

Git dans Visual Studio Code

Contents

1. Informations - Démarrage	2
Installation	2
Commandes vscode	2
Autres outils	2
Documentations	2
2. Clone d'un repository distant	2
Créer un nouveau projet sur une forge	2
Cloner le projet dans Visual Studio Code	2
3. Préparation de l'environnement de développement pour le projet	3
4. Création et versionnement des premiers fichiers	3
5. Ajout d'un fichier .gitignore	4
6. Envoi des modifications vers le dépôt distant	5
7. Visualiser l'historique avec le plugin GitHistory	5
8. DANGER : commit direct sans ajout dans le "staging area"/"index"	5
9. Initialisation du repository	7
10. Ajout du repository distant	7
Créer un nouveau projet sur une forge	7
Configurer ce dépôt distant comme remote pour le projet	8
11. Versionnement d'un fichier	8
12. Envoi des modifications vers le dépôt distant	8
13. On clone le projet en ligne de commande pour une utilisatrice Alice fictive	9
14. On récupère les modifications depuis le dépôt distant	10
15. Créer une branche	11
16. Publier la branche	13
17. Voir les commandes git passées par vscode	13
18. Branches qui divergent	14
19. Fusion de branches avec Git History	15
Faire le merge avec git history dans vscode	17
Apparition d'un conflit de fusion	19

20. Fusion de branches sans les plugins	23
21. Examiner les modifications avec le plugin GitLens	24

1. Informations - Démarrage

Installation

Pour installer Visual Studio Code : <https://code.visualstudio.com/>

Plugins Visual Studio Code utilisés :

- Python
- Git History
- GitLens
- autoDocstring

Commandes vscode

On peut rapidement trouver une commande vscode via le raccourci **Ctrl + Shift + P**

Raccourcis clavier vscode : https://code.visualstudio.com/docs/getstarted/keybindings#_keyboard-shortcuts-reference

Autres outils

Création et gestion d'environnements virtuels Python :

- <https://docs.pipenv.org/en/latest/>
- <https://projet.liris.cnrs.fr/edp/cafes-developpeur-liris/2019-01-15-environnements-virtuels-en-python.html>

Documentations

Documentation de référence sur git :

- <https://git-scm.com/book/en/v2>
- Précédents cafés développeurs : <https://projet.liris.cnrs.fr/edp/cafes-developpeur-liris/>

2. Clone d'un repository distant

Créer un nouveau projet sur une forge

Créer d'abord un projet "git dans vscode" sur le GitLab du laboratoire : <https://gitlab.liris.cnrs.fr>

Cloner le projet dans Visual Studio Code

- ouvrir une nouvelle fenêtre vscode (**Ctrl + Shift + N** ou **Fichier > Nouvelle fenêtre**)
- rechercher la commande de clonage via **Ctrl + Shift + P** (**git clone** n'est pas dans les menus de vscode)
- saisir **clone**
- choisir **Git: Cloner**
- saisir l'url du dépôt distant : <https://fconil@gitlab.liris.cnrs.fr/fconil/git-dans-vscode.git>
= `git clone https://fconil@gitlab.liris.cnrs.fr/fconil/git-dans-vscode.git`

- sélectionner le dossier où le repository sera cloné (on ne peut pas fixer un autre nom que celui par défaut qui découle du nom du dépôt distant, `git-dans-vscode`)
- saisir le mot de passe
- vscode propose alors d'ouvrir le projet dans la fenêtre courante

J'ajoute toujours mon login **fconil@** au début de l'url récupérée pour ne pas le taper à chaque opération avec le repository distant

3. Préparation de l'environnement de développement pour le projet

J'installe un linter, un formateur de code et un logiciel de test dans un environnement virtuel Python créé avec `pipenv`.

```
$ pipenv install -d --pre flake8 black pytest
```

Je récupère le chemin vers mon environnement virtuel.

```
$ pipenv --venv
/home/fconil/.local/share/virtualenvs/git-dans-vscode-sk2FQztf
```

Pour activer l'environnement virtuel quand on travaille dans le terminal (pas nécessaire on va travailler dans vscode).

```
$ pipenv shell
```

Ajout de ma configuration de base `.vscode` à ce projet et configuration du **pythonPath** vers le Python de l'environnement virtuel créé.

```
$ cat .vscode/settings.json
{
  "python.linting.flake8Enabled": true,
  "python.linting.pylintEnabled": false,
  "python.formatting.provider": "black",
  "editor.formatOnSave": true,
  "python.pythonPath": "~/.local/share/virtualenvs/git-dans-vscode-sk2FQztf/bin/python"
}
```

On se retrouve avec les fichiers suivants (en plus du dossier `.git`) :

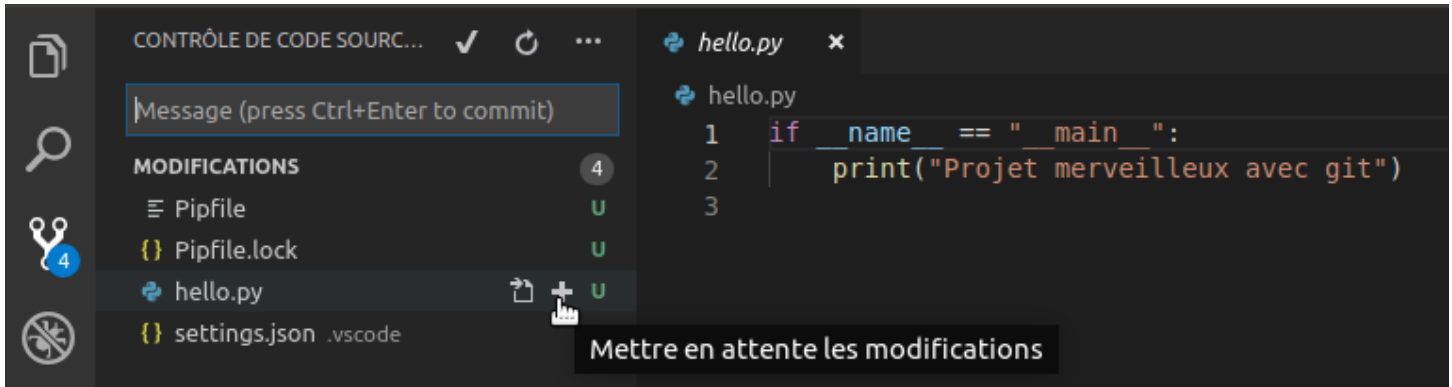
```
$ tree -a
.
├── Pipfile
├── Pipfile.lock
├── .vscode
│   └── settings.json
```

4. Création et versionnement des premiers fichiers

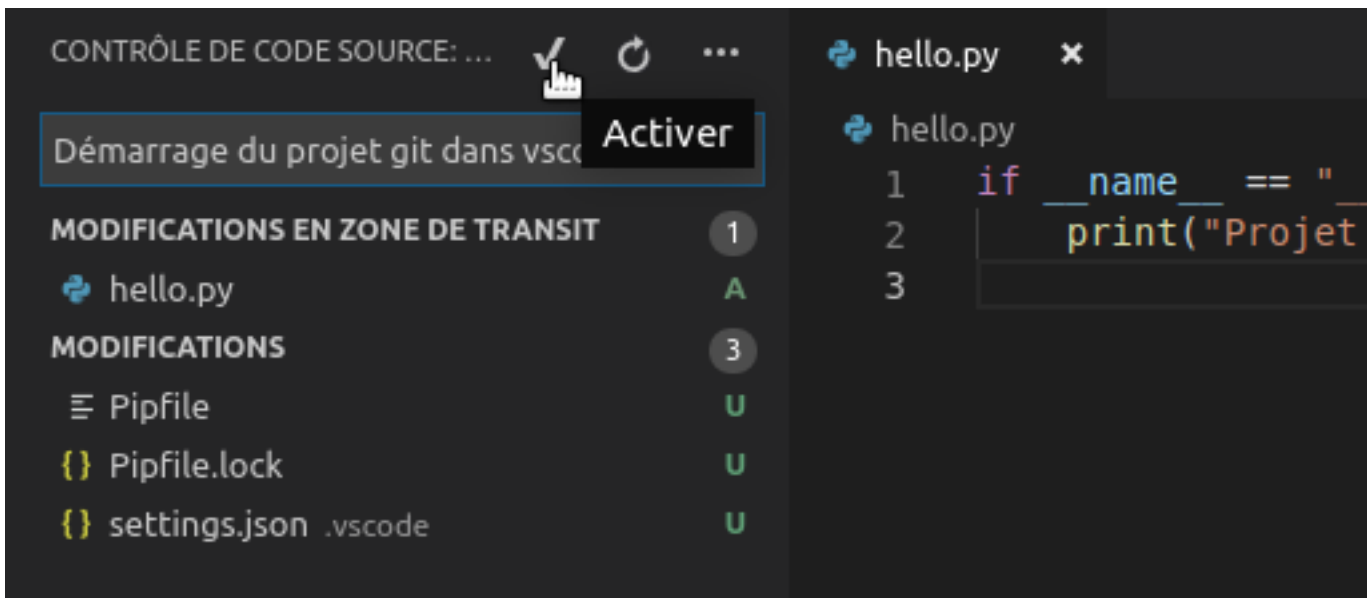
Création d'un premier fichier "hello.py".

```
if __name__ == "__main__":
    print("Projet merveilleux avec git")
```

- passer dans la fenêtre du gestionnaire de version, symbolisée par l'icône en forme de graphe 
- on voit plusieurs fichiers non suivis (U pour untracked)
- mettre le curseur sur le fichier `hello.py`
- cliquer sur l'icône + pour **Mettre en attente les modifications**, c.a.d les mettre dans l'**index** / le **staging area**
= `git add hello.py`

Figure 1: Ajouter un fichier *non suivi* (U) au gestionnaire de version

- une zone de saisie apparaît en haut qui permet de saisir le message de commit
- cliquer sur la “coche” en haut de la fenêtre du gestionnaire de version pour déclencher le commit
= `git commit -m "Démarrage du projet git dans vscode"`

Figure 2: Commit des fichiers placés dans le **staging area**/l'**index**

De la même manière, on ajoute les fichiers `Pipfile` et `Pipfile.lock` qui servent à gérer les packages Python utilisés dans le projet.

- ajouter par erreur `settings.json` et le retirer avec l'icône -, **Annuler la mise en attente**
= `git reset HEAD .vscode/settings.json`
- commit des fichiers ajoutés
= `git commit -m "Fichiers de gestion des packages Python utilisés"`

5. Ajout d'un fichier `.gitignore`

Ajout d'un fichier `.gitignore` pour ne pas être gêné par les fichiers / dossiers que l'on ne veut pas déposer dans le gestionnaire de version, comme la configuration de l'éditeur de code.

Mettre les lignes suivantes :

```
.vscode
*.pyc
*~
__pycache__
```

Ajouter le fichier et commiter “Liste des éléments que l’on ne veut pas versionner”.

6. Envoi des modifications vers le dépôt distant

- passer dans la fenêtre du gestionnaire de version, symbolisée par l’icône en forme de graphe
- cliquer sur l’icône  en haut de la fenêtre et sélectionner **push**
- saisir le mot de passe

C’est assez peu visuel contrairement à la ligne de commande.

7. Visualiser l’historique avec le plugin GitHistory

- cliquer sur l’icône en forme de losange , à droite des fichiers édités pour afficher l’historique avec le plugin “Git History”

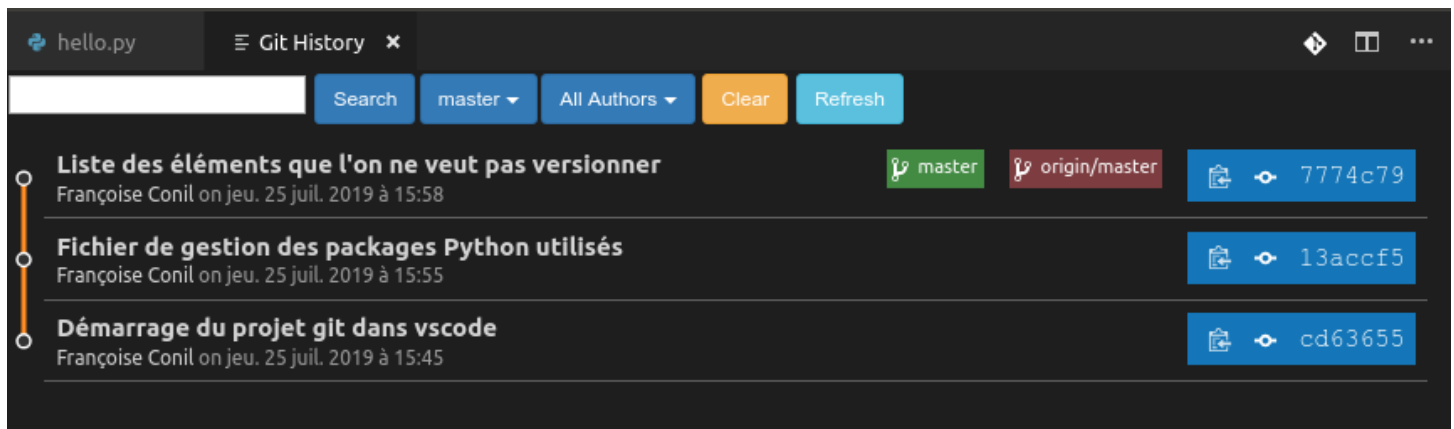


Figure 3: Historique des branches de l’espace de travail

- On peut voir toutes les branches ou une branche particulière
- On peut filtrer sur un contributeur en particulier
- On peut avoir quelques informations en cliquant sur un commit : liste des fichiers
- On peut faire des actions plus avancées en cliquant sur le **noeud**  à côté du hash du commit (branch, cherry pick, checkout, revert, merge, ...)

On aimerait voir le détail d’un commit, un diff mais ce n’est pas immédiat.

8. DANGER : commit direct sans ajout dans le “staging area”/“index”

Modifier le fichier “hello.py”

```
print("Licence : CC BY 3.0")
```

Essayer de commiter sans avoir ajouté le fichier au **staging area/index** (*mise en attente des modifications*), vscode propose de **tout ajouter** et de **le faire de façon permanente**.

Il ne faut surtout pas accepter d'ajouter systématiquement tous les fichiers au commit.

- passer dans la fenêtre du gestionnaire de version, symbolisée par l'icône en forme de graphe
- saisir un message de commit dans la zone prévue à cet effet
- cliquer sur la “coche” en haut de la fenêtre du gestionnaire de version

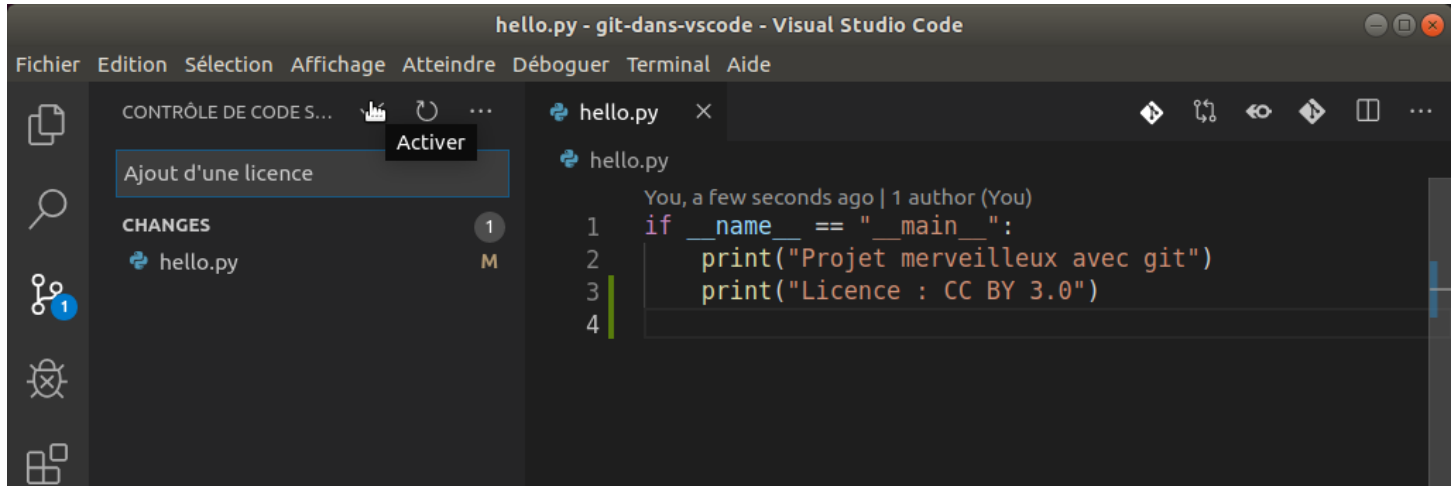


Figure 4: Déclenchement d'un commit sans avoir placé les fichiers **staging area**/l'**index**

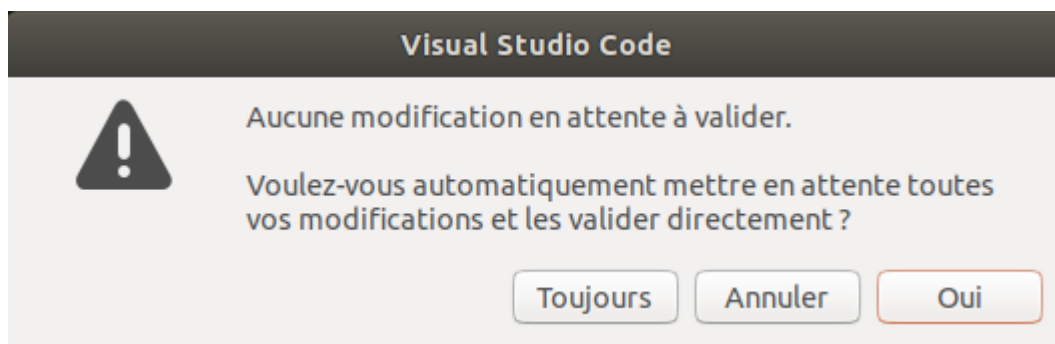


Figure 5: Ne pas accepter de faire **Toujours** cette action

Dans le cas où les fichiers ne sont pas suivis, vous pouvez vous retrouver avec des fichiers non désirés dans votre repository d'autant plus si vous n'avez pas défini de fichier `.gitignore`.

Résultat tous les fichiers sont mis dans le gestionnaire de version !

Vous pouvez annuler les modifications faites au fichier :

- passer dans la fenêtre du gestionnaire de version
- mettre le curseur sur le fichier `hello.py`
- cliquer l'icône “Ignorer les modifications”.
- confirmer

= `git checkout -- hello.py`

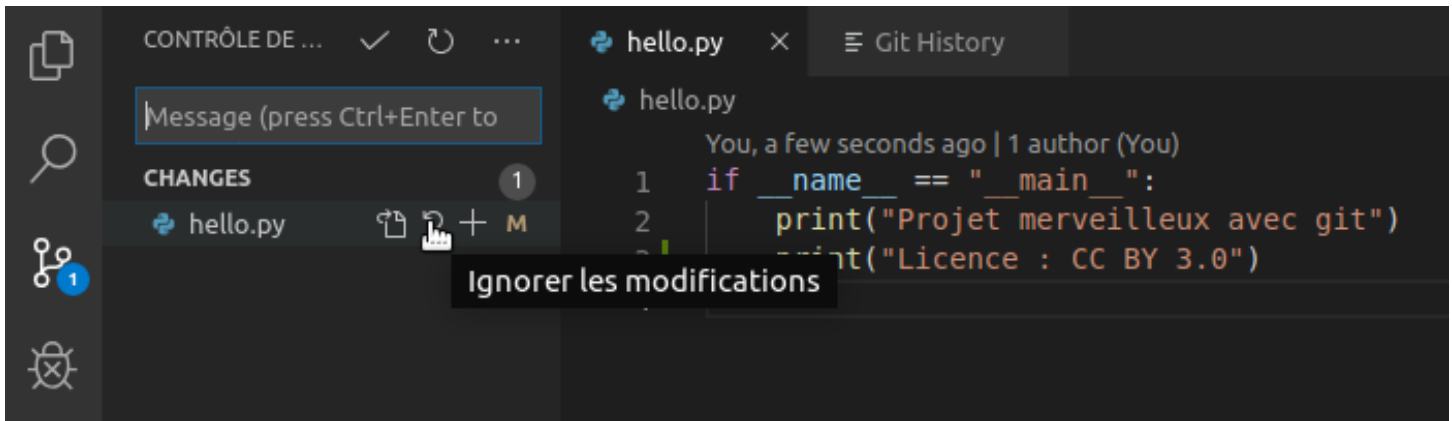


Figure 6:

9. Initialisation du repository

Créer un nouveau dossier projet dans Test :

```
$ mkdir git-dans-vscode-2
```

Dans une nouvelle fenêtre Visual Studio Code (Shit + Ctrl + N), ouvrir le dossier créé (Ctrl K, Ctrl O).

Passer dans la fenêtre du gestionnaire de version, symbolisée par l'icône en forme de graphe, vscode vous indique **Aucun fournisseur de contrôle de code source inscrit.**

Cliquer sur l'icône "+" dans cette fenêtre pour **Initialiser le dépôt**

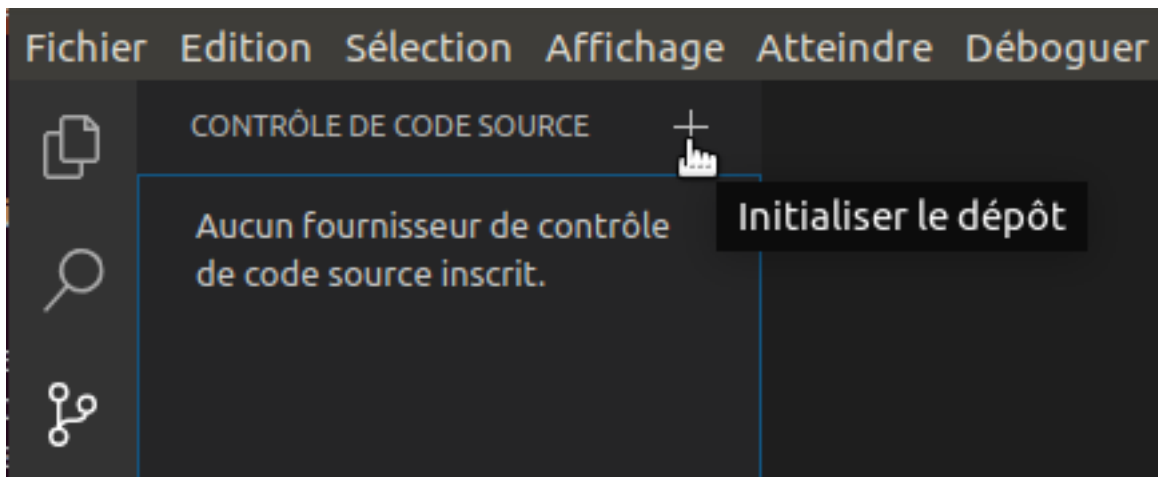


Figure 7:

Sélectionner le dossier créé `git-dans-vscode-2`.

10. Ajout du repository distant

Créer un nouveau projet sur une forge

Créer d'abord un projet "git dans vscode 2" sur le GitLab du laboratoire : <https://gitlab.liris.cnrs.fr>

Configurer ce dépôt distant comme remote pour le projet

- rechercher la commande d'ajout de dépôt distant via **Ctrl + Shift + P** (la commande n'est pas dans les menus de vscode)
- saisir **remote**
- choisir **Git: Ajouter un dépôt distant**
- saisir le Nom du dépôt distant : **origin** (c'est le nom utilisé par défaut par de nombreuses commandes git, il est dommage que vscode ne le suggère pas quand il n'y a aucun remote existant dans un projet)
- saisir l'url du dépôt distant : `https://fconil@gitlab.liris.cnrs.fr/fconil/git-dans-vscode-2.git`
= `git remote add origin https://fconil@gitlab.liris.cnrs.fr/fconil/git-dans-vscode-2.git`

Comme précédemment, j'ajoute mon login **fconil@** au début de l'url récupérée pour ne pas le taper à chaque opération avec le repository distant.

11. Versionnement d'un fichier

Création d'un fichier "hello.py".

```
if __name__ == "__main__":  
    print("Projet merveilleux avec git")
```

Ajouter le fichier au **staging area/index** et commiter "Démarrage du projet".

12. Envoi des modifications vers le dépôt distant

Dans le cas d'un dépôt initialisé localement et envoyé vers un dépôt distant, il faut créer la branche **master** sur le dépôt distant en utilisant la commande **Publier la branche** et non la commande **git push**.

- aller dans la fenêtre du gestionnaire de version
- cliquer sur l'icône ... en haut de la fenêtre et sélectionner **Publier la branche**
= `git push -u origin master`
- saisir le mot de passe

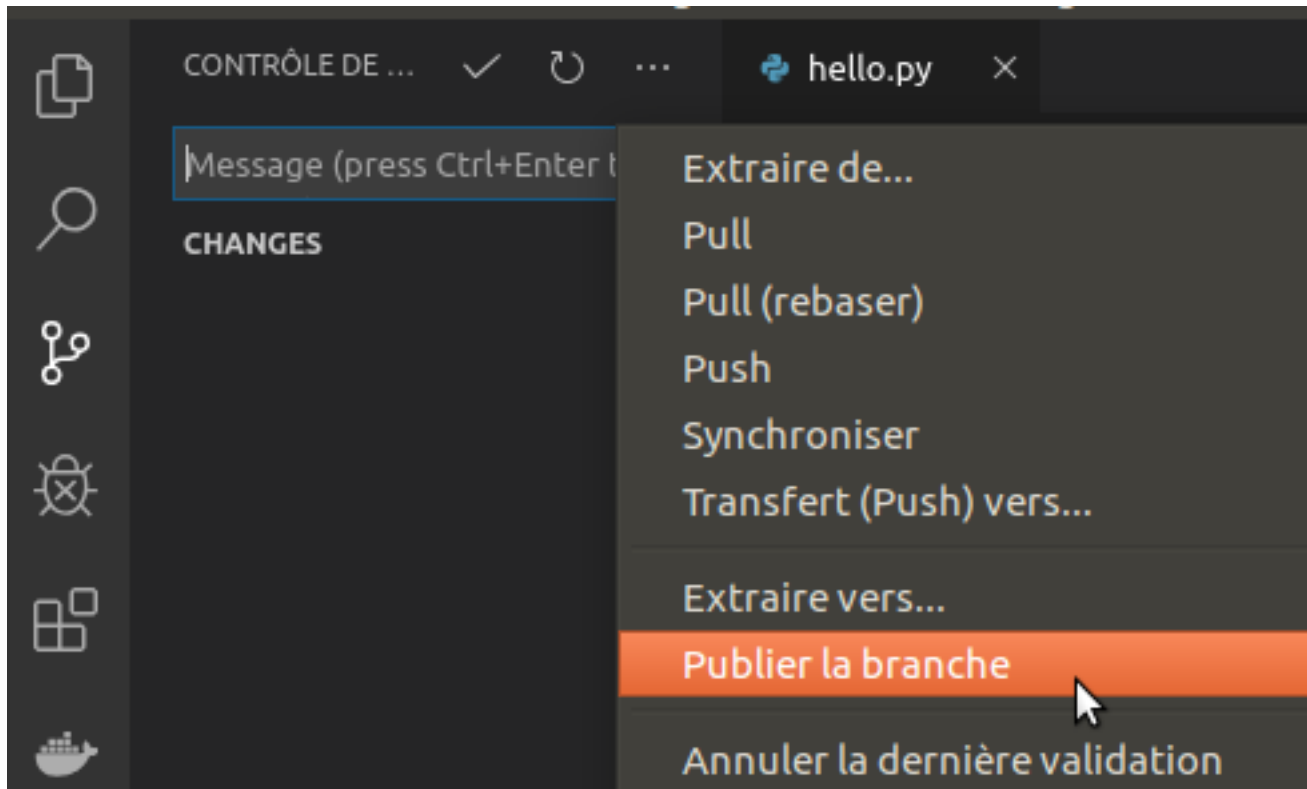


Figure 8:

Dans ce cas, la liaison avec la branche du dépôt distant est correctement effectuée et vous pourrez faire des `git push` normalement par la suite.

Faire un **Push** de la branche n'aurait pas créé d'erreur sur le moment mais vous auriez été incapable de récupérer du code du dépôt distant par la suite sans corriger la configuration, via :

```
git branch --set-upstream-to=origin/<branch> master
```

Visual Studio indique cette commande lorsque le `git pull` échoue.

13. On clone le projet en ligne de commande pour une utilisatrice Alice fictive

Revenir sur le projet cloné initialement `git-dans-vscode` et récupérer la configuration locale du projet par la commande :

```
$ git config --local -l
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
remote.origin.url=https://fconil@gitlab.liris.cnrs.fr/fconil/git-dans-vscode.git
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
branch.master.remote=origin
branch.master.merge=refs/heads/master
```

Cloner le projet dans un nouveau dossier pour une utilisatrice Alice fictive.

```
$ cd Test
$ git clone https://fconil@gitlab.liris.cnrs.fr/fconil/git-dans-vscode.git git-dans-vscode-alice
```

On change quelques paramètres pour l'affichage

```
$ git config user.name "Alice"
$ git config user.email "alice@liris.cnrs.fr"
```

Alice fait une modification dans `hello.py` :

```
print("Durée du projet : 18 mois")
```

qu'elle envoie sur le dépôt distant.

```
$ git commit -am "Ajout de la durée du projet"
[master 4b591f8] Ajout de la durée du projet
 1 file changed, 1 insertion(+)
$ git push
Password for 'https://fconil@gitlab.liris.cnrs.fr':
Décompte des objets: 3, fait.
Delta compression using up to 8 threads.
Compression des objets: 100% (3/3), fait.
Écriture des objets: 100% (3/3), 353 bytes | 353.00 KiB/s, fait.
Total 3 (delta 1), reused 0 (delta 0)
To https://gitlab.liris.cnrs.fr/fconil/git-dans-vscode.git
 7774c79..4b591f8  master -> master
```

14. On récupère les modifications depuis le dépôt distant

- dans la fenêtre du gestionnaire de version, symbolisée par l'icône en forme de graphe
- cliquer sur l'icône ... en haut de la fenêtre et sélectionner **pull**
- saisir le mot de passe

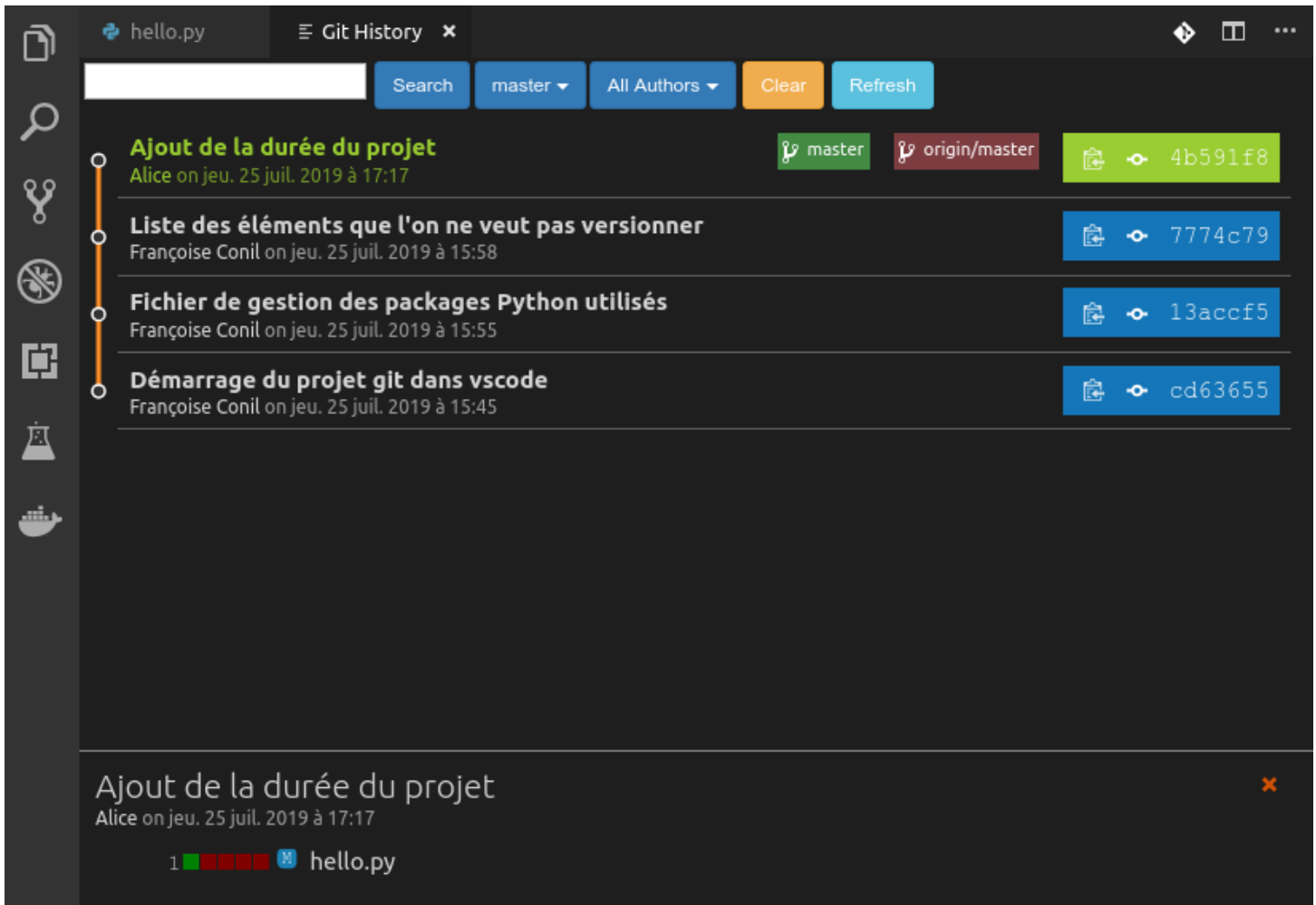


Figure 9: Visualisation de l'historique avec les modifications d'Alice

Si on clique sur le commit d'Alice, puis sur le fichier `hello.py`, on a une liste d'options proposées dont **Compare against previous version** qui affiche un diff du fichier entre la version sélectionnée et la version précédente.

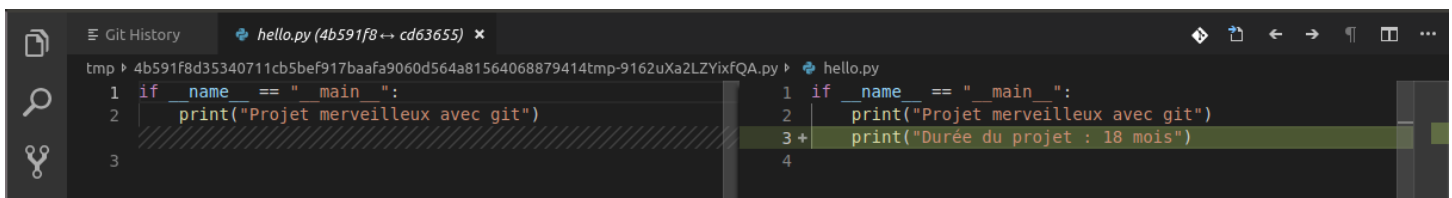


Figure 10: Visualisation des modifications faites par Alice, diff graphique sur le fichier

15. Créer une branche

En bas, à gauche de la fenêtre de d'application, on a une icône de graphe avec la branche courante **master**,  master.

Si on clique sur cette zone, on a une liste de possibilités (pur vscode) :

- Créer une branche ...

- b. Créer une branche à partir de ...
- c. master
- d. origin/master

On choisit **Créer une branche ...** et on saisit le nom de la branche à créer : **liste-des-auteurs**.

On voit que la branche courante a changé en bas à gauche, .

On crée un nouveau fichier `utiles.py` avec une fonction `liste_auteurs()`.

```
def liste_auteurs():
    return ["Alice", "Bob"]
```

et on va appeler cette fonction dans `hello.py`.

```
from utiles import liste_auteurs

auteurs = liste_auteurs()
if auteurs is not None and len(auteurs) > 0:
    print("Liste des auteurs : ", ", ".join(auteurs))
```

Aller dans la fenêtre du gestionnaire de version et cliquer sur l'icône + pour **Mettre en attente les modifications** des 2 fichiers `utiles.py` et `hello.py`

```
= git add hello.py utiles.py
```

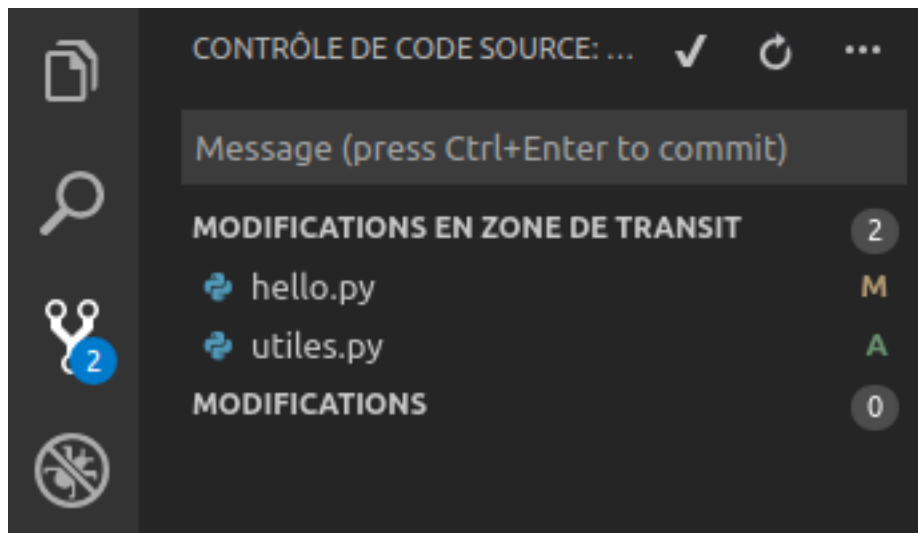
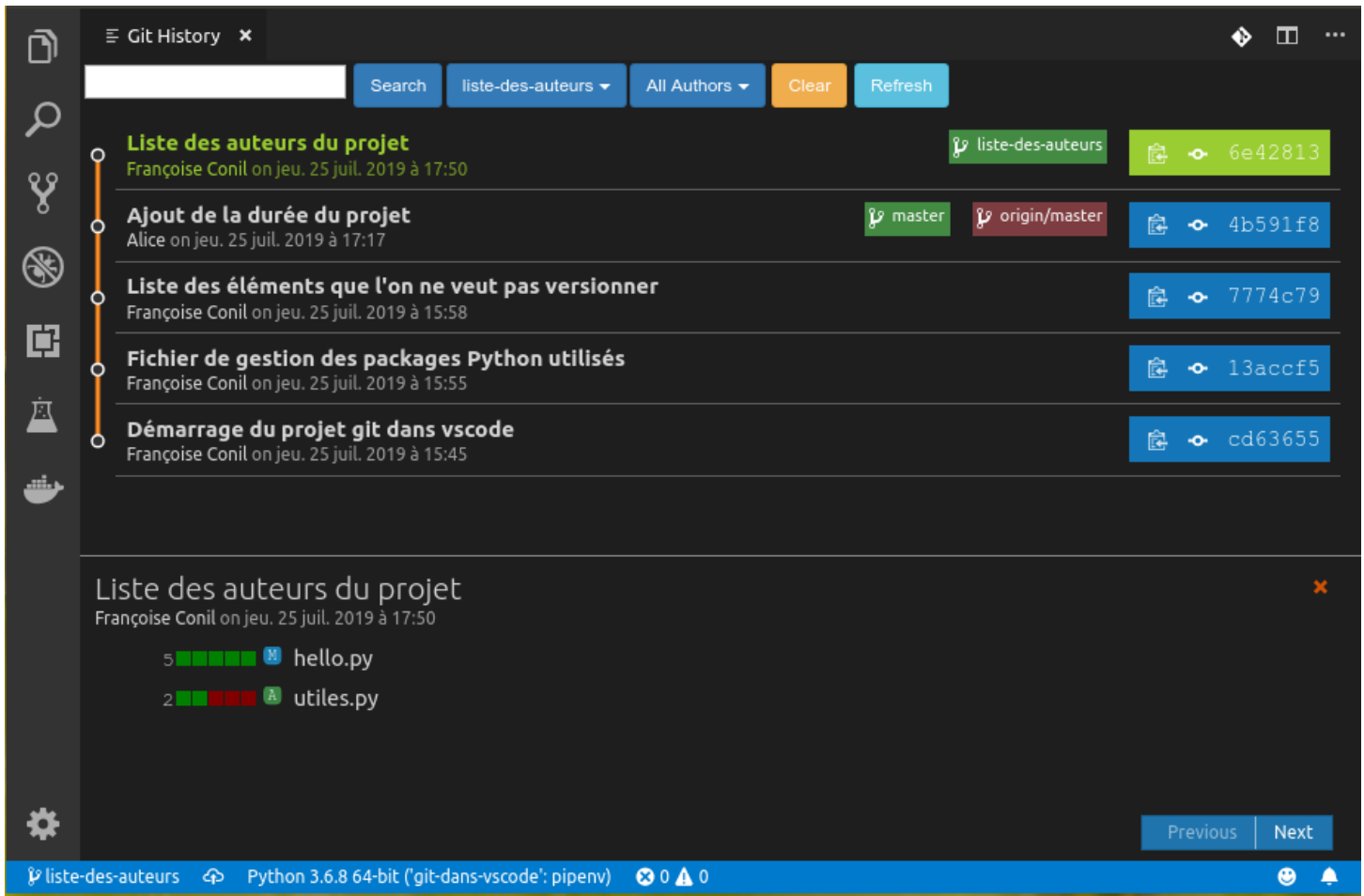


Figure 11: État des fichiers de l'espace de travail : ajouté A et modifié M

On commit les modifications en attente.

```
= git commit -m "Liste des auteurs du projet"
```

On peut visualiser l'état courant avec `git history` en cliquant sur l'icône en forme de losange en haut, à droite.

Figure 12: Visualisation de l'historique de la branche **liste-des-auteurs**

16. Publier la branche

On veut envoyer la nouvelle branche vers le dépôt distant.

- dans la fenêtre du gestionnaire de version
- cliquer sur l'icône ... en haut de la fenêtre et sélectionner **Publier la branche**
- saisir le mot de passe

```
= git push -u origin liste-des-auteurs
```

La commande envoie les modifications vers le dépôt distant et établit la liaison entre la branche locale et la branche sur le dépôt distant.

```
$ git branch -vv
* liste-des-auteurs 6e42813 [origin/liste-des-auteurs] Liste des auteurs du projet
  master            f6be99e [origin/master] Ajout du financement
```

17. Voir les commandes git passées par vscode

vscode passe des commandes beaucoup plus spécifiques que ce que l'on fait manuellement, mais il peut être intéressant de les afficher pour vérifier ce qu'il a fait.

- dans la fenêtre du gestionnaire de version
- cliquer sur l'icône ... en haut de la fenêtre et sélectionner **Afficher la sortie git**

On voit qu'il a bien passé le paramètre `-u` lors du `git push`. Accessoirement on voit la réponse du server GitLab.

```
> git push -u origin liste-des-auteurs
remote:
remote: To create a merge request for liste-des-auteurs, visit:
remote:   https://gitlab.liris.cnrs.fr/fconil/git-dans-vscode/merge_requests/new?merge_request%5Bsource_branch%5D=liste-des-auteurs
remote:
To https://gitlab.liris.cnrs.fr/fconil/git-dans-vscode.git
* [new branch]      liste-des-auteurs -> liste-des-auteurs
```

18. Branches qui divergent

Alice modifie le fichier `hello.py` dans la branche **master** et envoie ses modifications au dépôt distant.

```
print("Financement : ANR DVCS")
```

Alice commit ses modifications et les envoie au dépôt distant.

```
= git commit -m "Ajout du financement" = git push
```

Dans vscode, revenir sur la branche **master**, via la zone en bas à gauche de la fenêtre et récupérer des modifications d'Alice.

- dans la fenêtre du gestionnaire de version
- cliquer sur l'icône ... en haut de la fenêtre et sélectionner **pull**
- saisir le mot de passe

On peut visualiser l'état courant avec `git history` en spécifiant que l'on veut voir toutes les branches.

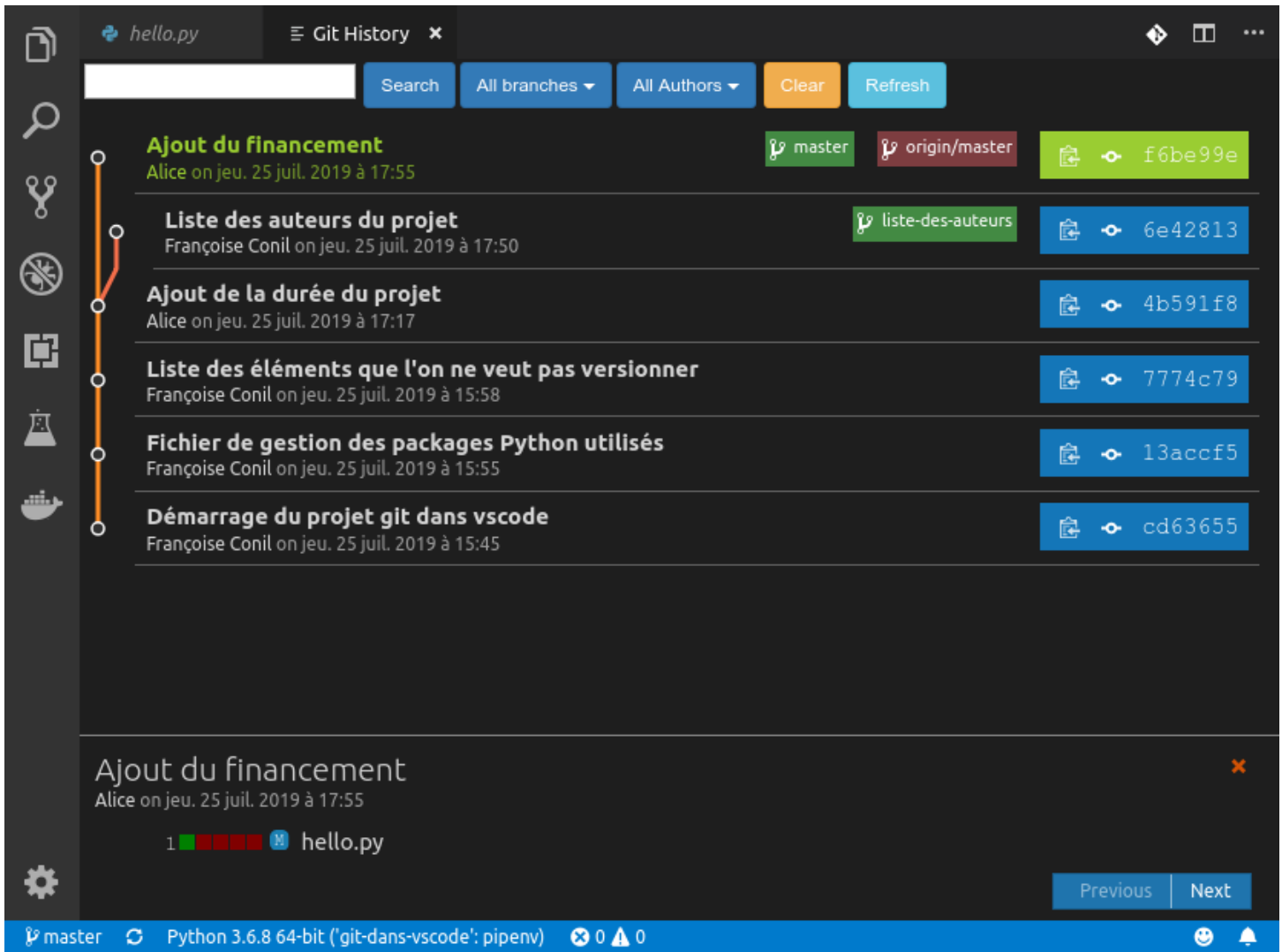


Figure 13: Visualisation de l'évolution parallèles des 2 branches

19. Fusion de branches avec Git History

Ça devient compliqué malheureusement en utilisant l'interface proposée par ce plugin.

Alice récupère le code de la branche **liste-des-auteurs** :

```
$ git fetch
```

```
$ git checkout liste-des-auteurs
```

La branche '**liste-des-auteurs**' est paramétrée pour suivre la branche distante '**liste-des-auteurs**' depuis '**origin**'.
Basculement sur la nouvelle branche '**liste-des-auteurs**'

```
$ git branch -vv
```

```
* liste-des-auteurs 6e42813 [origin/liste-des-auteurs] Liste des auteurs du projet
  master           f6be99e [origin/master] Ajout du financement
```

elle ajoute la liste des tutelles :

```
# utiles.py
def liste_tutelles():
```

```
return ["CNRS", "Université Lyon 1", "INSA", "Université Lyon 2", "École Centrale"]
```

```
# hello.py
```

```
tutelles = liste_tutelles()
if tutelles is not None and len(tutelles) > 0:
    print("Liste des tutelles : ", ", ".join(tutelles))
```

puis elle envoie ses modifications vers le dépôt distant.

```
$ git commit -am "Ajout des tutelles"
[liste-des-auteurs 6185527] Ajout des tutelles
2 files changed, 7 insertions(+)
```

```
$ git push
```

```
Password for 'https://fconil@gitlab.liris.cnrs.fr':
```

```
Décompte des objets: 4, fait.
```

```
Delta compression using up to 8 threads.
```

```
Compression des objets: 100% (4/4), fait.
```

```
Écriture des objets: 100% (4/4), 516 bytes | 516.00 KiB/s, fait.
```

```
Total 4 (delta 2), reused 0 (delta 0)
```

```
remote:
```

```
remote: To create a merge request for liste-des-auteurs, visit:
```

```
remote: https://gitlab.liris.cnrs.fr/fconil/git-dans-vscode/merge_requests/new?merge_request%5Bsource_branch%5D=liste-des-auteurs
```

```
remote:
```

```
To https://gitlab.liris.cnrs.fr/fconil/git-dans-vscode.git
```

```
6e42813..6185527 liste-des-auteurs -> liste-des-auteurs
```

Elle corrige un oubli dans `hello.py` et envoie sa correction vers le dépôt distant.

```
from utiles import liste_auteurs, liste_tutelles
```

```
$ git add hello.py
```

```
$ git commit -m "Oubli d'import"
```

```
[liste-des-auteurs 0695452] Oubli d'import
```

```
1 file changed, 1 insertion(+), 1 deletion(-)
```

```
$ git push
```

```
Password for 'https://fconil@gitlab.liris.cnrs.fr':
```

```
Décompte des objets: 3, fait.
```

```
Delta compression using up to 8 threads.
```

```
Compression des objets: 100% (3/3), fait.
```

```
Écriture des objets: 100% (3/3), 304 bytes | 304.00 KiB/s, fait.
```

```
Total 3 (delta 2), reused 0 (delta 0)
```

```
remote:
```

```
remote: To create a merge request for liste-des-auteurs, visit:
```

```
remote: https://gitlab.liris.cnrs.fr/fconil/git-dans-vscode/merge_requests/new?merge_request%5Bsource_branch%5D=liste-des-auteurs
```

```
remote:
```

```
To https://gitlab.liris.cnrs.fr/fconil/git-dans-vscode.git
```

```
6185527..0695452 liste-des-auteurs -> liste-des-auteurs
```

Dans vscode :

- passer sur la branche **liste-des-auteurs**, via la zone en bas à gauche de la fenêtre =>
- dans la fenêtre du gestionnaire de version, cliquer sur l'icône ... en haut de la fenêtre et sélectionner **pull**
- saisir le mot de passe



Faire le merge avec git history dans vscode

- revenir sur la branche **master** dans vscode, via la zone en bas à gauche de la fenêtre => 
- cliquer sur l'icône en forme de losange en haut, à droite pour afficher l'historique avec le plugin "Git History"
- demander la visualisation de toutes les branches
- cliquer l'icône en forme de **noeud**  à côté du hash du dernier commit de la branche **liste-des-auteurs**
- dans le menu qui s'affiche, choisir l'option **merge this (hash) commit into current branch**

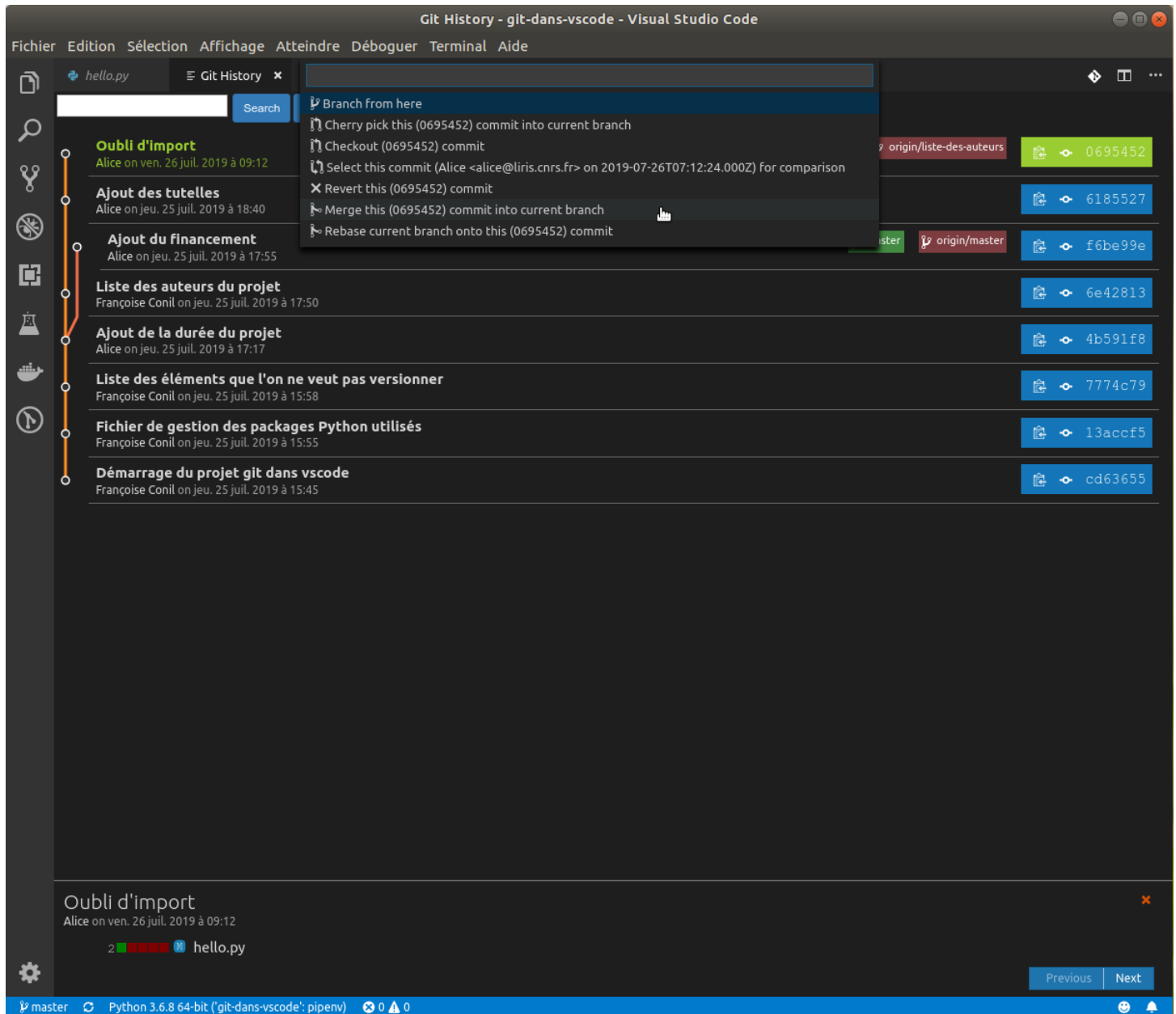
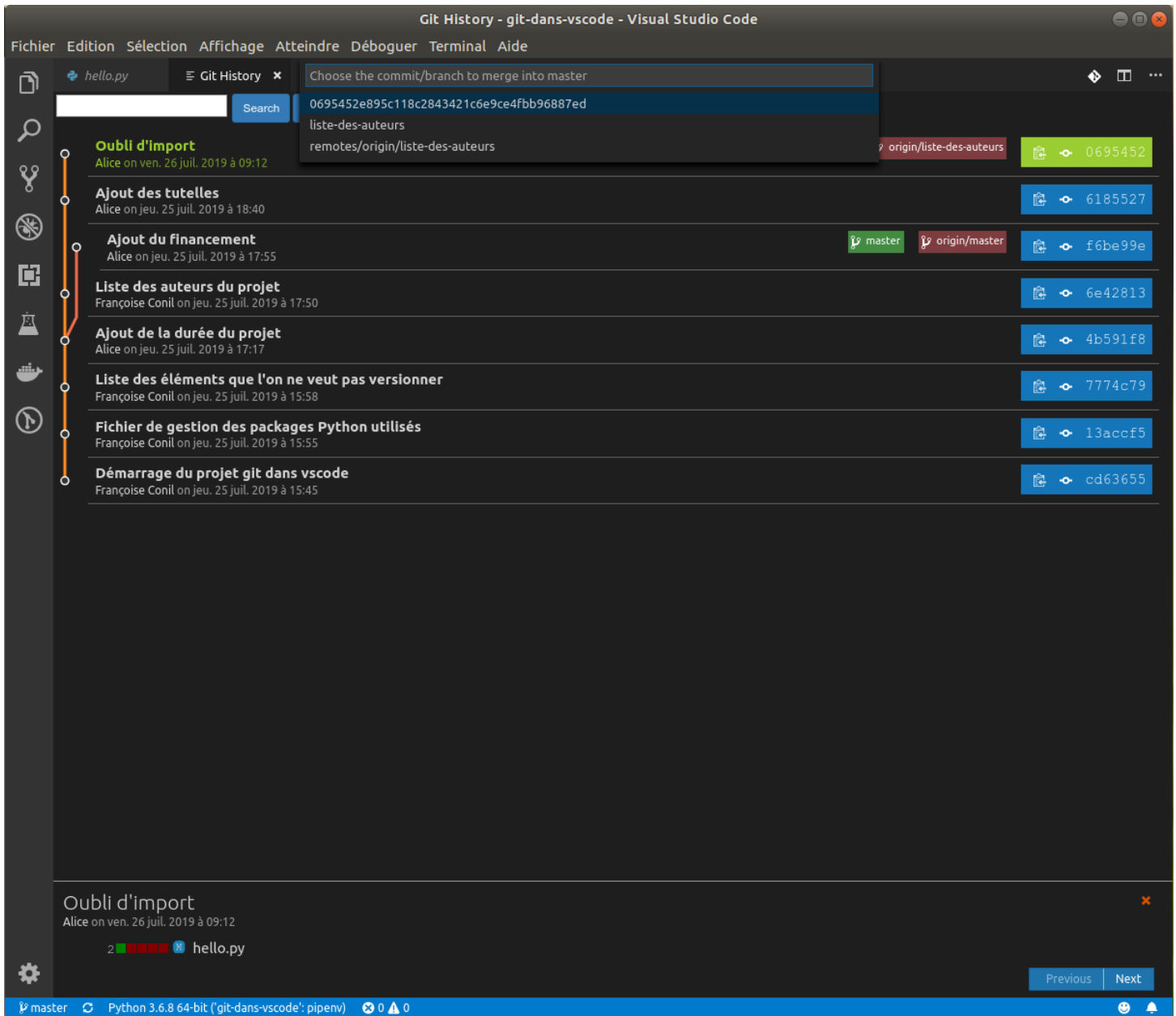


Figure 14: Les options de merge ne sont pas claires, on a l'impression que l'on va merger **un seul commit**

- le plugin propose alors plusieurs options, choisir la fusion avec le hash du pointeur de la branche **liste-des-auteurs**, cela paraît redondant et porte à confusion

Figure 15: Choisir la fusion avec la branche **liste-des-auteurs**

- confirmer l'opération de fusion

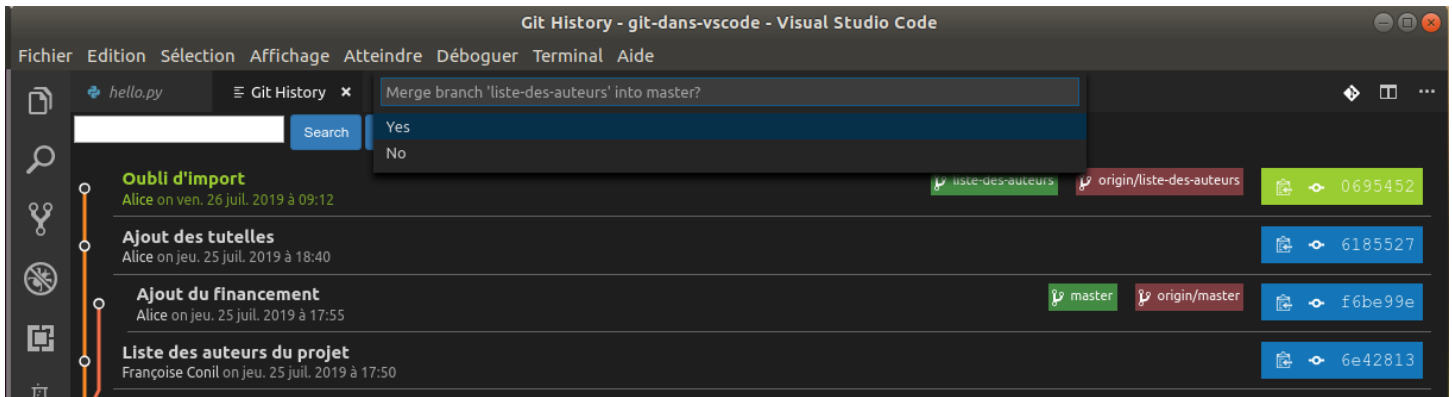


Figure 16: Confirmer la fusion

Apparition d'un conflit de fusion

On se retrouve avec un fichier fusionné et un conflit sur le fichier `hello.py`.

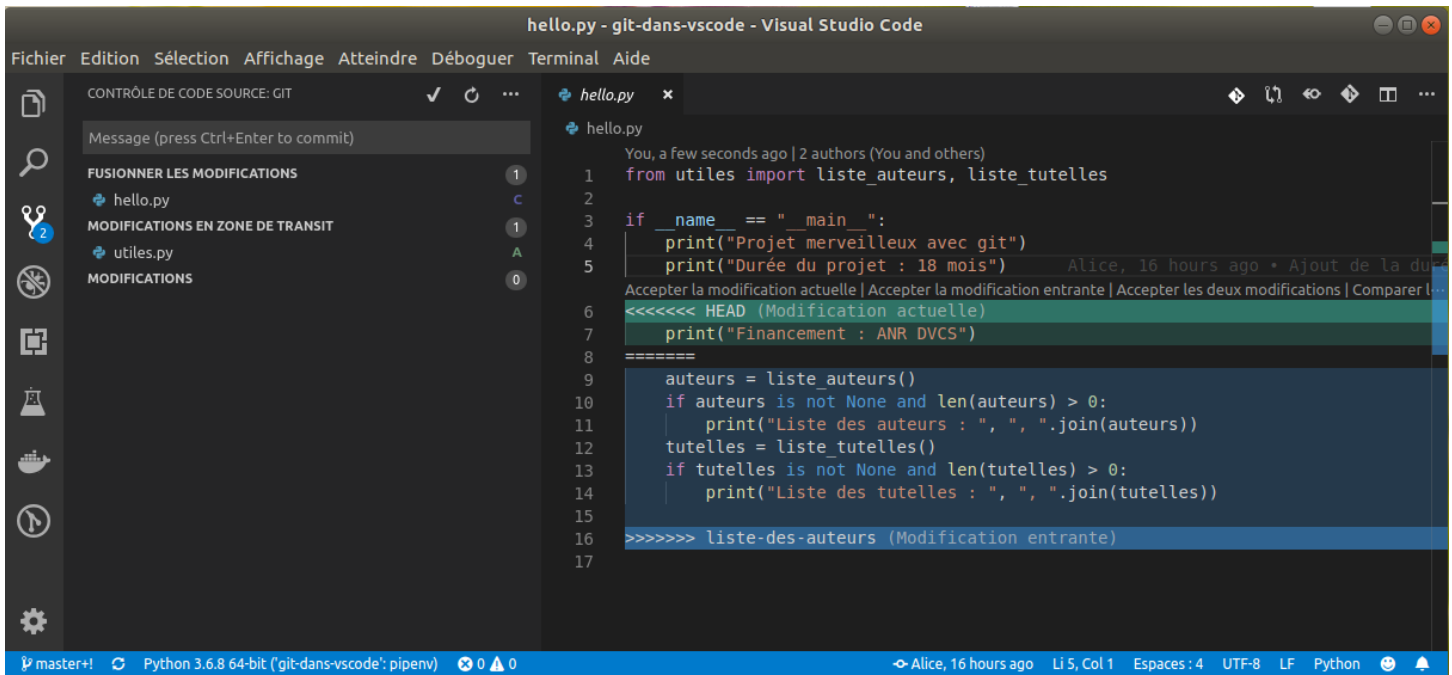
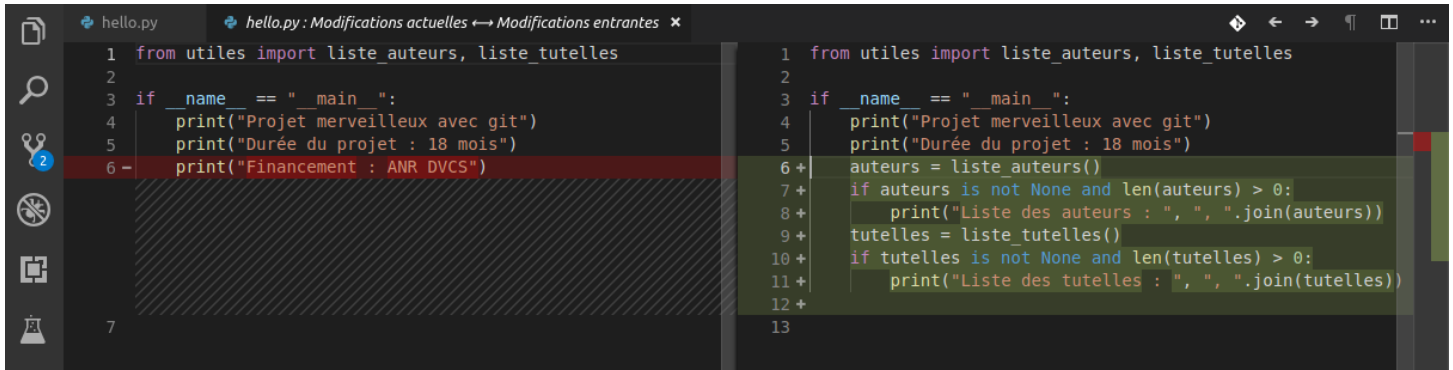


Figure 17: Affichage du conflit de merge par Git History

Lorsque l'on clique sur le fichier en conflit, 4 options sont proposées. C'est plutôt clair.

- Accepter la modification actuelle
- Accepter la modification entrante
- Accepter les deux modifications
- Comparer les modifications

Figure 18: Choix de l'option **Comparer les modifications**

Le **Ctrl + Z** fonctionne, c'est pratique.

On peut également résoudre le conflit à la main.

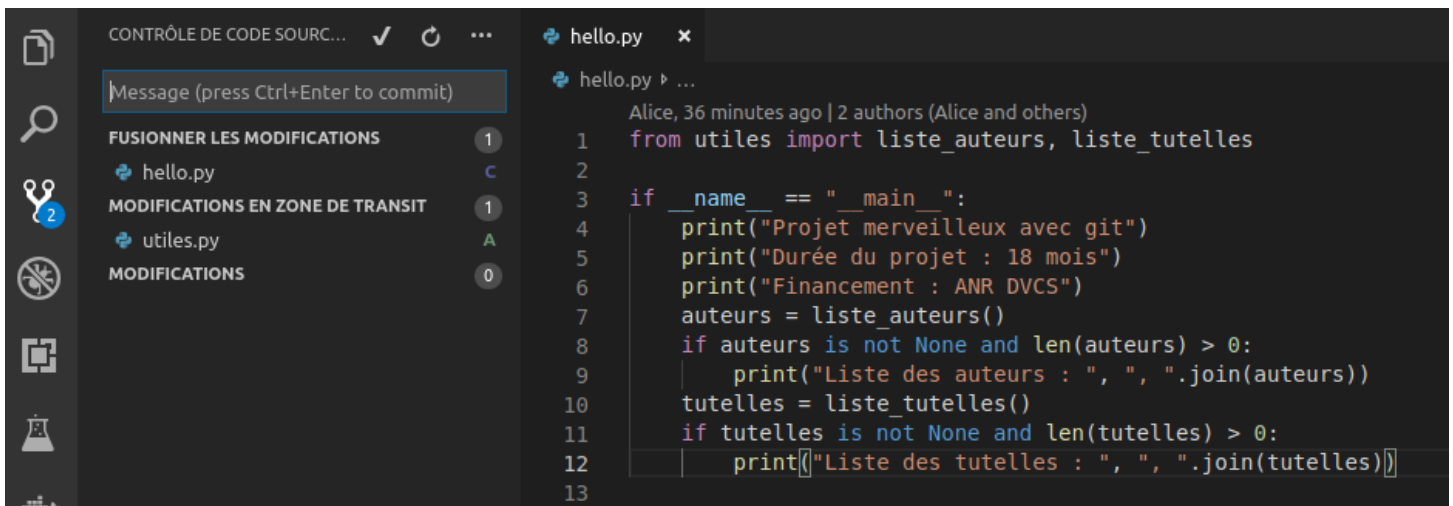
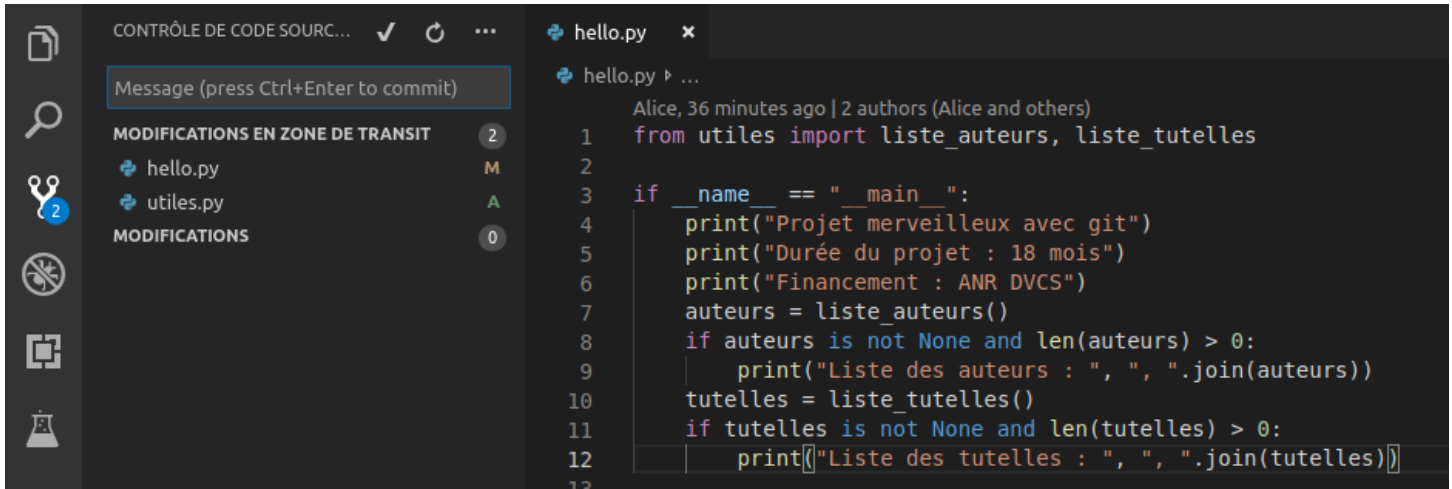


Figure 19: Résolution manuelle du conflit

Une fois le conflit résolu, on ajoute le fichier en conflit au **staging area/index**

- cliquer sur l'icône + pour **Mettre en attente les modification**
= `git add hello.py`

Figure 20: Ajout du fichier sur lequel le conflit a été résolu au **staging area/index**

Il ne reste plus qu'à saisir un message de commit et valider pour enregistrer les modifications dans le dépôt local

- saisir le message de commit dans la zone prévue à cet effet en haut de la fenêtre
- cliquer sur la "coche" en haut de la fenêtre du gestionnaire de version

= `git commit -m "Fusion de la branche liste-des-auteurs"`

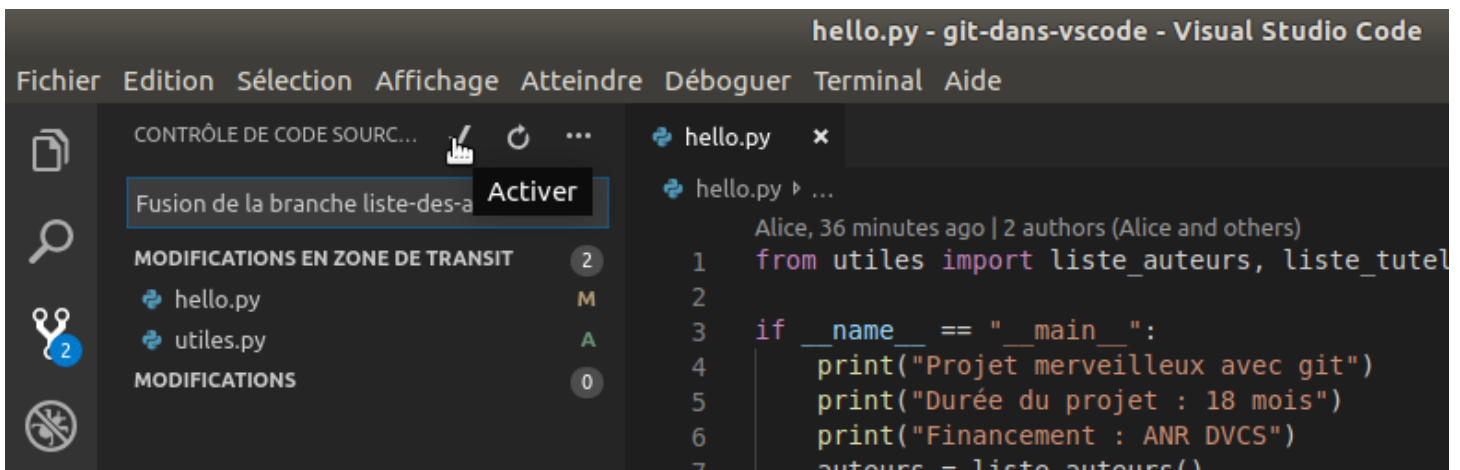


Figure 21: Commit de la fusion des branches après résolution du conflit

On peut visualiser l'arborescence résultante avec "Git History"

- cliquer sur l'icône en forme de losange en haut, à droite pour afficher l'historique avec le plugin "Git History"

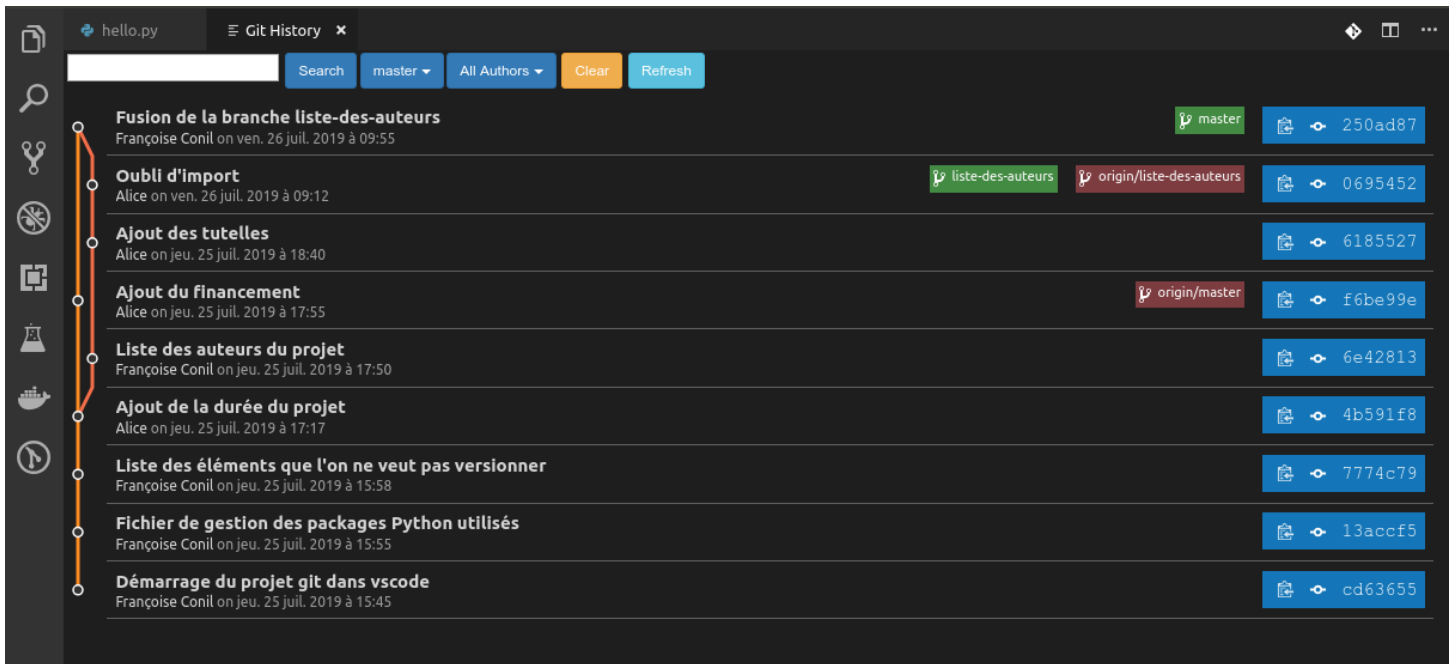
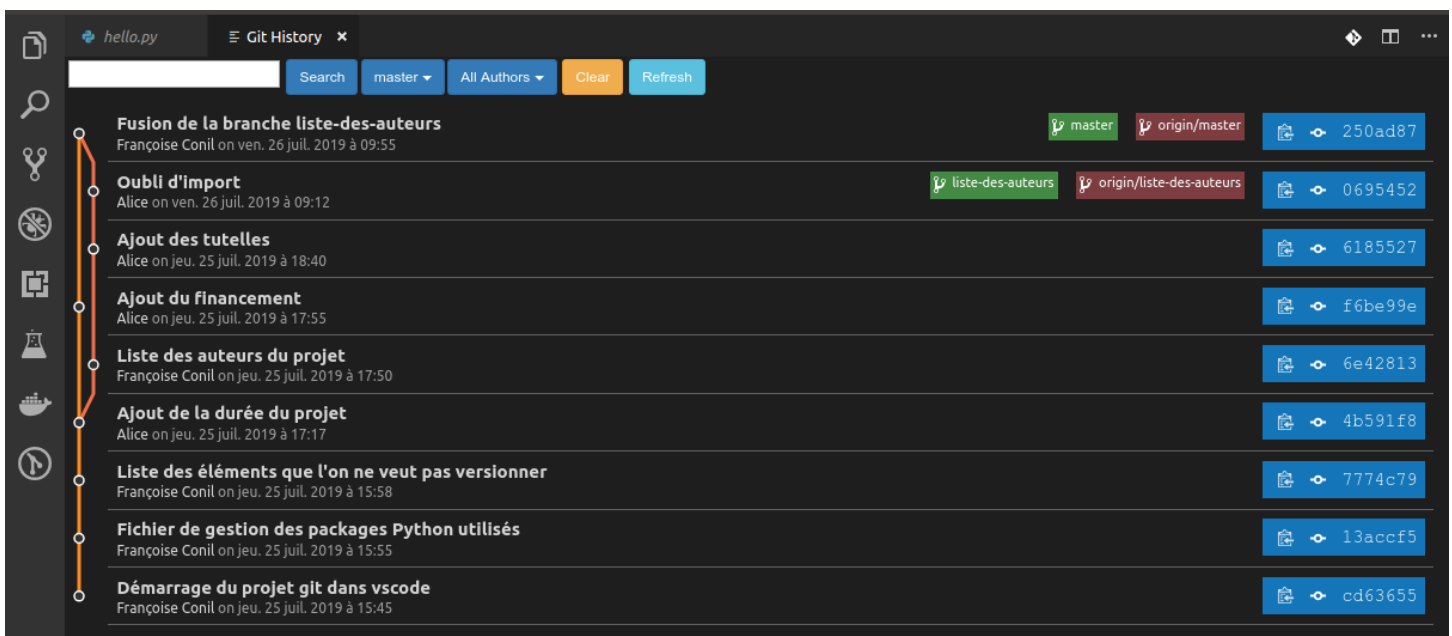


Figure 22: Visualisation de la fusion des branches avec Git History

Transférer les modifications vers le dépôt distant

- cliquer sur l'icône **...** en haut de la fenêtre et sélectionner **push**
- saisir le mot de passe

On voit alors que le pointeur **origin/master** est au même commit que le pointeur de la branche locale **master**.

Figure 23: Après le **push** les pointeurs **origin/master** et **master** sont au même niveau

20. Fusion de branches sans les plugins

Pour fusionner une branche sans les plugins, il faut passer par la recherche de commande de Visual Studio Code.

- revenir sur la branche **master** dans vscode, via la zone en bas à gauche de la fenêtre => 
 - rechercher la commande d'ajout de dépôt distant via **Ctrl + Shift + P**
 - saisir **merge**
 - choisir **Git: Merge branch**
 - sélectionner la branche à fusionner
- = `git checkout master`
- = `git merge licence`

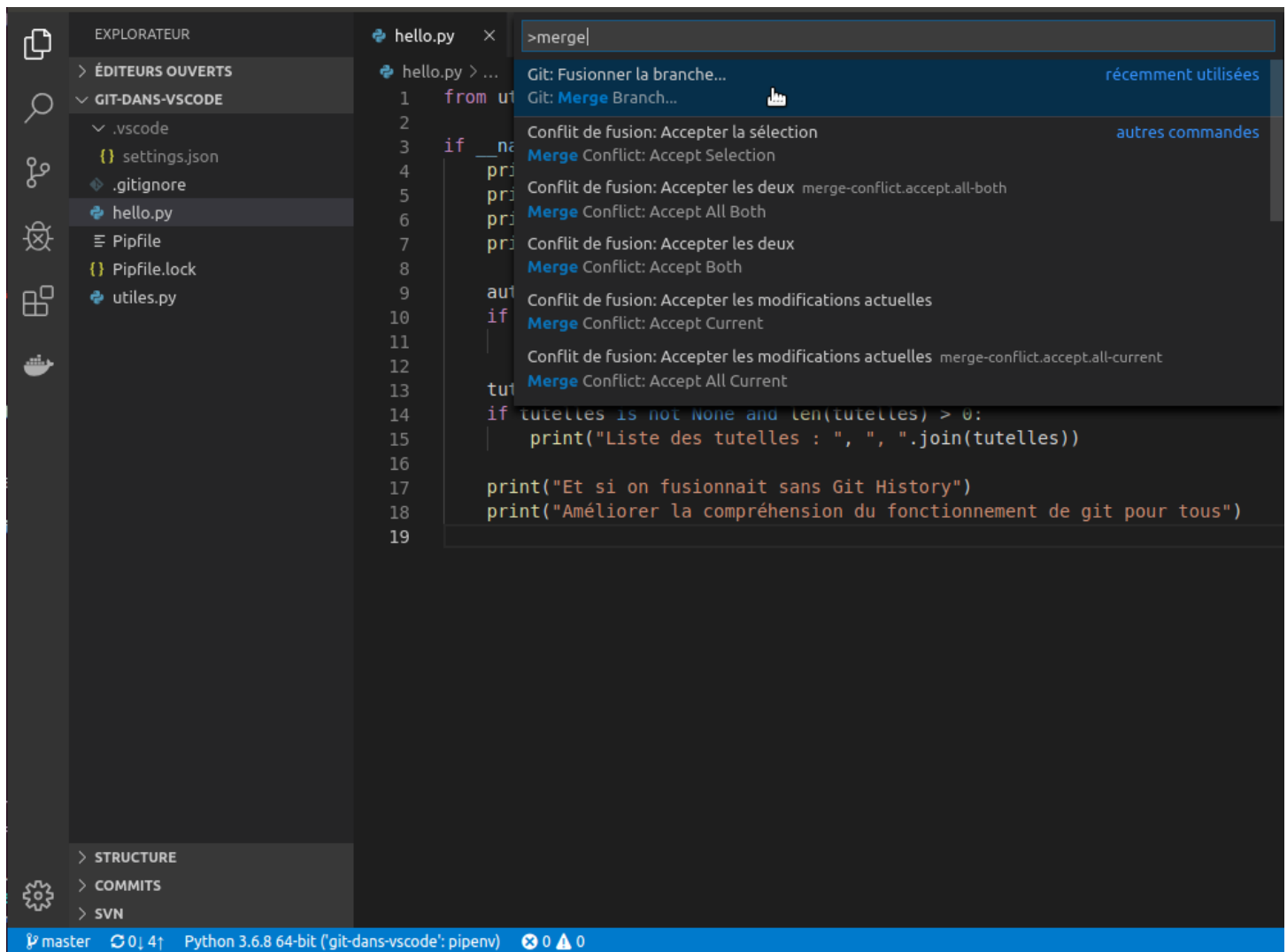


Figure 24: Exécution de la commande **merge** de Visual Studio Code pour la fusion de branches

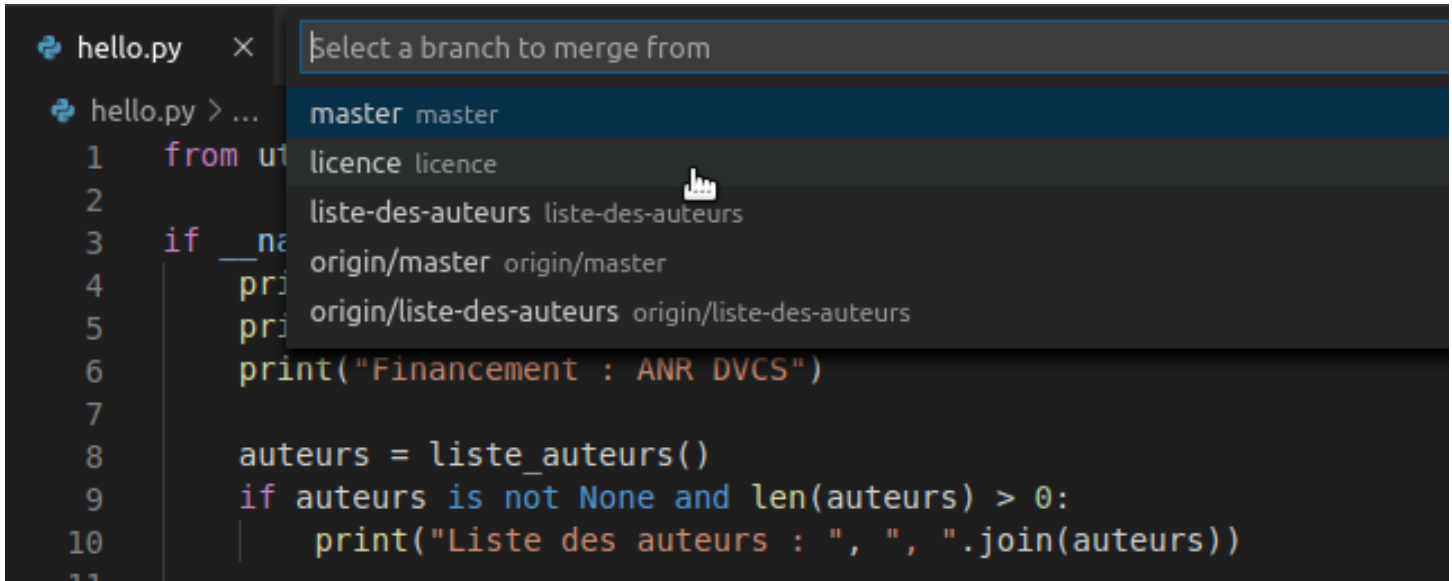


Figure 25: Sélection de la branche à fusionner

21. Examiner les modifications avec le plugin GitLens

GitLens facilite l'étude des modifications faites au code via les annotations fournies par `git blame` et la fonctionnalité CodeLens de vscode. CodeLens permet d'afficher de l'information contextuelle sur le code.

Le plugin semble assez riche et je n'ai pas eu le temps de m'en servir beaucoup, je ne le détaillerai donc pas.

<https://github.com/eamodio/vscode-gitlens/tree/master>

Quand on édite un fichier, si on se place sur une ligne, on a un texte discret à droite qui indique qui a modifié cette ligne en dernier, il y a combien de temps et le message du commit. Si on place le curseur sur ce texte, on a quelques informations complémentaires et la possibilité de déclencher d'autres actions (voir le diff de ce commit avec l'icône en forme de double chevrons)

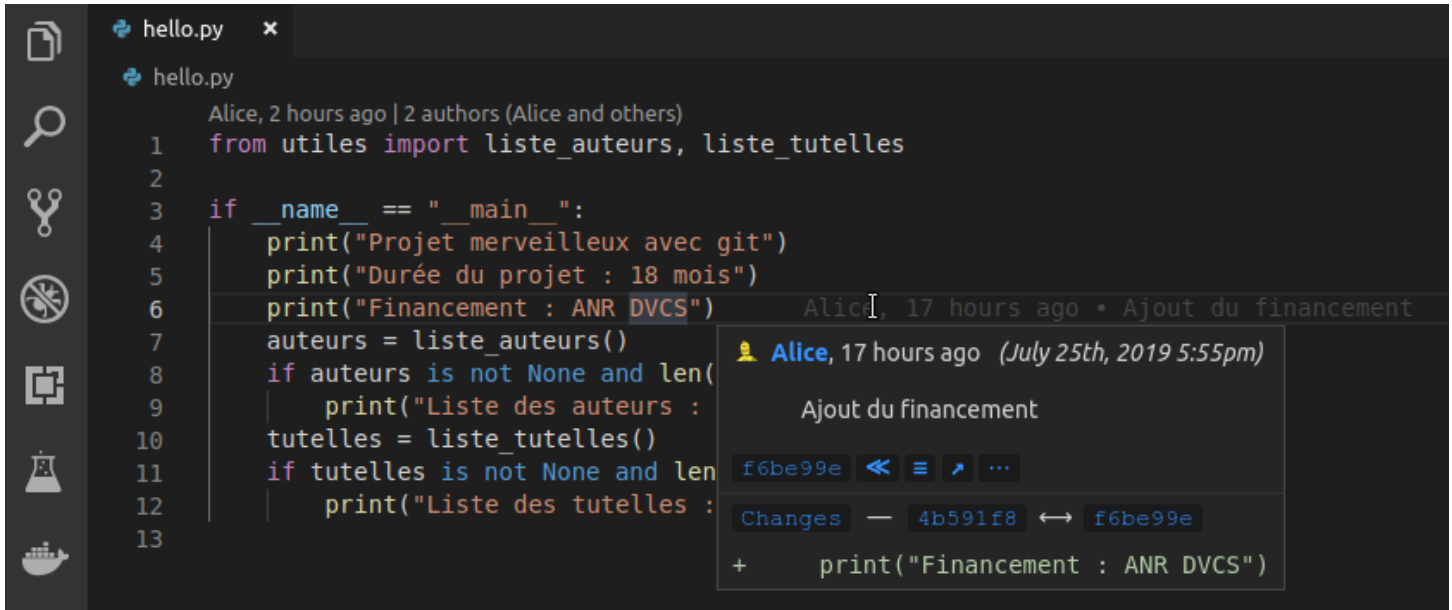
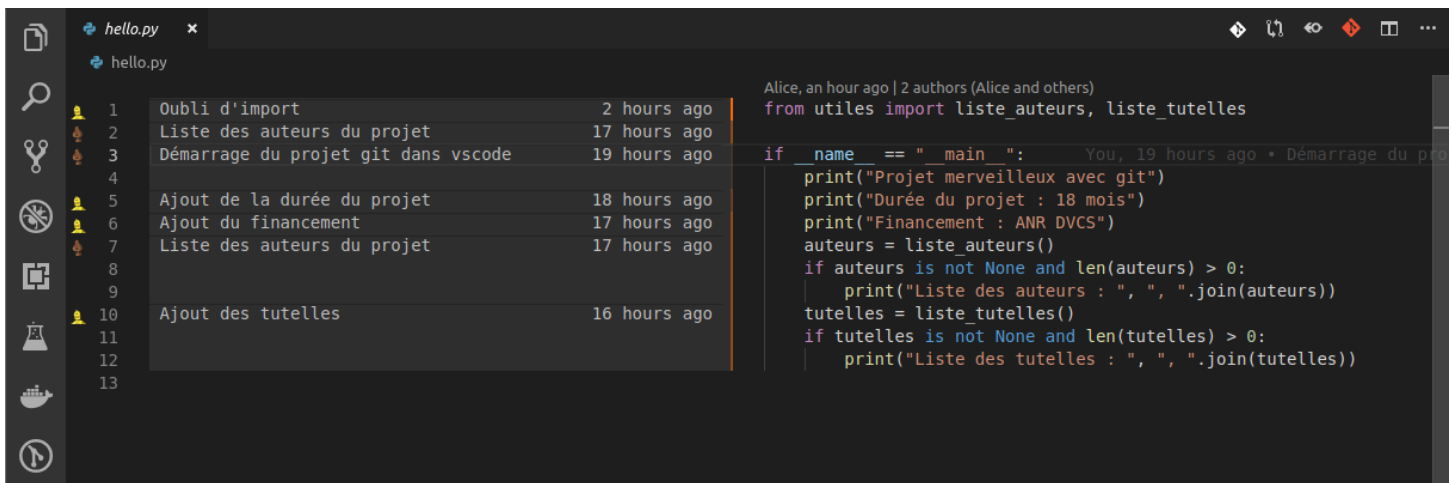


Figure 26: Affichage contextuel des modifications de la ligne courante

GitLens ajoute 2 icônes dans la barre d'outils de la fenêtre d'édition :

- une icône en forme de losange, , quasiment identique à l'icône du plugin "Git History" à gauche de l'icône de fractionnement des fenêtres. Cette icône permet de visualiser qui a modifié chaque ligne du fichier

Figure 27: Affichage de type **blame** avec le plugin GitLens

= git blame

- une icône en forme de noeud avec un arc orienté, . Cette icône permet de visualiser les changements apportés au fichier courant. Quand on clique dessus, de nombreuses autres icônes apparaissent et on voit un diff de la version courante avec la version précédente. On peut naviguer dans les diff des versions successives du fichier et chercher les différentes modifications

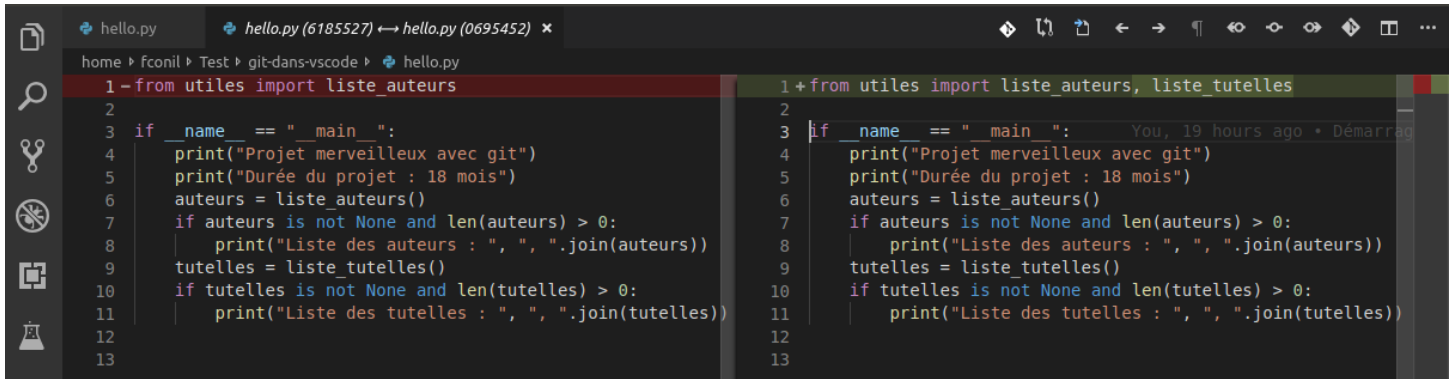


Figure 28: Affichage des modifications successives apportées au fichier avec le plugin GitLens

GitLens ajoute aussi une icône à la barre d'activité, , qui permet d'afficher une fenêtre où l'on trouve plusieurs vues :

- a new **Repositories** view to visualize, navigate, and explore Git repositories
- a new **File History** view to visualize, navigate, and explore the revision history of the current file
- a **Line History** view to visualize, navigate, and explore the revision history of the selected lines of current file
- a **Search Commits** view to search and explore commit histories by message, author, files, id, etc
- a **Compare** view to visualize comparisons between branches, tags, commits, and more

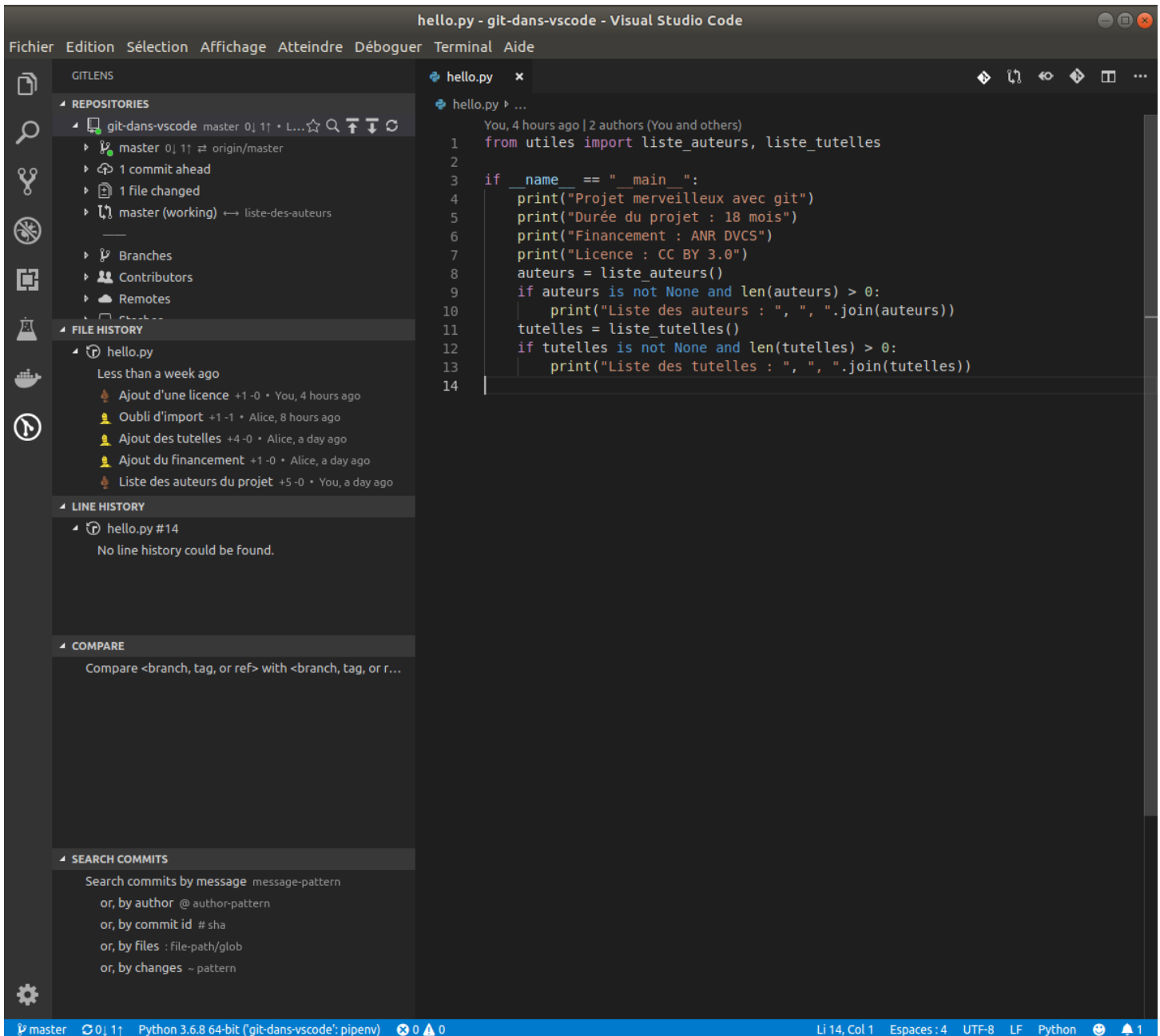


Figure 29: Différentes vues proposées par le plugin GitLens

Cet ensemble de vues peut être utile lorsque l'on travaille à plusieurs sur un projet. Il faut expérimenter.

Bug :

Malheureusement quand ce plugin est activé, il rentre en conflit avec la **vue Structure** de la **vue Explorateur** qui facilite la navigation dans les éléments d'un fichier important (liste des classes, des méthodes, des fonctions, ...)

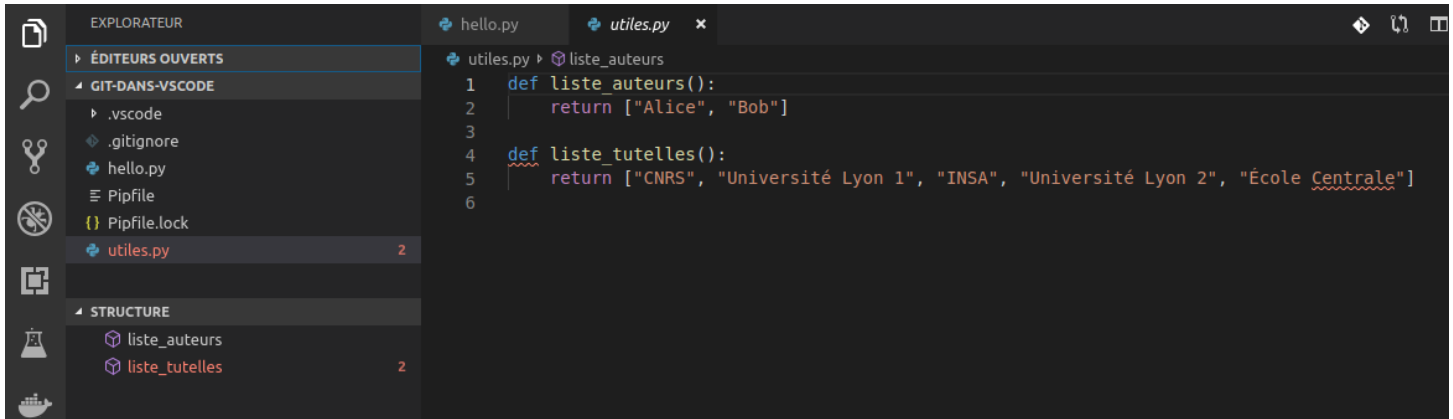


Figure 30: La **vue Structure** ne fonctionne plus quand le plugin GitLens est activé sur un projet

Barre d'activité vscode