2024

Utilisation de Git avec PyCharm

CHERIFA BEN KHELIL

UNE PARTIE DU TEXTE DE CE DOCUMENT A ETE GENEREE A L'AIDE D'UNE IA GENERATIVE. |

Table des matières

Utilisation de Git avec PyCharm : Guide étape par étape

Prérequis

- Installer **PyCharm Community Edition** (https://www.jetbrains.com/fr-fr/pycharm/download/).
- Avoir **Git** installé sur votre machine (https://git-scm.com/downloads).
- Avoir un compte sur la plateforme **GitHub** ou **GitLab** (https://github.com/).

---

1. Approches pour connecter PyCharm à GitHub

Il existe plusieurs façons de lier et d'utiliser vos projets avec **GitHub** et **PyCharm**, selon votre méthode de travail préférée :

1. **Créer le projet sur GitHub**, puis le **cloner dans PyCharm** pour commencer à travailler localement.
2. **Créer le projet dans PyCharm**, puis l'**envoyer sur GitHub** en créant un repository à ce moment-là.
3. **Créer le projet dans PyCharm** et le **lier à Git dès sa création**, pour configurer rapidement le suivi des versions et intégrer GitHub dès le départ.

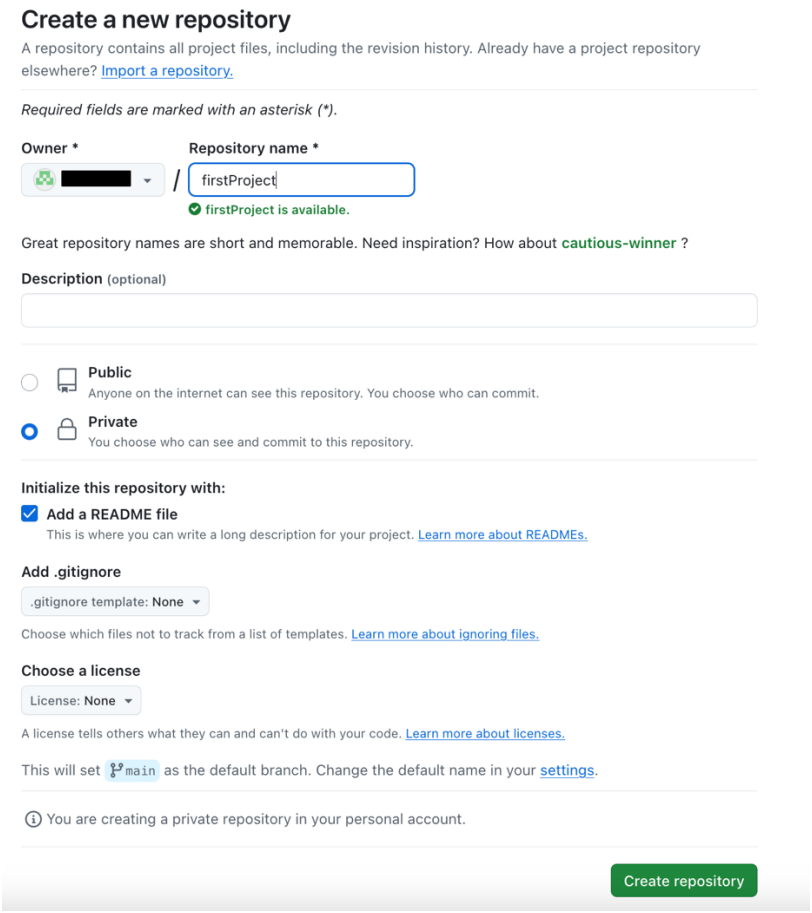
Dans ce document, nous présenterons en détail les étapes nécessaires pour mettre en œuvre chacune de ces méthodes.

1.1. Créer le projet sur GitHub, puis le cloner dans PyCharm

Cette méthode commence sur **GitHub** et implique les étapes suivantes :

Étape 1 : Créer le repository sur GitHub

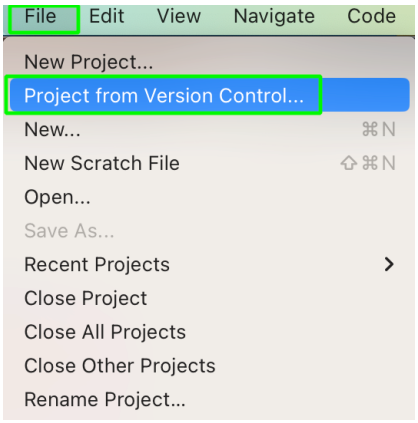
1. Connectez-vous à votre compte **GitHub**.
2. Créez un **nouveau dépôt en** cliquant sur le bouton "**New Repository**".
3. Renseignez un nom pour le repository (*il est recommandé de le créer en privé*) et, si nécessaire, configurez les options requises, telles que l'ajout d'un fichier **README**, .**gitignore** ou d'une licence.
4. Cliquez sur **Create repository** pour finaliser cette étape.



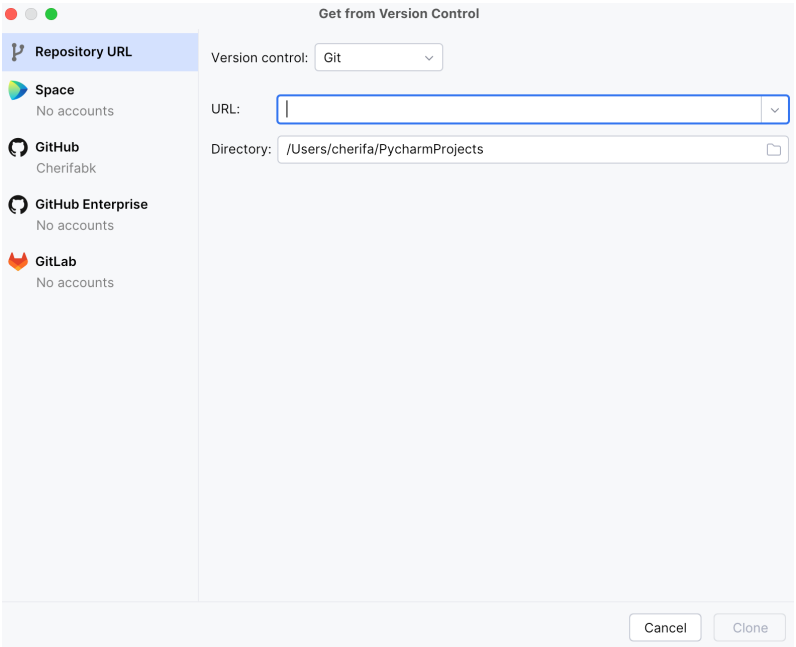
5. Vous pouvez maintenant copier l'URL du dépôt distant (par exemple, ***https://github.com/votre-utilisateur/nom-du-depot.git***) pour lier votre dépôt local ou pour cloner le dépôt.

Étape 2 : Cloner le repository dans PyCharm

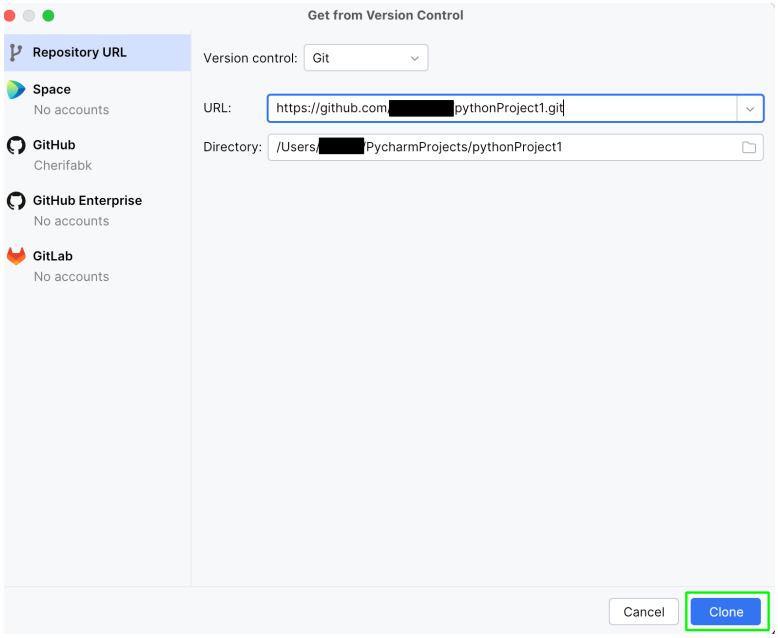
6. Ouvrez **PyCharm**.
7. Depuis l'écran d'accueil, cliquez sur "**Get from Version Control**" (*ou allez dans **File** > **Project from Version Control** > **Git***).



8. Dans le champ "**URL**", collez l'URL de votre repository **GitHub** que vous avez copiée.



9. Sélectionnez un dossier local où cloner le projet et cliquez sur "**Clone**".



10. Le projet sera importé dans **PyCharm**, vous permettant ainsi de le modifier et de le synchroniser avec **GitHub**. Votre dépôt local est lié au dépôt distant.

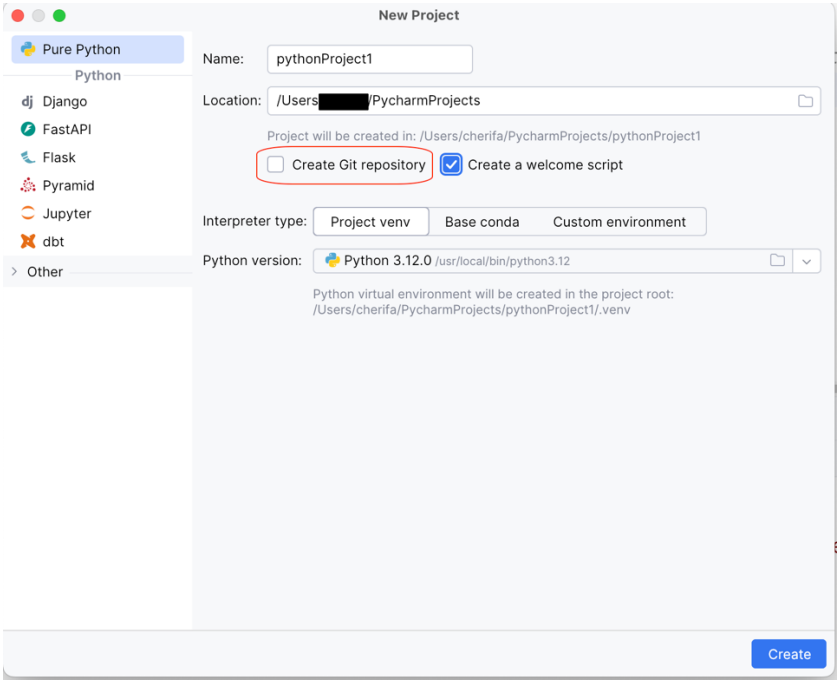
1.2. Créer le projet dans PyCharm, puis l'envoyer sur GitHub

Avec cette méthode, le projet est d'abord créé localement, puis exporté vers **GitHub** :

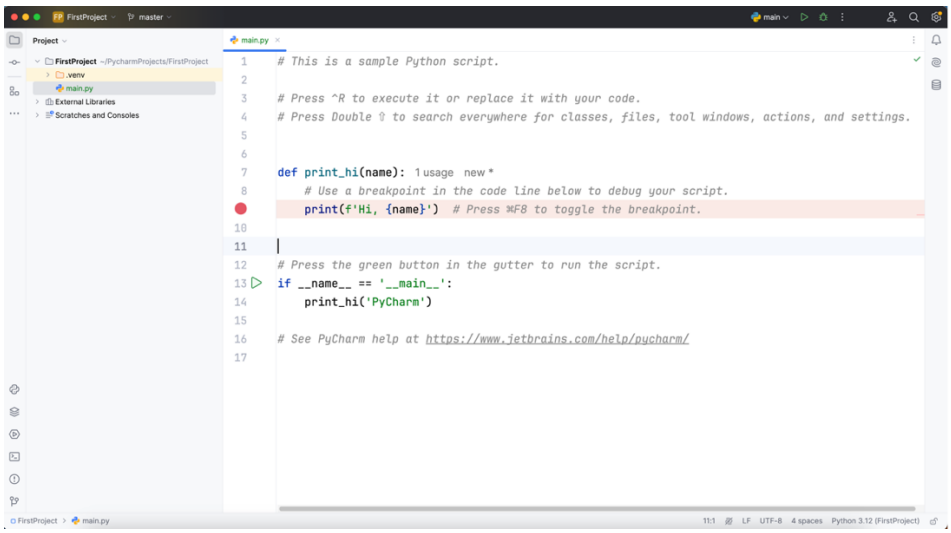
Étape 1 : Créer un projet local dans PyCharm

1. Ouvrez **PyCharm** et sélectionnez "**New Project**".

2. Donnez un nom à votre projet et choisissez son emplacement.
3. Désélectionnez l'option "**Create Git repository**".



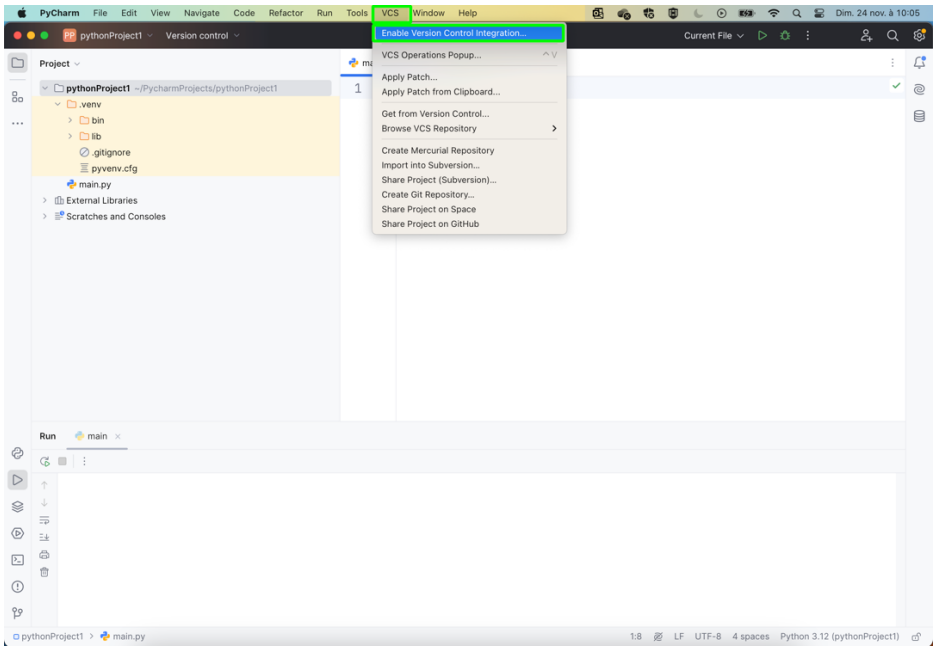
4. Créez votre projet.
5. Si vous avez coché la case "**Create a welcome script**", un tableau de bord s'ouvrira, et le fichier **main.py** sera créé automatiquement avec un script de démarrage. Si vous n'avez pas coché cette option, vous pouvez créer le fichier manuellement en suivant ces étapes : cliquez sur ***File > New > Python File***, puis nommez le fichier **main.py** (*ou choisissez un autre nom de votre choix*).



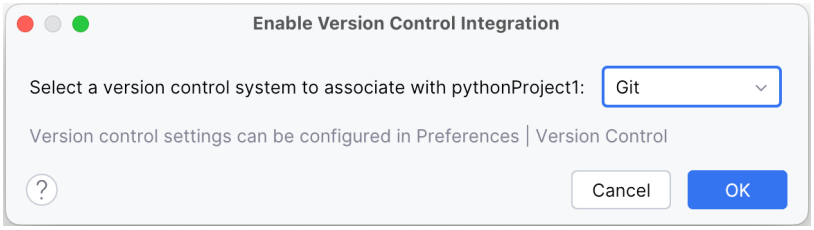
Sur la partie gauche de l'interface, vous trouverez ***la structure de votre projet***, qui répertorie tous les dossiers et fichiers présents dans votre répertoire de projet.

Étape 2 : Configurer Git et envoyer le projet sur GitHub

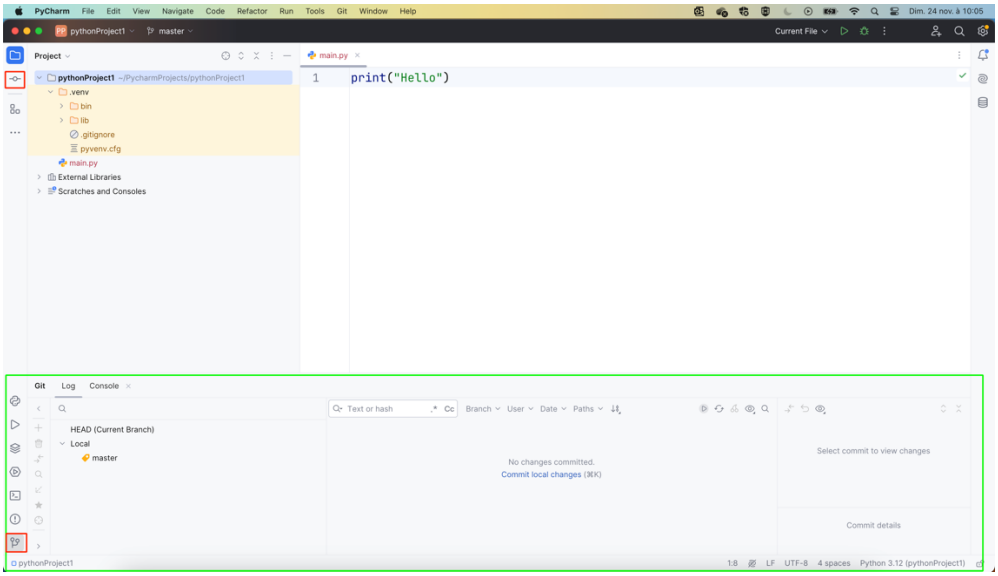
6. Une fois le projet créé, allez dans la barre de menu et sélectionnez **VCS** > **Enable Version Control Integration**.



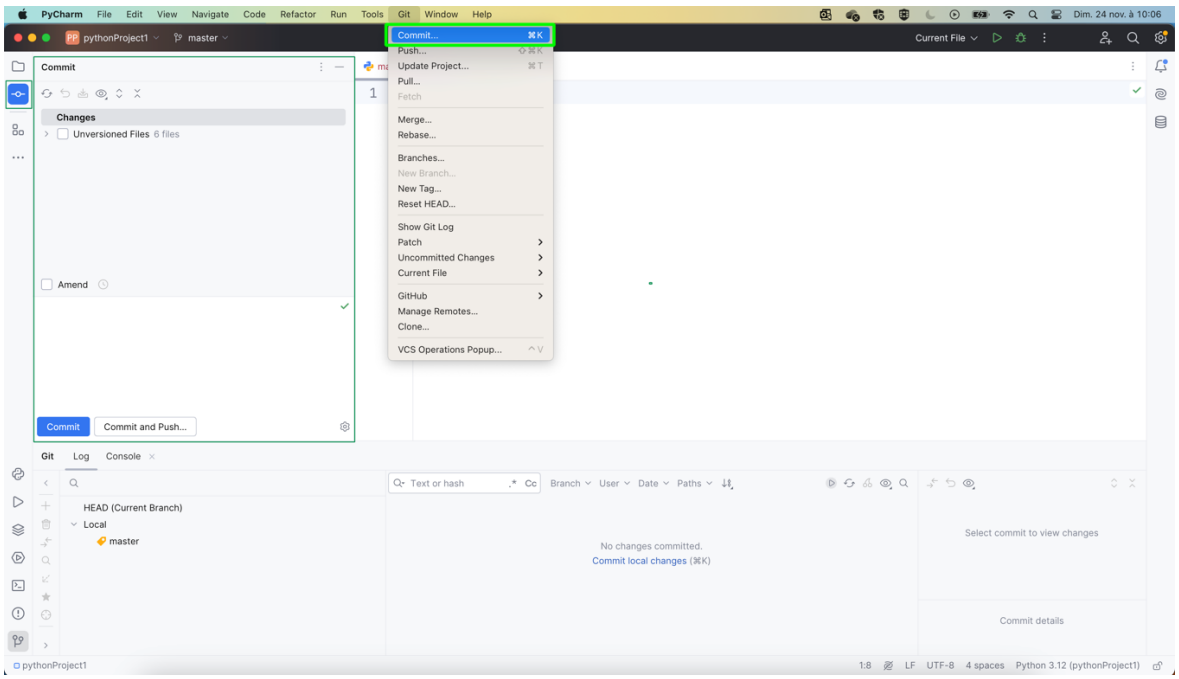
7. Dans la fenêtre qui apparaît, choisissez **Git** comme système de versionnement, puis cliquez sur **OK**.

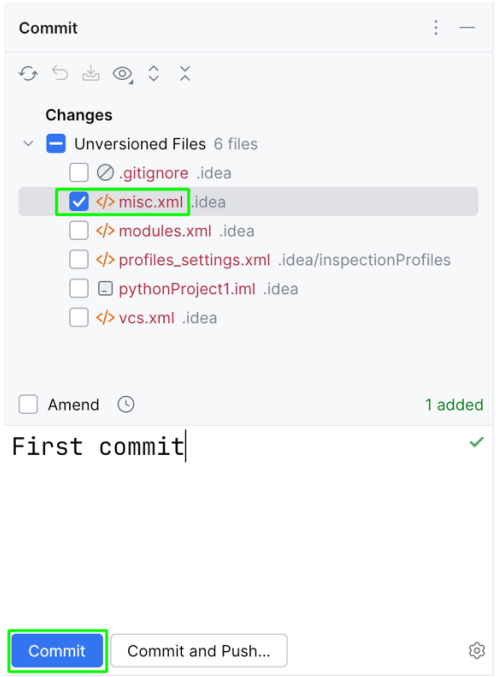


8. **PyCharm** initialise alors un dépôt Git dans le dossier de votre projet.

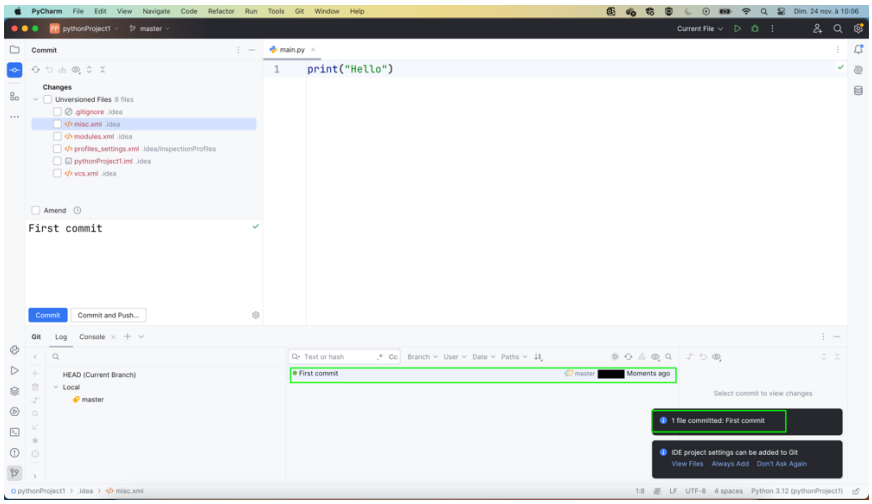


9. Faites un premier commit : cliquez sur "**Commit**" et ajoutez tous les fichiers que vous voulez inclure dans votre dépôt.

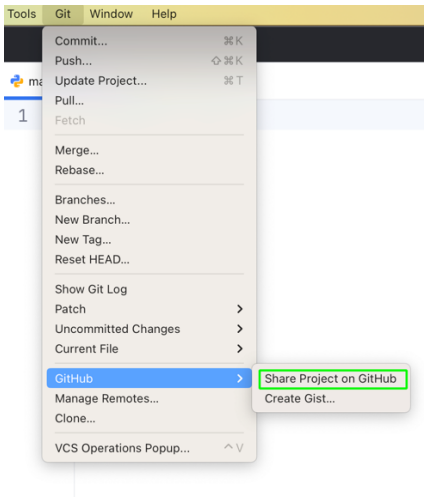




10. Cliquez sur "**Commit**" pour valider. Le commit apparaîtra dans l'historique des commits de **PyCharm**.



11. Allez dans **Git** > **GitHub** > **Share Project on GitHub**.



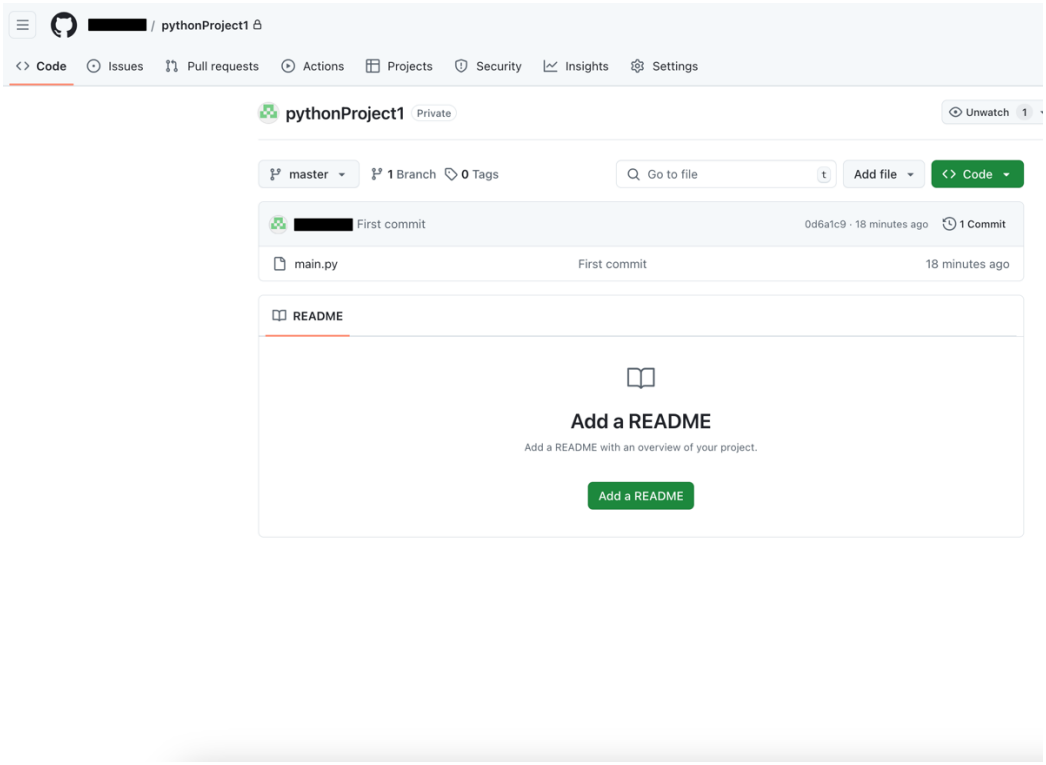
12. Une fenêtre apparaîtra où vous pouvez entrer les informations de votre dépôt :

- **Nom du dépôt** : Choisissez un nom pour votre dépôt (par défaut, le nom du projet est utilisé).
- **Description** : Entrez une description si nécessaire.
- **Visibilité** : Sélectionnez si le dépôt doit être public ou privé.

13. Donnez un nom au repository et cliquez sur "**Share**".

Remarque : Si vous n'êtes pas connecté à **GitHub** dans **PyCharm**, vous serez invité à vous connecter.

14. Le projet local sera automatiquement envoyé sur **GitHub**.

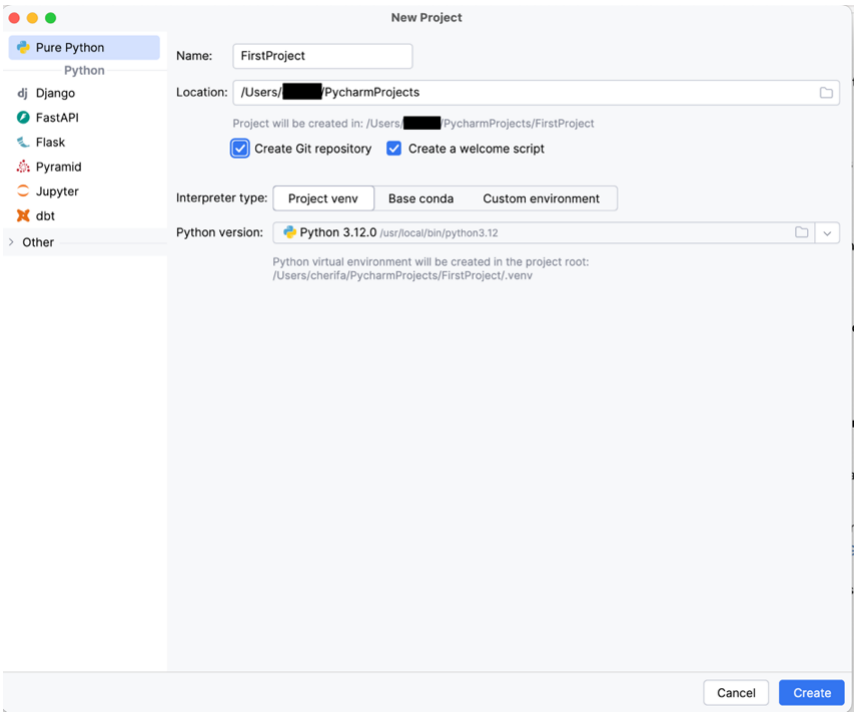


1.3. Créer le projet dans PyCharm et le lier à Git dès sa création

Cette méthode configure Git dès le début :

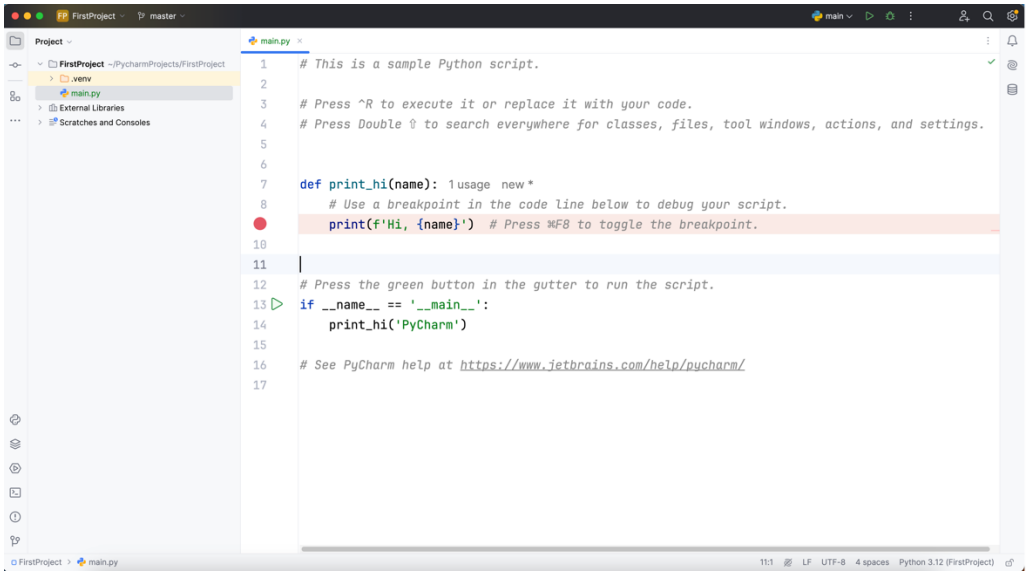
Étape 1 : Créer le projet avec un repository Git local

1. Ouvrez **PyCharm** et sélectionnez **New Project**.
2. Donnez un nom à votre projet et choisissez son emplacement.
3. Cochez la case "**Create Git repository**" pour initialiser un dépôt Git dans votre projet. Vous pouvez également cocher la case "**Create a welcome script**" si vous souhaitez inclure un script de démarrage, mais cette option est facultative.



4. Cliquez sur **Create**.

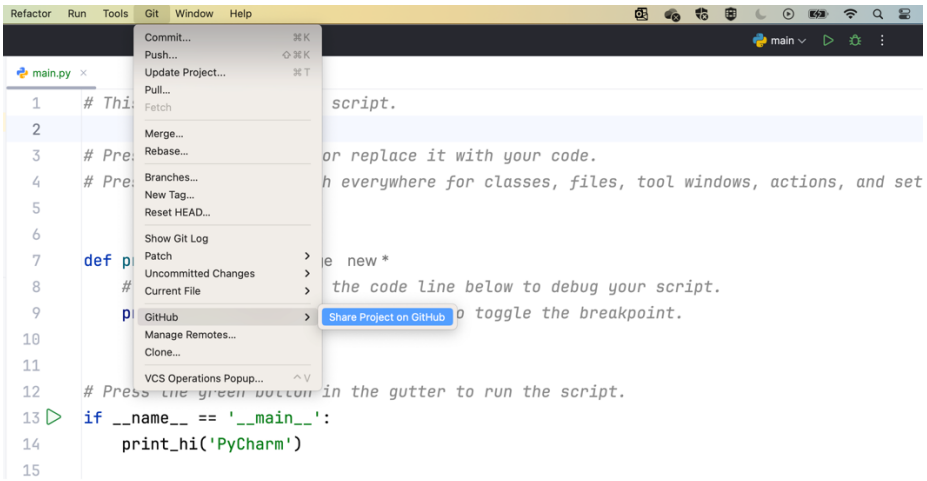
5. Si vous avez coché la case "**Create a welcome script**", un tableau de bord s'ouvrira, et le fichier **main.py** sera créé automatiquement avec un script de démarrage. Si vous n'avez pas coché cette option, vous pouvez créer le fichier manuellement en suivant ces étapes : cliquez sur ***File > New > Python File***, puis nommez le fichier **main.py** (*ou choisissez un autre nom de votre choix*).



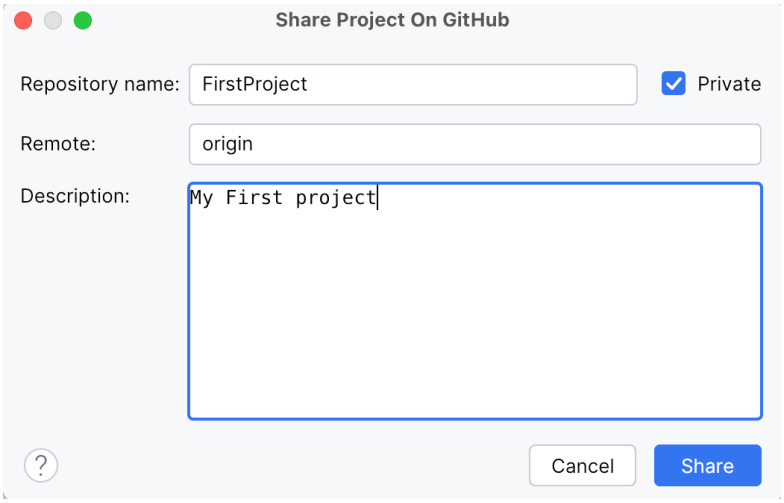
Sur la partie gauche de l'interface, vous trouverez ***la structure de votre projet***, qui répertorie tous les dossiers et fichiers présents dans votre répertoire de projet.

Étape 2 : Partager votre projet local avec un dépôt distant GitHub

6. Dans la barre d'outils de PyCharm, allez dans **Git** > **GitHub** > **Share Project on GitHub**.



7. Une fenêtre apparaîtra où vous pouvez entrer les informations de votre dépôt :
   - **Nom du dépôt** : Choisissez un nom pour votre dépôt (par défaut, le nom du projet est utilisé).
   - **Description** : Entrez une description si nécessaire.
   - **Visibilité** : Sélectionnez si le dépôt doit être public ou privé.

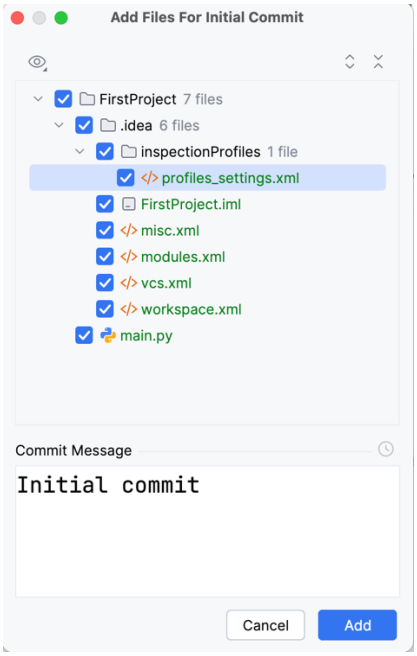


8. Cliquez sur **Share**.

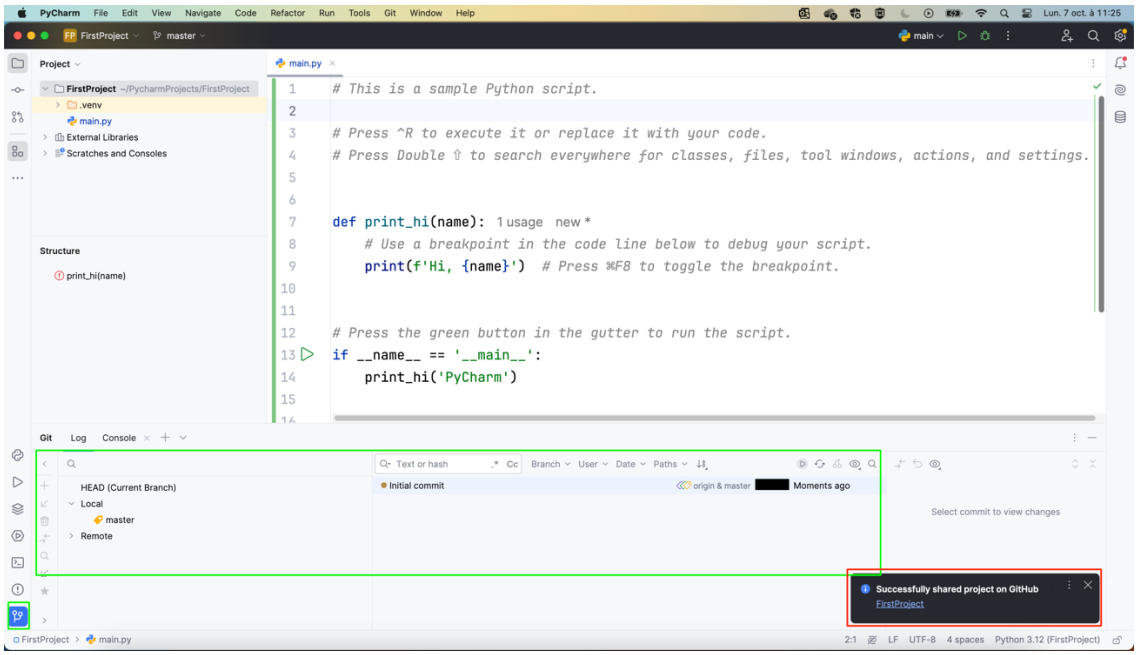
Remarque : Si vous n'êtes pas connecté à **GitHub** dans **PyCharm**, vous serez invité à vous connecter.

9. Une fenêtre s'ouvrira pour confirmer les fichiers à ajouter au dépôt.

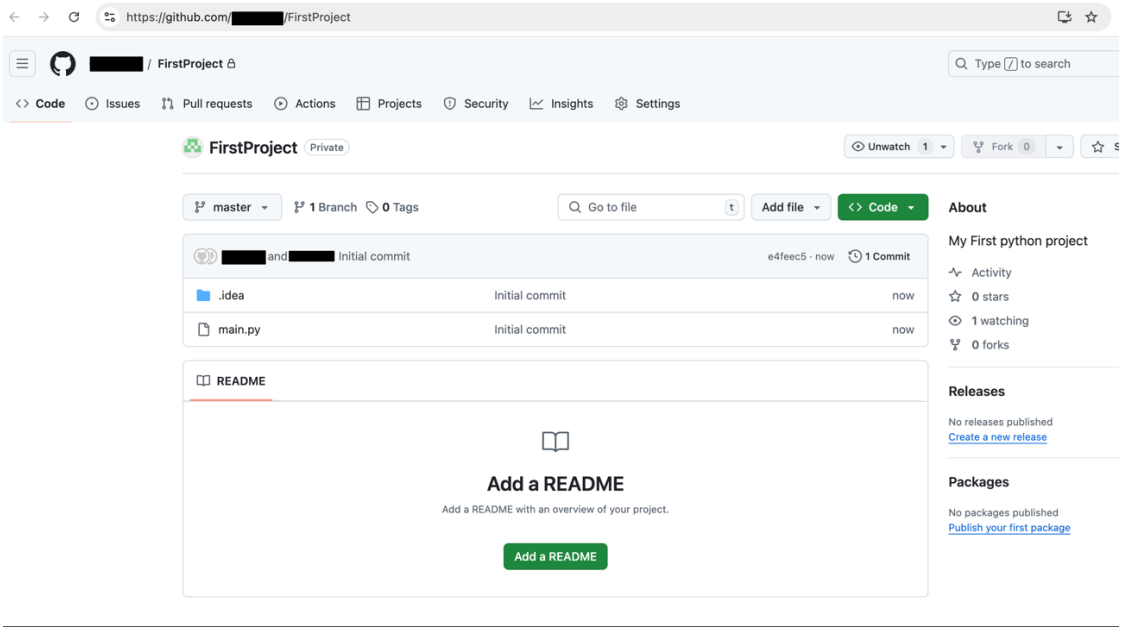
10. Sélectionnez les fichiers que vous souhaitez inclure, ajoutez un message de commit (par exemple, "*Initial commit*"), puis cliquez sur **Add and Commit**.



11. Votre premier commit est maintenant enregistré. Pour visualiser l'historique des commits, cliquez sur l'icône Git située à gauche. Cela vous permettra d'afficher les dépôts ainsi que l'historique de vos commits.

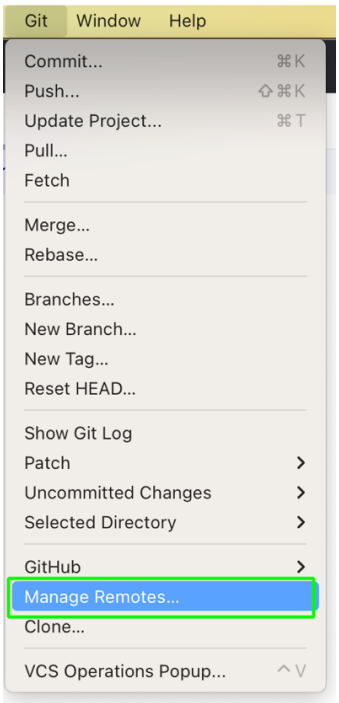


12. Votre projet ainsi que vos premiers changements sont maintenant partagés sur GitHub, et vous pouvez y accéder directement via l'interface GitHub.

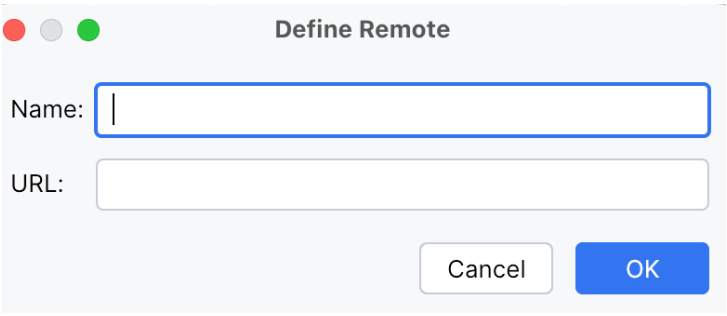


Précision : Si vous avez déjà créé votre repository **GitHub**, vous pouvez établir le lien directement en ajoutant l'URL du repository dans **PyCharm**. Contrairement à l'option "Share", qui vous permet de partager le projet avec d'autres utilisateurs ou de publier le code, cette méthode lie simplement votre projet local à GitHub pour la gestion du versionning :

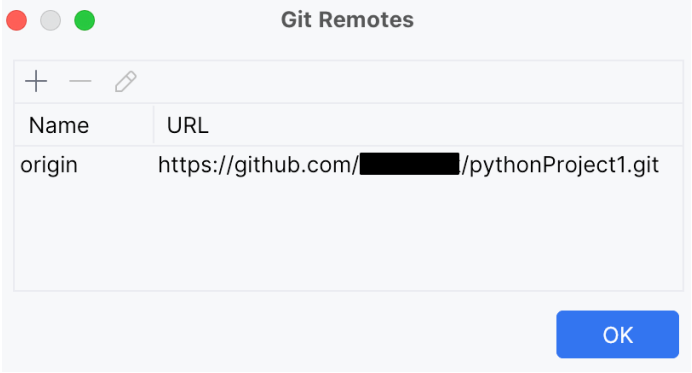
1. Allez dans **Git > Manage Remotes**.



2. Cliquez sur "**Add**" et entrez l'URL de votre repository GitHub que vous avez créé au préalable.



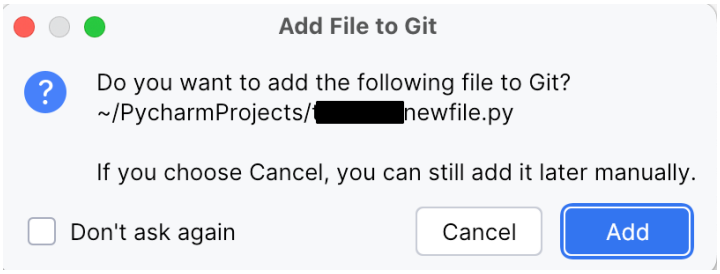
3. Validez pour établir la connexion entre votre repository local et GitHub.



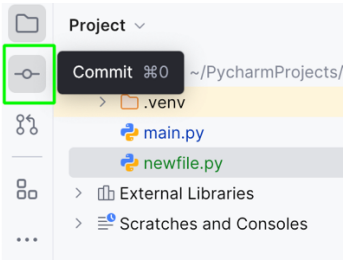
2. Gestion des commits, push et pull entre PyCharm et GitHub

2.1. Créer un nouveau Commit

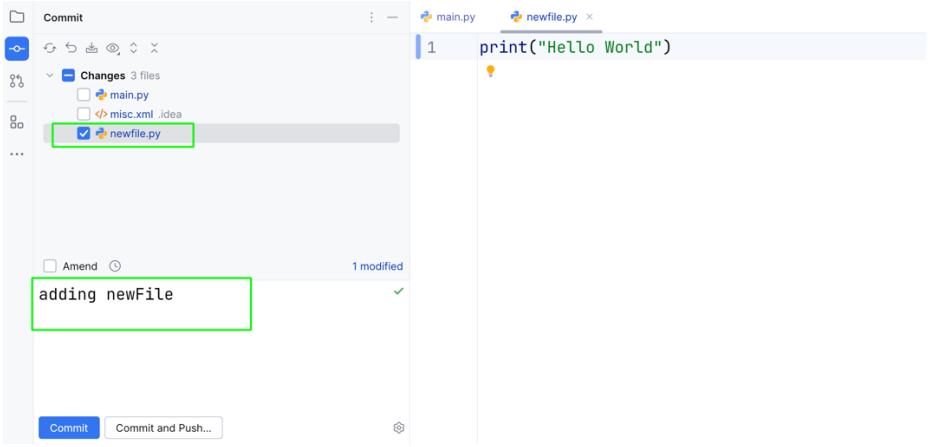
1. À chaque nouvel ajout de fichier dans votre projet, une fenêtre apparaîtra pour vous demander si vous souhaitez ajouter le fichier au suivi Git.



2. Sinon, vous pouvez réaliser cette opération plus tard. Dans l'onglet **Project**, faites un clic droit sur le fichier ou le dossier concerné, puis sélectionnez **Git > Add** pour ajouter vos fichiers au suivi Git.
3. Allez dans **Git > Commit** ou cliquez simplement sur l'icône Commit dans le menu vertical à côté de l'onglet Project.



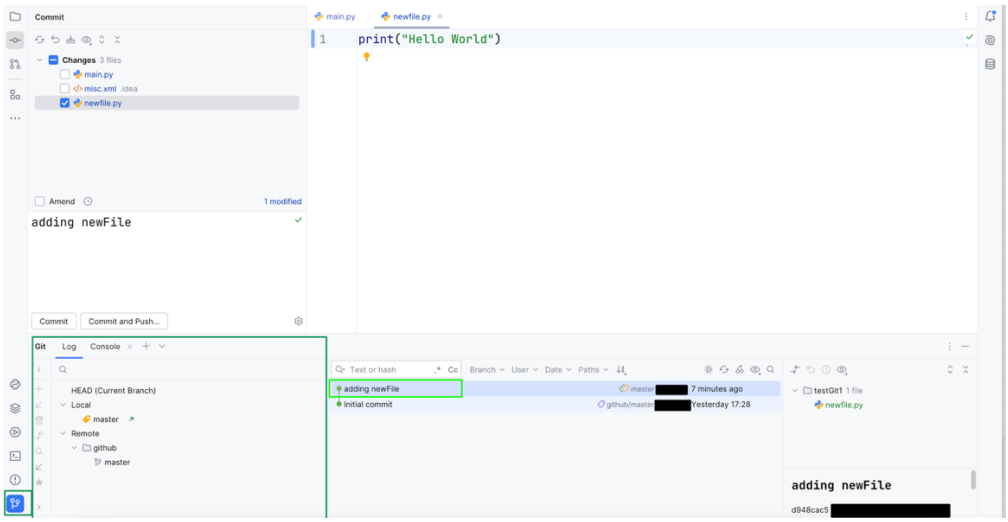
4. Une fenêtre de *commit* s'ouvre. ***Sélectionnez*** les fichiers que vous souhaitez inclure dans votre commit, puis ajoutez un message de commit décrivant les changements (par exemple : "*Second commit*" ou "*adding newFile.py*").



5. Cliquez sur **Commit** pour valider les changements. Vous avez maintenant enregistré votre nouveau *commit*.
6. Attention, le bouton **Commit and Push** combine deux actions : le commit local et l'envoi vers le dépôt distant :

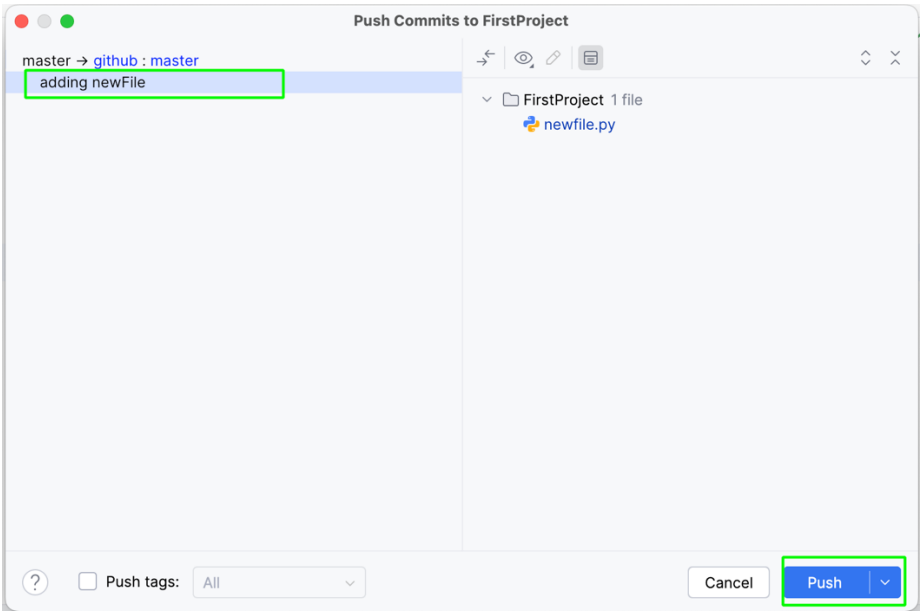
   Commit : *PyCharm enregistre vos modifications dans le dépôt local, avec un message décrivant les changements avec le message associé.*

   Push : *Après le commit, PyCharm envoie automatiquement ces modifications vers le dépôt distant (par exemple GitHub), mettant ainsi à jour le dépôt distant avec vos nouveaux commits.*
7. Vous pouvez visualiser l'historique des commits en cliquant sur l'icône Git, située tout en bas du menu vertical à gauche de l'écran.

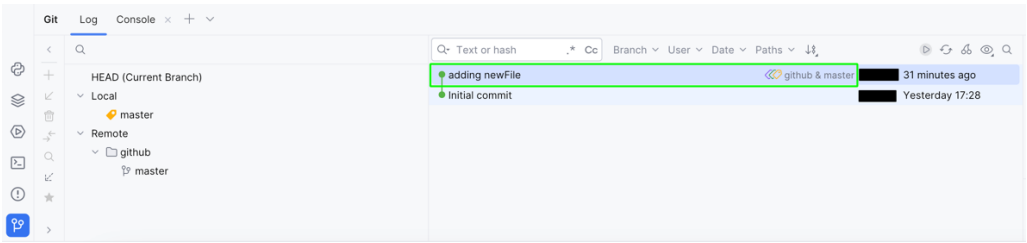


2.2. Effectuer un Push

1. Dans **PyCharm**, allez dans **Git** > **Push**.
2. Vérifiez que votre ***commit*** est prêt à être envoyé, puis cliquez sur **Push** pour envoyer votre code au dépôt distant.



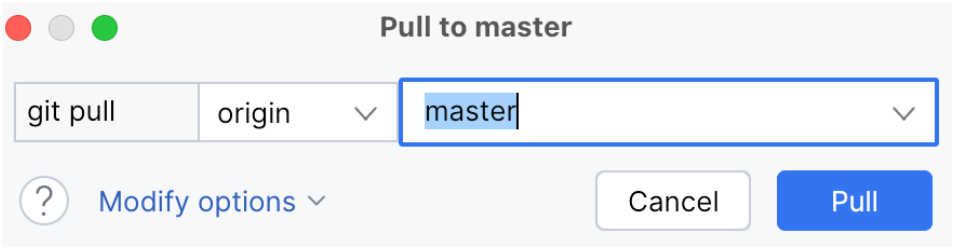
3. Vous pouvez observer le changement dans l'historique des commits du dépôt distant, après le push, confirmant que les modifications ont bien été envoyées.



2.3. Effectuer un Pull

Si vous souhaitez récupérer les dernières modifications apportées à votre projet sur **GitHub**, vous devez effectuer un ***pull*** pour synchroniser votre projet local avec le repository distant. Voici les étapes détaillées :

1. Dans **PyCharm**, allez dans **Git** > **Pull**.
2. Sélectionnez la branche que vous souhaitez mettre à jour, puis cliquez sur "Pull"



Cela récupérera les derniers changements effectués sur le repository distant, que ce soit par d'autres utilisateurs ou depuis un autre appareil. Le pull permet de mettre à jour votre projet local en fusionnant les nouvelles modifications du repository distant avec votre travail actuel. Si des conflits surviennent, **PyCharm** vous avertira et vous proposera des options pour résoudre ces conflits avant de finaliser l'intégration.

2.4. Étapes Suivantes : Travailler avec Git

- **Pour effectuer de nouveaux commits** : Après avoir modifié un fichier, allez dans **Git** > **Commit**, ajoutez un message de commit, puis validez.
- **Pour synchroniser les changements** : Utilisez les options **Push** pour envoyer les changements, et **Pull** pour récupérer les dernières modifications du dépôt distant.

3. Étapes à suivre pour gérer efficacement le travail sur le projet

Voici un scénario illustrant la gestion des commits, ***pull*** et ***push*** sur un projet **PyCharm** entre ***Alice*** et ***Jean*** :

3.1. Initialisation du dépôt Git

- **Jean** crée un dépôt distant sur **GitHub**, puis ajoute le projet initial.
- **Jean** envoie ensuite ce projet vers **GitHub** avec un **push** initial.
- **Alice** clone ce dépôt distant en utilisant **PyCharm** : **File** > **Project from Version Control** > **Git**, puis entre l'URL du dépôt de **Jean** pour obtenir la version la plus récente du projet.

3.2. Commits

- **Alice** travaille sur une fonctionnalité et effectue plusieurs modifications. Une fois ses modifications terminées, elle effectue un **commit** local avec un message explicite, par exemple : **"Ajout de la fonctionnalité de gestion des utilisateurs"**.
- **Jean**, de son côté, travaille également sur une autre fonctionnalité et fait un **commit** local, avec un message du type : **"Amélioration de la gestion des permissions"**.

3.3. Pull avant de reprendre le travail

- Avant de commencer à travailler le matin, **Jean** fait un **pull** pour s'assurer qu'il dispose des dernières modifications apportées par **Alice** : **Git** > **Pull**. Cela lui permet d'intégrer ses modifications avec celles d'Alice et d'éviter d'éventuels conflits.
- Idem pour **Alice** : avant de commencer à travailler sur ses nouvelles fonctionnalités, elle fait un **pull** pour obtenir les dernières modifications envoyées par **Jean**.

3.4. Push après avoir commité

- Une fois que **Alice** a terminé ses modifications et a effectué son commit, elle fait un **push** pour envoyer ses changements vers le dépôt distant sur **GitHub** : **Git** > **Push**. Cela permet à **Jean** de récupérer ses modifications et de les intégrer dans son propre environnement de travail (avec un pull plus tard).
- De son côté, **Jean** effectue également un **push** après avoir terminé ses changements, afin que **Alice** puisse récupérer ses modifications les plus récentes.

3.5. Gestion des conflits

Imaginons que **Alice** et **Jean** aient tous deux modifié le même fichier (par exemple, *gestion_utilisateurs.py*). Si **Jean** fait un **push** avant **Alice**, cette dernière pourrait rencontrer un conflit lorsqu'elle tente de faire un **pull**.

- **Alice** reçoit une notification de conflit lors du **pull**. **PyCharm** lui permet de résoudre ce conflit en choisissant quelle version conserver, ou en fusionnant les modifications manuellement.

- Une fois le conflit résolu, **Alice** effectue un nouveau **commit** pour enregistrer la résolution et effectue un **push** pour mettre à jour le dépôt distant.

3.6. Pratique recommandée

- **Jean** et **Alice** doivent toujours **committer régulièrement** pour sauvegarder leurs progrès, afin d'éviter de trop grandes différences entre leurs versions locales.
- Ils doivent également **pousser fréquemment** leurs changements vers le dépôt distant, afin que l'autre puisse voir et intégrer rapidement les modifications.
- Avant chaque **push**, **Jean** et **Alice** doivent toujours effectuer un **pull** pour récupérer les dernières modifications du dépôt distant, réduisant ainsi le risque de conflits.