** 'adresse du dépôt'

Une fois le dépôt cloné, on peut modifier les fichiers directement sur notre ordinateur.

Pour mettre à jour notre travail sur notre dépôt en ligne on commence par mettre à jour notre fichier local par un pull :

- **git pull**

Puis on fait un add, un commit et un push, pour le renvoyer sur git en ligne :

- **git add** 'nom du projet ou du fichier'

- `git commit -m "Commentaire"`
- `git push`

—> **Pensez à faire entrer après chaque ligne de commande !**

Normalement, si l'on vérifie en ligne on peut voir le fichier mis à jour.

A noter que pour supprimer un fichier par l'invite de commande, on utilise `rm` suivi du nom du fichier puis entrer.

4 *Gestion de fichiers par GitHub Desktop*

4.1 *Explications :*

Intéressons-nous à l'utilisation d'un logiciel permettant de simplifier l'usage de GitHub. En effet, la console peut sembler quelque peu... *perturbante* pour les néophytes. GitHub a par conséquent développé un outil simplifiant l'usage de Git : GitHub Desktop **tadaaaam**



Ce outil permet de remplacer l'utilisation de la console de commande par une interface intuitive de gestion.

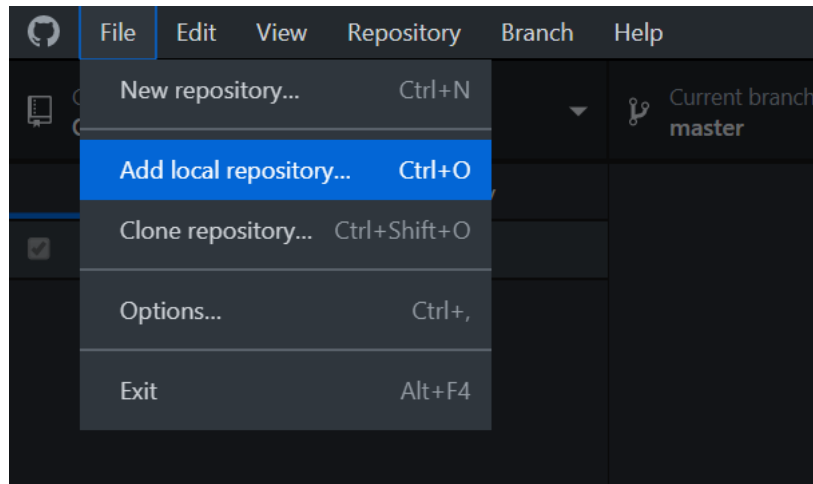
4.2 *Lancement du logiciel :*

Vous devez tout d'abord vous connecter à votre compte GitHub.

Ici, vous avez la possibilité de créer un répertoire pour ensuite le synchroniser avec votre compte GitHub via : **New repository**.

Il est par ailleurs possible d'ajouter un dossier local (sur votre ordi) comme répertoire en ligne GitHub via : **Add local repository**.

Enfin, option qui nous intéresse le plus : la capacité de cloner un répertoire depuis le compte en ligne GitHub sur votre ordinateur via **Clone repository**.



4.3 Cloner un répertoire :

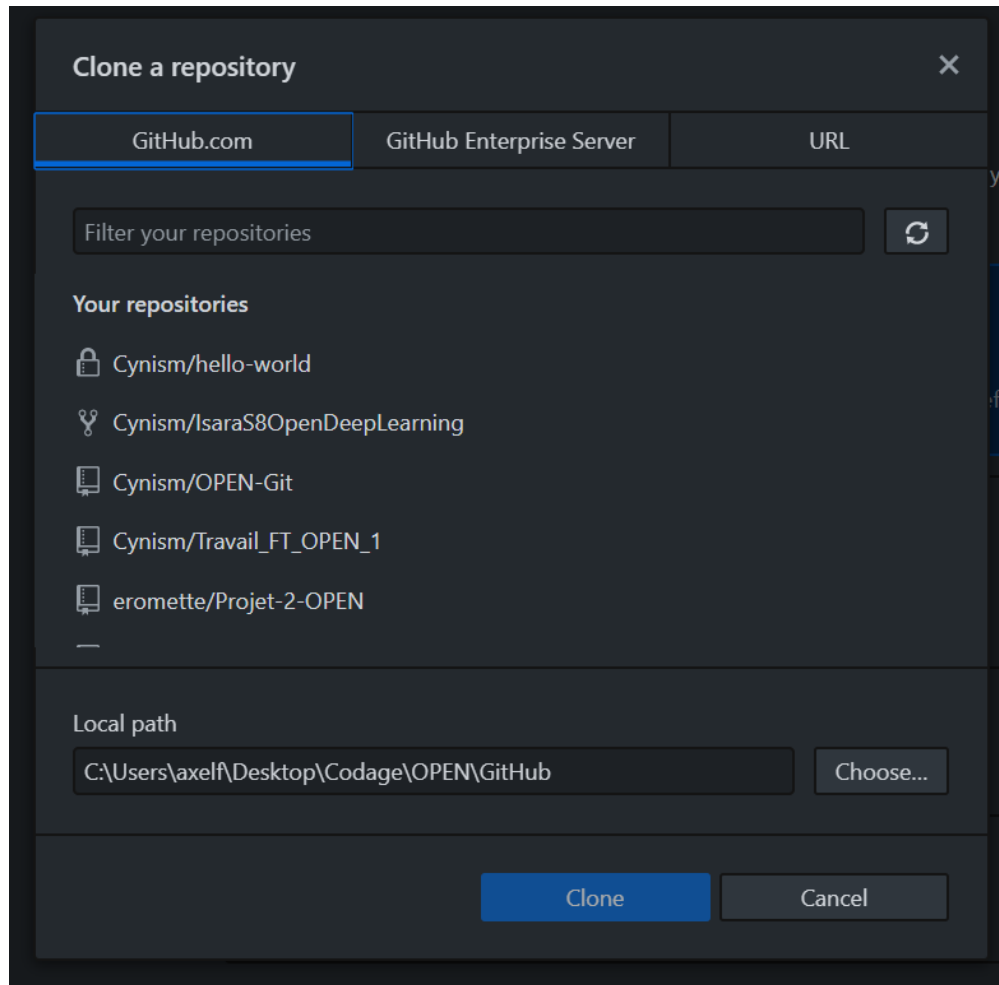
Lors du clonage d'un répertoire, vous avez 3 onglets :

- *GitHub.com* vous permet de sélectionner un répertoire que vous possédez déjà sur votre compte en ligne, que ce soit un répertoire à vous, un où vous êtes autorisés à éditer ou un *fork*, c'est à dire un suivi des mises à jour d'un répertoire d'une personne, sans pour autant y avoir la possibilité de l'éditer.

Une fois le répertoire sélectionné, vous pouvez choisir l'emplacement d'enregistrement via *Local path* puis *Browse*.

- *GitHub Enterprise Server* sert dans le cadre d'une entreprise, nous ne nous y intéressons pas.

- *URL* vous permet de faire comme le premier onglet sauf qu'au lieu de choisir de cloner un répertoire venant de votre compte en ligne, vous avez la possibilité de directement le cloner via un lien qui vous a été fourni par le propriétaire.



4.4 *Commit de fichiers :*

Une fois votre répertoire ajouté, vous pouvez le sélectionner en haut à gauche via la petite flèche dans la liste *Current repository*.

L'avantage de ce logiciel est qu'il enregistre en temps réel les modifications effectuées dans le répertoire. Ainsi, il est possible de voir sur cet exemple des fichiers qui ont été :

- supprimés : le carré rouge
- modifiés : le carré orange
- ajoutés : le carré vert

Il est possible de choisir, via les case à cocher sur le côté à gauche, les fichiers qui seront synchronisés entre le répertoire local et le répertoire en ligne accessible à tout ceux qui ont les droits.

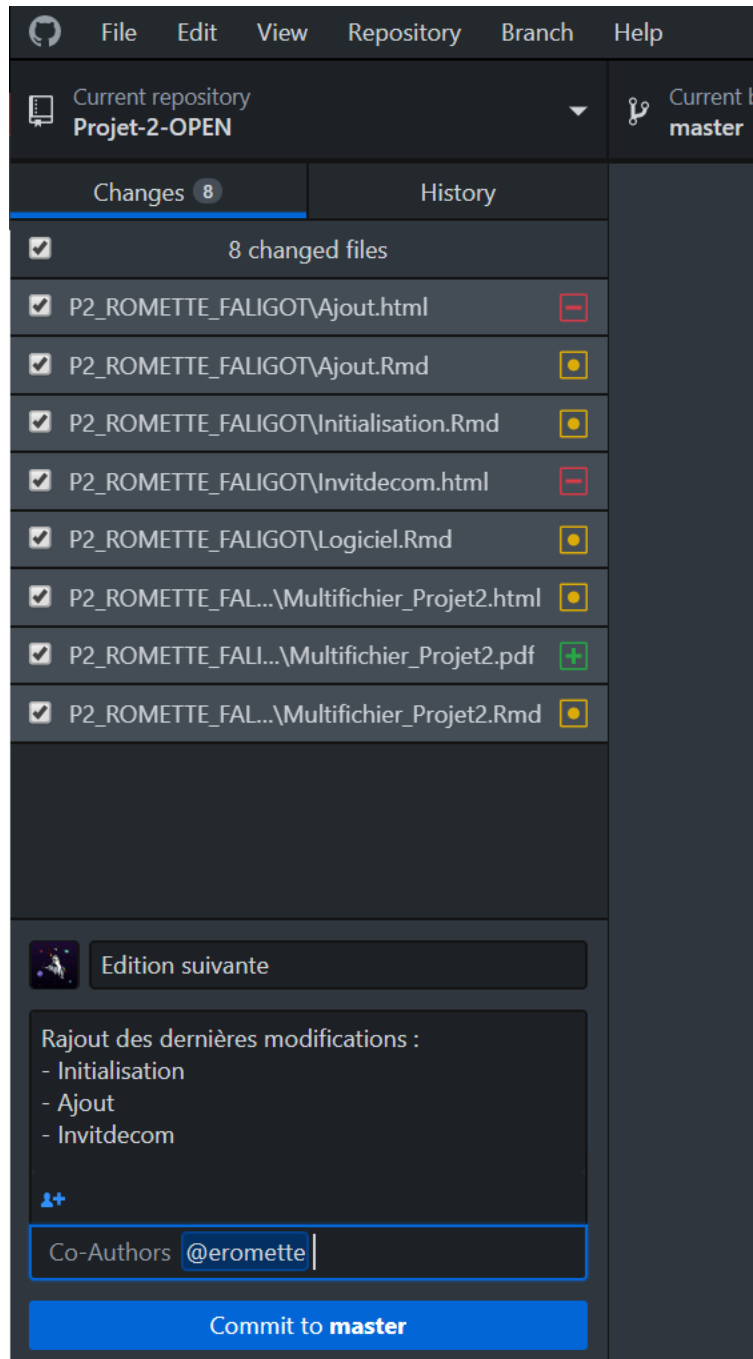
En bas se trouve une zone permettant d'effectuer les *commits*, c'est à dire les indications de mise à jour des fichiers du répertoire.

Ainsi, ici on observe que l'utilisateur *Cynism* a effectué un commit :

- contenant tous les fichiers cochés au dessus

- ayant pour titre : *Edition suivante*
- ayant pour description détaillée le détail des modifications
- ayant pour co-auteur : *eromette*, lui permettant d'apparaître dans l'historique des commits.

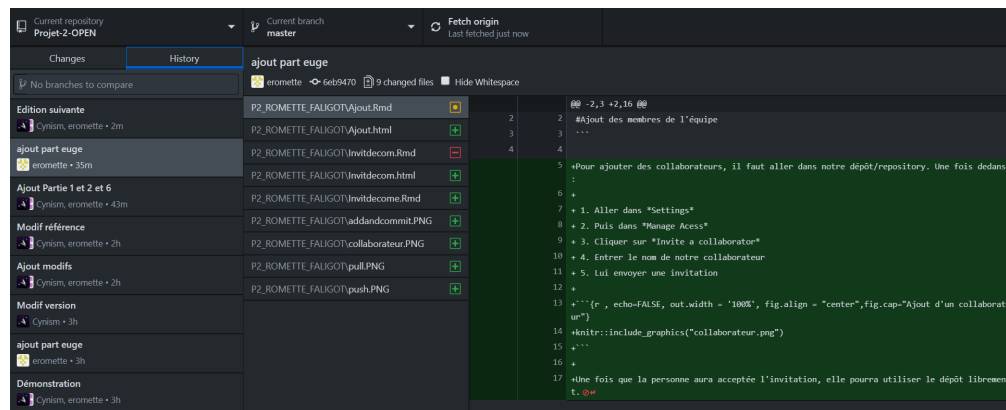
Une fois ceci rempli, il est possible de lancer le commit, ici dans la branche *master*. Pour synchroniser le commit et les fichiers avec la version en ligne, il est nécessaire de *push* (expliqué peu après).



4.5 Historique des commits :

L'interface permet de voir l'historique des changements effectués sur le répertoire :

- les noms des auteurs
- la date/heure
- le nom du commit
- le contenu par commit (les changements apportés)
- les changements apportés au sein de tous les fichiers par commit
- en vert : ce qui a été ajouté
- en rouge : ce qui a été enlevé
- en gris : ce qui n'a pas été modifié



4.6 Le principe de “branch” :

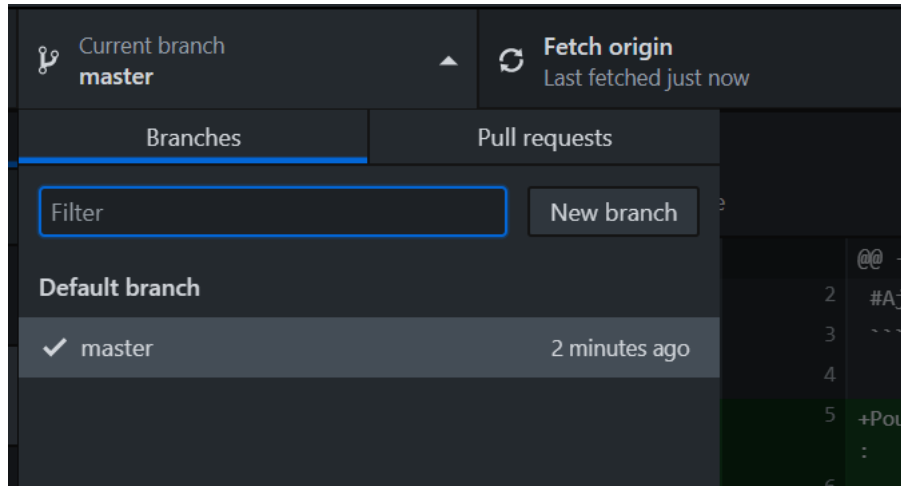
Lors d’une édition par plusieurs personnes d’un même répertoire, dans le cas où l’on souhaiterait partir sur des versions différentes avec les mêmes bases, ou versions non finies nécessitant encore des modifications, il est possible de créer des *branches*.

La principale se nomme *master* et contient le fichier principal.

Il est possible d’en créer une (ou d’en sélectionner une) en cliquant sur *Current branch* puis en choisissant celle que l’on veut éditer ou en créer une.

Lorsque l’on souhaite fusionner 2 branches, il suffit de faire une *Pull request*, c’est à dire une demande de fusion des branches.

Il y aura toutefois auparavant une annonce sur l’accueil du logiciel comme quoi il est possible de fusionner des branches, et qu’il faut choisir laquelle sera fusionnée à laquelle.



4.7 Effectuer un push/pull :

4.7.1 Push :

Lorsqu'un commit a été effectué, il suffit de cliquer, dans le menu principal, sur le bouton qui s'appelle *Push origin*.

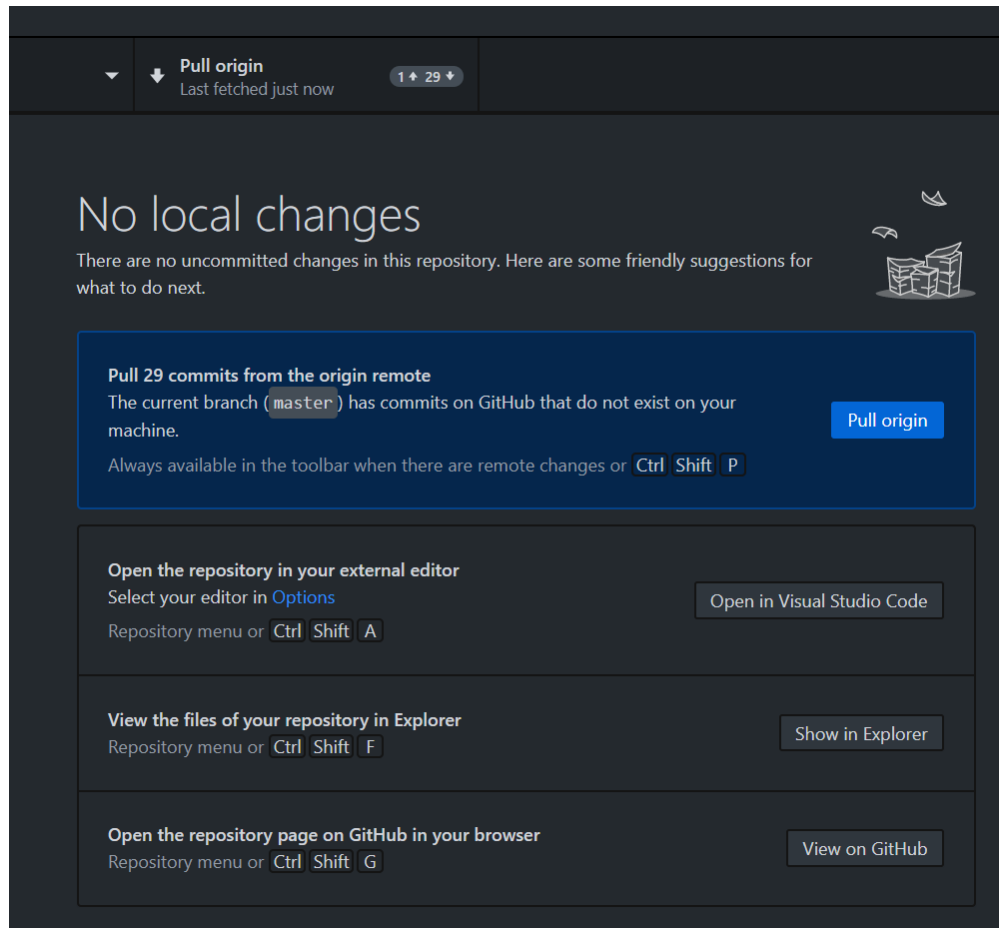
Cela permettra de synchroniser le commit avec le répertoire en ligne et l'afficher dans l'historique.

Sur l'image, il est indiqué *Pull origin*, toutefois cela peut aussi être *Push* ou *Fetch origin*, le premier permettant d'envoyer, le second de vérifier des changements.

4.7.2 Pull :

Lorsque des modifications ont été apportées en ligne, il est nécessaire de récupérer les dernières versions via : *Pull origin*.

Il est possible qu'il y ait des conflits, en ce cas le plus simple est d'aller dans le dossier concerné (manuellement), puis de supprimer le(s) fichier(s) qui pose souci, ou du moins le déplacer sur le Bureau si on souhaite le conserver.



5 *Références*

<https://ndpsoftware.com/git-cheatsheet.html#loc=index;>
<https://education.github.com/git-cheat-sheet-education.pdf>
<https://desktop.github.com/>
<https://github.com/>