

TP d'introduction à Git

Attention : ce TP est à réaliser en dehors de la machine virtuelle utilisée pour les deux premiers TP, et présuppose que Git et Visual Studio Code sont (correctement) installés sur votre machine hôte ! Les versions de VS Code évoluent chaque mois, et il est possible que les illustrations du sujet soient différentes de votre version. Assurez-vous d'avoir la **dernière** version.

Exercice 1 : premiers pas avec Git

Dans ce premier exercice, vous allez aborder l'utilisation de Git à travers l'interface graphique mise à votre disposition dans Visual Studio Code. Le support de Git est intégré à VS Code, mais ce dernier ne joue que le rôle *d'interface* avec la version de Git installée sur le système (2.0.0 ou supérieure).

1. Avant tout, commencez par configurer votre identité git, à l'aide de la commande

```
git config --global --edit
```

💡 Sous Windows, vous pouvez ouvrir une *Invite de commandes* ou *Powershell* ; sous Linux ou MacOS, vous utiliserez un *Terminal*.

💡 Normalement, une section `[user]` a été créé automatiquement lors de l'installation de git, mais son contenu est commenté. Décommentez les lignes `name` et `email` (vous pouvez bien sûr modifier leur contenu).

```
[user]
  name  = johndoe
  email = john@doe.com
```

De plus, cette commande ouvre votre éditeur de texte par défaut ; vous pouvez le modifier en modifiant la variable `core.editor` ; par exemple, pour choisir VS Code, rajoutez les lignes suivantes (le `--wait` est indispensable) :

```
[core]
  editor = code --wait
```

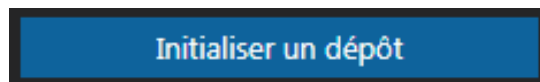
2. Créez un **dossier de travail** (*working directory* dans la terminologie Git) (peu importe son emplacement sur le disque) nommé `testgit-vscode`, et **ouvrez le dans VS Code**.
3. Cliquez sur le bouton **Contrôle de code source** dans la **barre d'activités** (généralement tout à gauche) de VS Code :



💡 Si la barre d'activités est masquée, vous pouvez l'afficher en cliquant sur le menu **Affichage** puis **Apparence**, puis **Afficher la Barre d'activités**.

💡 Si votre version de VS Code est en anglais, vous pouvez installer le plug-in *French Language Pack for Visual Studio Code* depuis la section *Extensions* de l'éditeur.

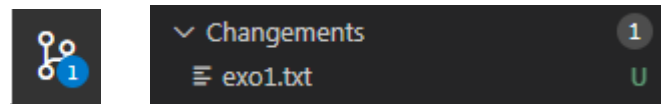
4. Cliquez ensuite sur le bouton



Le dossier est désormais configuré pour que son contenu soit versionné et contrôlé par Git.

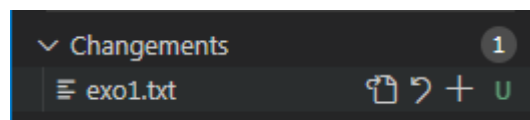
💡 Ceci se traduit par l'apparition d'un sous-dossier caché nommé `.git` dans notre dossier.

5. Créez un fichier nommé `exo1.txt` dans ce dossier, et complétez-le de quelques lignes de texte (peu importe le contenu). Git signale qu'un changement a eu lieu dans le dossier :

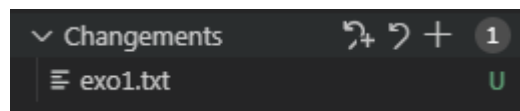


💡 La lettre 'U' signifie que ce fichier n'est pas encore suivi par Git (*Untracked*).

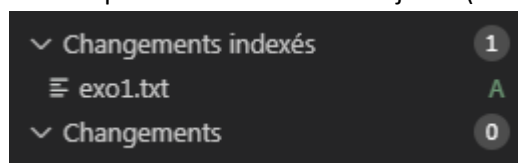
6. Ajoutez le fichier `exo1.txt` à la **zone de transit** (*staging area*) en cliquant sur le symbole qui apparaît quand vous survolez le fichier :



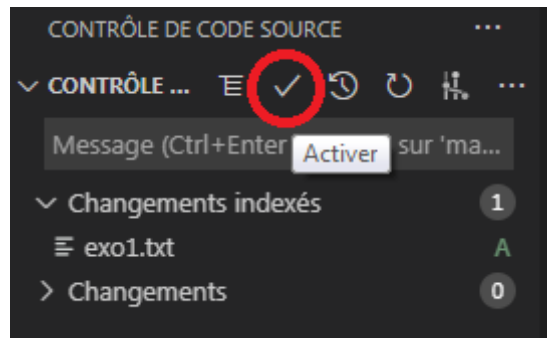
💡 Quand vous travaillez sur plusieurs fichiers, vous pouvez également utiliser le '+' situé en face de *Changements*, ce qui a pour effet de passer *tous* les changements en zone de transit



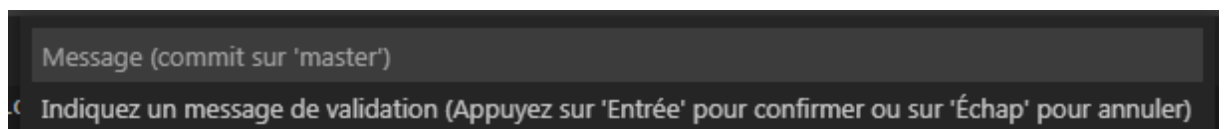
Git nous informe avec la lettre 'A' que le fichier a bien été ajouté (*Added*) en zone de transit :



Pour l'instant, les modifications apportées à `exo1.txt` ne se trouvent que dans une zone temporaire ; il n'existe pas encore de version numérotée de ce fichier. Pour cela, on doit **valider** (*commit*) les changements, en cliquant sur le bouton ci-dessous :

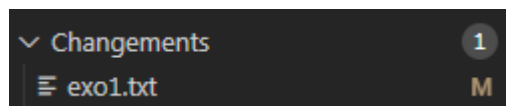


💡 Un *commit* doit **toujours** être accompagné d'un message indiquant les changements effectués. Si vous n'en spécifiez pas un avant de *commiter*, un pop-up apparaît pour vous demander de le saisir :



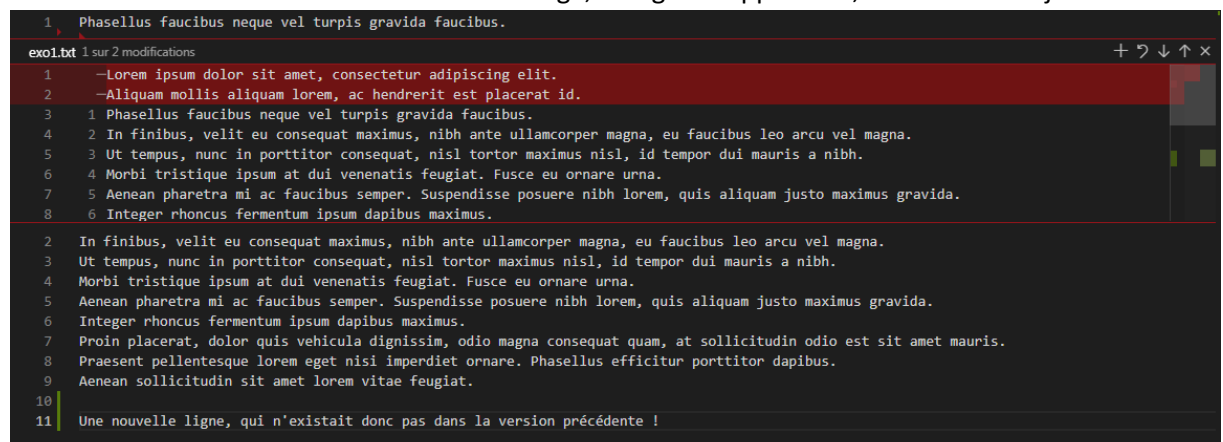
Saisissez le message « Création du fichier `exo1.txt` », puis validez. **Vous venez de créer une première version du fichier !**

7. Apportez une modification au fichier, puis enregistrez-le. Cette fois, Git devrait vous indiquer que le fichier a été modifié, avec une lettre 'M' :

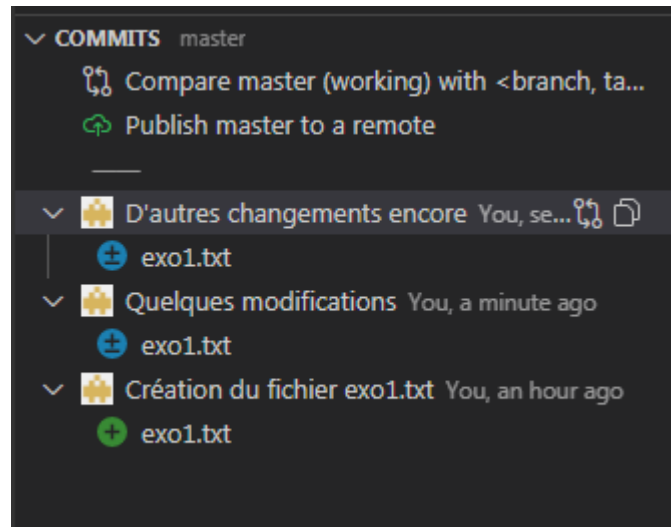


commitez-le ; puis répétez l'opération (de sorte à avoir **quatre versions** du fichier).

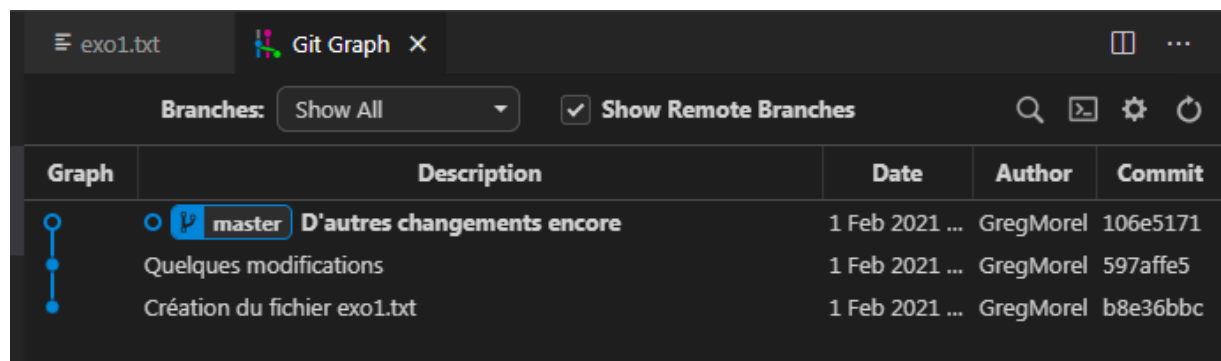
💡 Des extensions de VS Code comme **GitLens** et **Git Graph** permettent de visualiser facilement les différences d'une version à l'autre : en rouge, les lignes supprimées, en vert celles ajoutées :



On peut également visualiser la liste des *commits* (toujours classés du dernier au premier) :

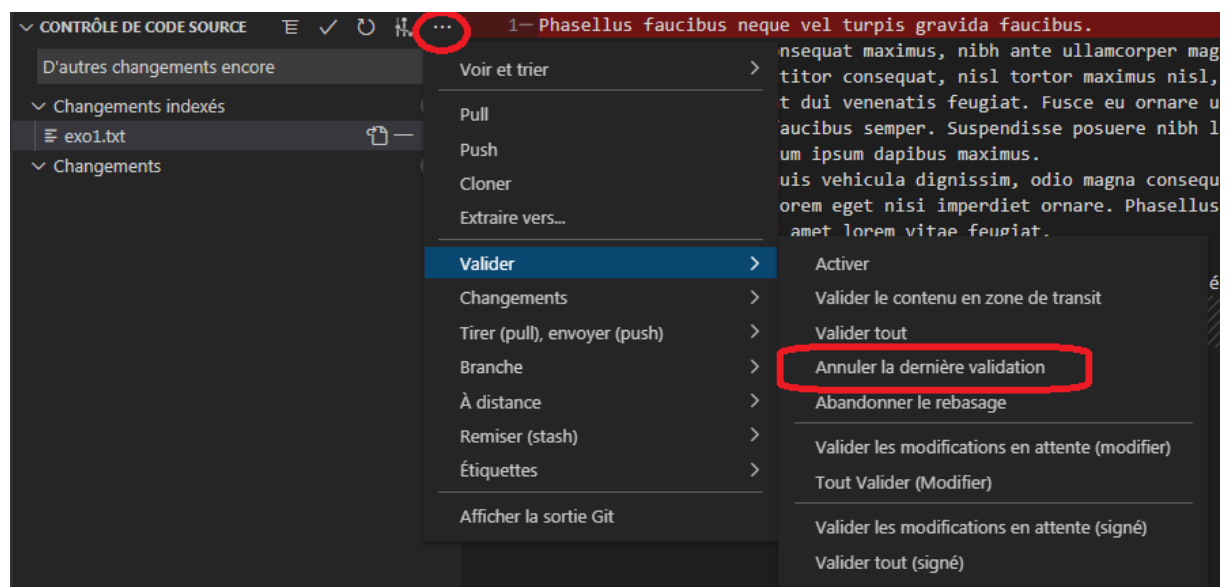


Avec GitLens

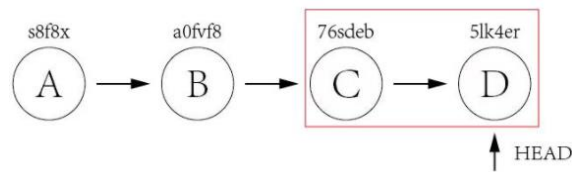


Avec Git Graph

- Il est possible d'annuler le **dernier commit** (par exemple pour corriger un oubli) ; cela **n'efface pas les modifications apportées au fichier** ; simplement, la validation est annulée et le fichier revient en zone de transit :

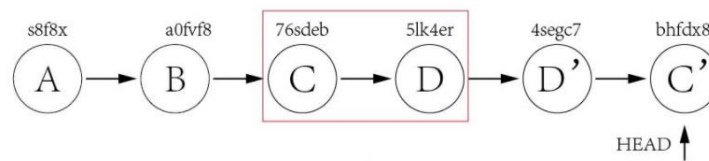


9. Supposons à présent que l'on décide de ne pas conserver les changements apportés dans les deux derniers commits, et qu'on souhaite donc revenir à la version 2. Par exemple, sur la figure ci-dessous, on souhaiterait revenir à la version B du fichier :

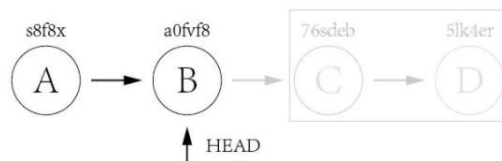


Git offre deux possibilités pour revenir à une version antérieure d'un dépôt :

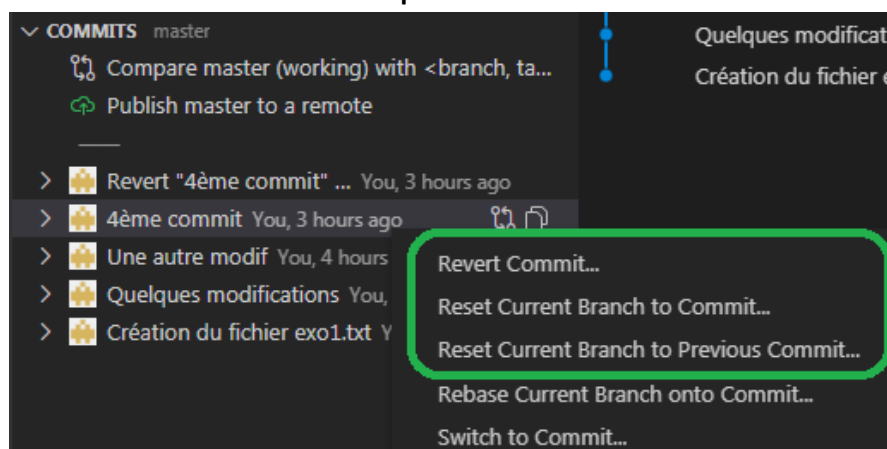
- **revert** : l'annulation d'un commit constitue un nouveau commit (l'annulation du commit D donne une nouvelle version D', puis l'annulation du commit C donne une nouvelle version C') ; l'avantage est qu'on **conserve tout l'historique des modifications** :



- **reset** : c'est une méthode plus radicale et moins recommandée car on **réinitialise** l'historique à un état antérieur, perdant ainsi les modifications qui avaient apportées par la suite :



Etrangement, il n'existe pas dans VS Code de moyen simple (graphique) de procéder à un *revert* ou un *reset* (du moins jusqu'à la version 1.52). C'est cependant très simple si on se tourne vers des extensions comme *Git History*, *Gitlens* ou *Git Graph* déjà évoqués ci-dessus. Par exemple, avec GitLens, il suffit d'un **clic droit dans l'historique** :



Installez ces extensions et procédez à un *reset* pour revenir à la version 2 du fichier.

Vous disposez désormais des bases nécessaires à l'utilisation de Git !

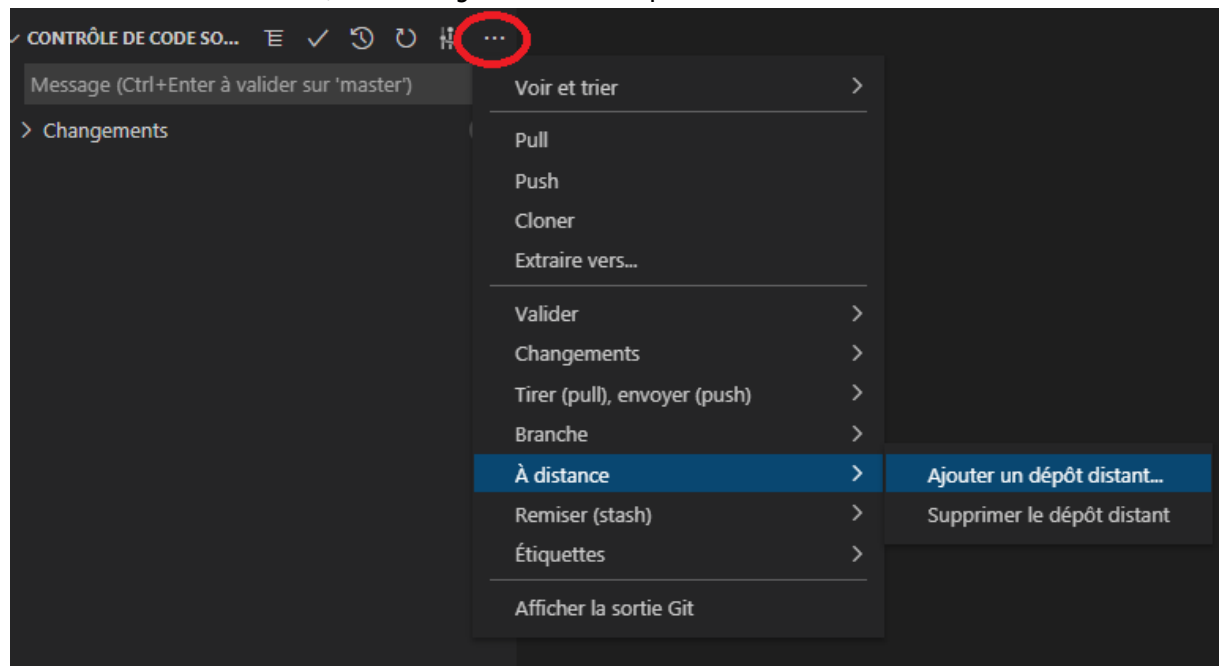
Exercice 2 : travail avec un dépôt distant

Dans le premier exercice, tout le travail a été fait en *local*, dans le dossier *testgit-vscode*. Dans ce second exercice, nous apprendrons à pousser nos modifications vers un dépôt distant (pour sauvegarder un travail en ligne, ou pour collaborer avec d'autres personnes...). Nous travaillerons avec GitHub.

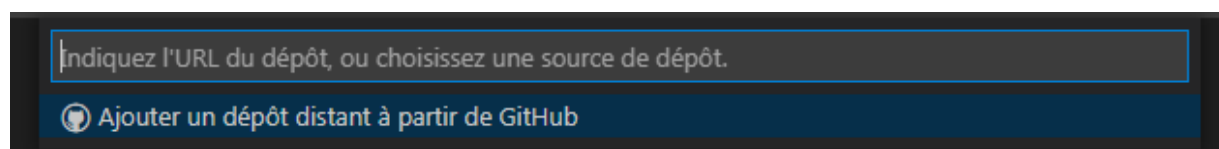
1. Commencez par créer un compte sur le site github.com si vous n'en avez pas déjà un, puis et créez un nouveau dépôt **public** en cliquant sur le bouton New (vous pouvez l'appeler **TP-Git** par exemple) :



2. L'étape suivante consiste à indiquer à VS Code que notre dépôt local (le dossier *testgit-vscode* créé au début de l'exercice 1) doit être lié à notre nouveau dépôt distant (*TP-Git*). Pour cela, dans la section « Contrôle de code source » de VS Code, cliquez sur ... comme ci-dessous, puis sélectionnez **A distance**, et enfin **Ajouter un dépôt distant** :



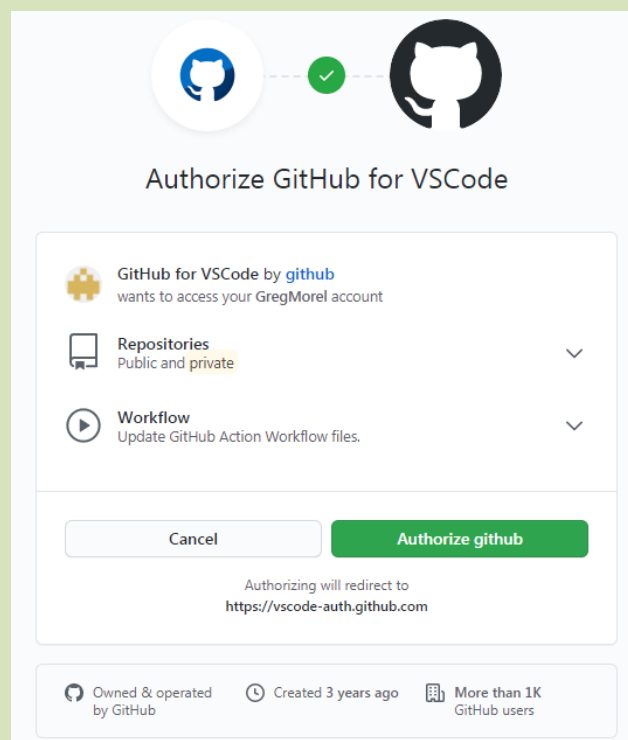
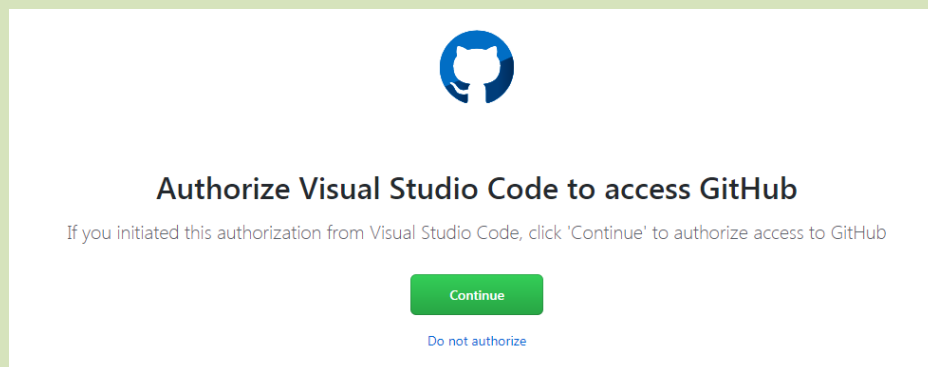
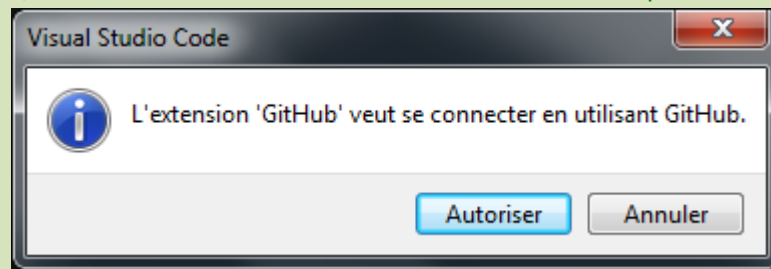
Ensuite, cliquez sur « Ajouter un dépôt distant à partir de Github » :



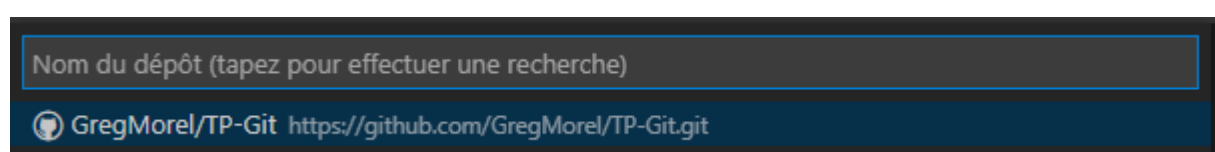
3. Cliquez ensuite sur



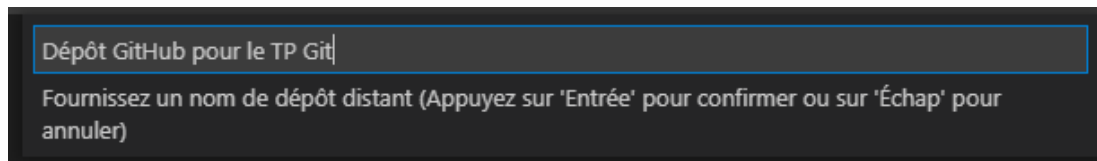
La première fois, vous devrez autoriser VS Code à accéder à votre dépôt GitHub :



Vous devriez alors voir la liste de vos projets / dépôts GitHub, dont le dépôt *TP-Git* :

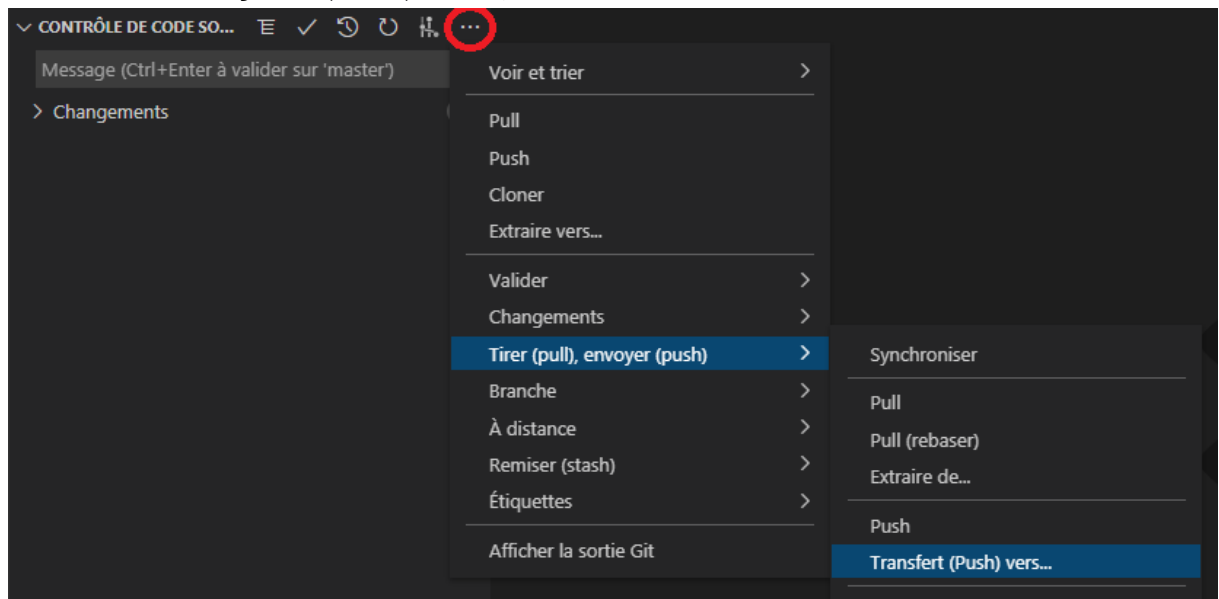


Sélectionnez le dépôt ; VS Code vous demande d'indiquer ensuite un nom permettant d'identifier rapidement ce dépôt :

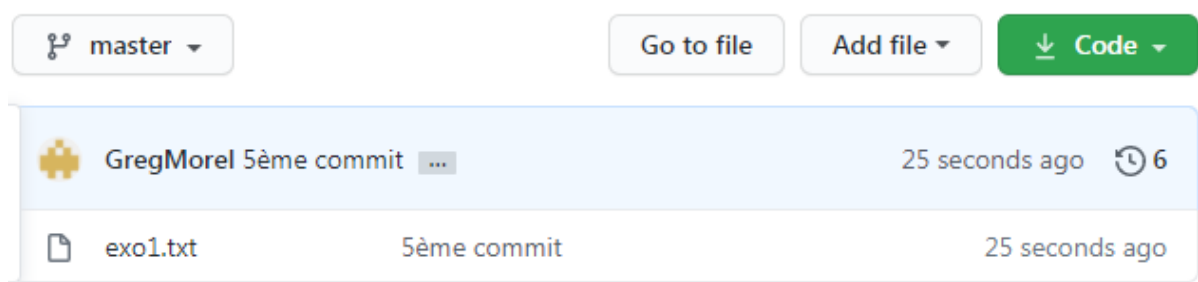


Le dépôt distant est configuré !

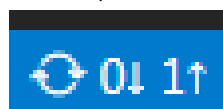
4. Pour envoyer votre travail local sur le dépôt distant, faites un **push** (la première fois, il faut sélectionner **Transfert (Push) vers**) :



Si vous actualisez votre dépôt GitHub, vous devriez constater que les fichiers ont bien été publiés.



5. Désormais, vous pourrez publier vos futures modifications sur votre dépôt GitHub par un simple **Push**, ou même en cliquant sur l'icône suivante, située en bas de la fenêtre VS Code :



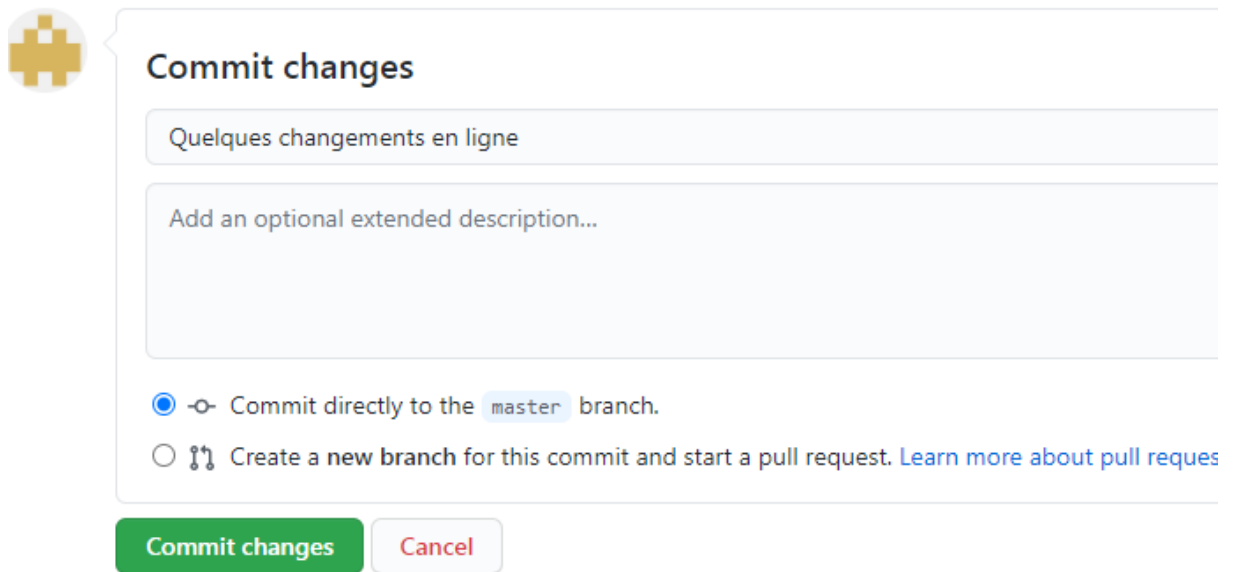
Celle-ci indique dans le cas présent qu'il existe *un* commit local à envoyer sur le dépôt distant, et qu'il n'existe pas de commit distant à rapatrier en local.

⚠ Toute modification doit d'abord être validée en local avant d'être poussée vers le dépôt distant.

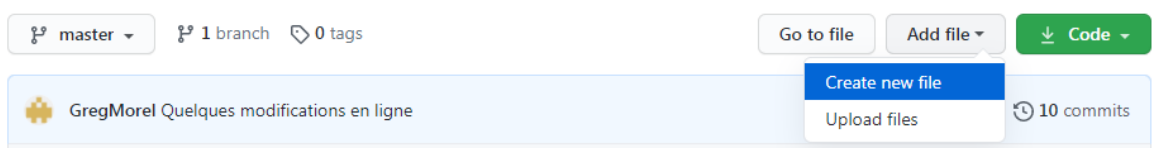
6. Nous allons à présent faire l'inverse : rapatrier en local des modifications effectuées sur le dépôt distant. Rendez-vous sur GitHub et modifiez le fichier `exo1.txt` directement en ligne, en cliquant sur **le nom du fichier** puis sur l'icône *Edit* :



N'oubliez pas de sauvegarder les modifications en cliquant sur le bouton *Commit changes* en bas de page !



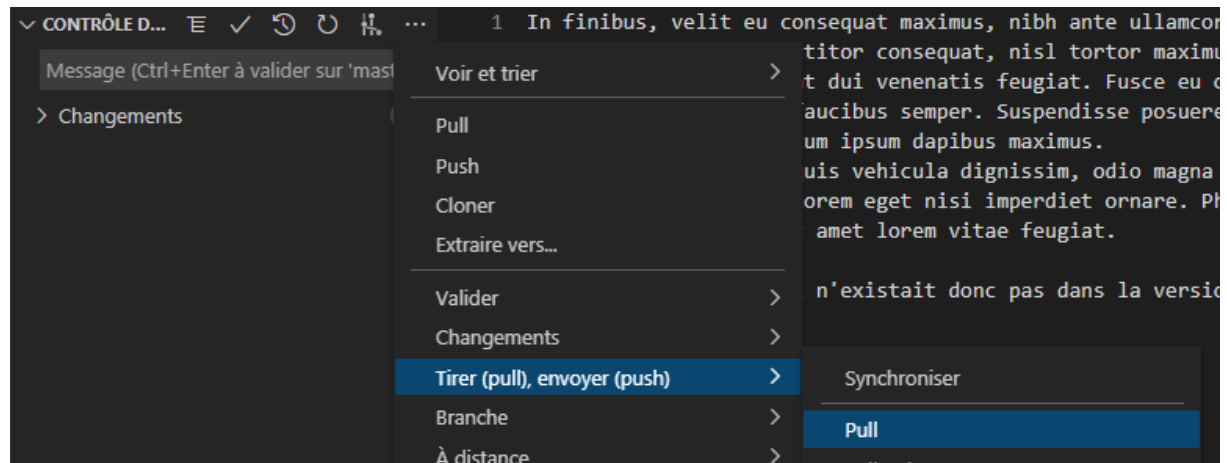
7. Sur GitHub toujours, depuis la page principale de votre dépôt, créez un nouveau fichier nommé `exo2.txt` :



8. Nous allons à présent récupérer ces changements effectués à distance vers notre dépôt local. Généralement, VS Code effectue un **git fetch** à intervalles réguliers, c'est-à-dire qu'il regarde si des changements ont eu lieu en local ou à distance, **sans procéder au téléchargement** de ces modifications. Vous devriez ainsi retrouver l'icône de tout à l'heure, indiquant cette fois qu'une modification a été effectuée sur le dépôt distant :



Pour rapatrier les modifications de manière effective, vous pouvez soit cliquer sur ce bouton, soit effectuer un **git pull** :



Vous savez désormais utiliser un dépôt local et un dépôt distant

Exercice 3 : résolution des conflits

Dans ce troisième exercice, nous allons aborder une tâche récurrente dans un contexte de travail collaboratif : la gestion des conflits sur les fichiers.

Imaginez que vous réalisez un projet en binôme ; vous créez un dépôt sur GitHub pour centraliser tout le travail effectué et éviter les échanges de fichiers par mail ou clé USB. Chacun des deux étudiants dispose d'une copie locale du dépôt, ce qui lui permet de travailler indépendamment de l'autre étudiant ; régulièrement il pousse ses modifications sur le dépôt en ligne pour que l'autre étudiant en ait connaissance.

Mais que se passe-t-il si vous essayez de pousser un fichier qui a été entretemps modifié par votre binôme ? Dans la plupart des cas, Git arrivera à réaliser seul la fusion des modifications ; mais il y a des situations où c'est à vous d'indiquer explicitement les modifications à conserver. On parle de **résolution de conflits**.

1. Créez un fichier **exo3.txt** (soit localement soit en ligne), et complétez-le avec le texte suivant ; puis synchronisez les deux dépôts.

- *Voici mon secret. Il est très simple : on ne voit bien qu'avec le cœur. L'essentiel est invisible pour les yeux.*
- *L'essentiel est invisible pour les yeux, répéta le petit prince, afin de se souvenir.*
- *C'est le temps que tu as perdu pour ta rose qui fait ta rose si importante.*
- *C'est le temps que j'ai perdu pour ma rose..., fit le petit prince, afin de se souvenir.*
- *Les hommes ont oublié cette vérité, dit le renard. Mais tu ne dois pas l'oublier. Tu deviens responsable pour toujours de ce que tu as apprivoisé. Tu es responsable de ta rose...*

2. Dans la version **en ligne**, ajoutez la ligne suivante à la fin du fichier :

Le Petit Prince (Saint-Exupéry, 1943).

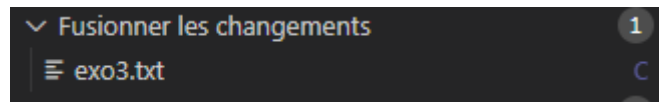
Cette modification représente un changement apporté par votre binôme.

3. Modifiez la version **locale**, ajoutez quelques lignes de texte au début du fichier (peu importe le contenu) ; et à la fin du fichier, la ligne suivante :

Extrait du livre "Le Petit Prince" d'Antoine de Saint-Exupéry, paru en 1943

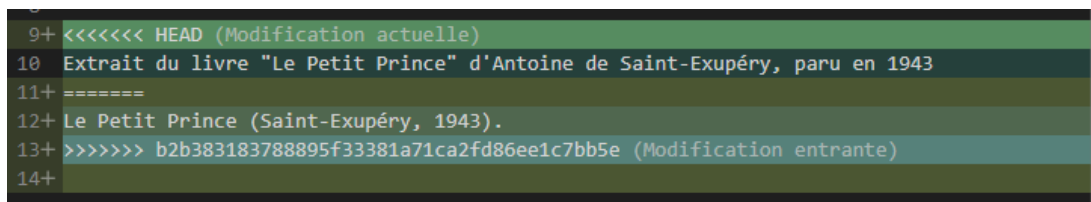
Validez vos modifications **en local** (commit).

4. Essayez à présent de pousser vos modifications en ligne (push). Vous devriez rencontrer un conflit, symbolisé par la lettre C dans l'outil de contrôle de code source :

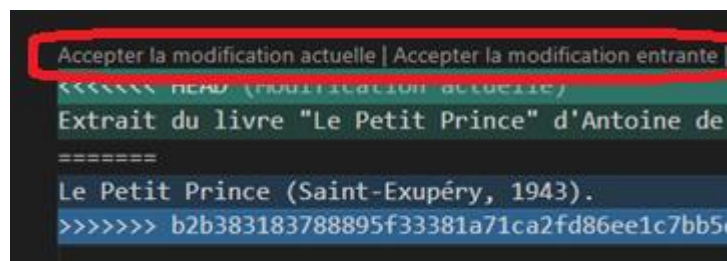


Git a partiellement réussi à résoudre le conflit :

- Les premières lignes ont été correctement ajoutées
- Il existe un conflit sur la dernière ligne : Git ne sait pas quelle version conserver (la vôtre, ou celle de votre binôme ?)



Vous pouvez résoudre ce conflit « à la main », en ne conservant que le texte qui doit persister dans la nouvelle version. Mais le plus simple est de cliquer sur le fichier conflictuel ; VS Code vous propose alors pour chaque conflit la possibilité de conserver la modification actuelle ou la modification entrante :

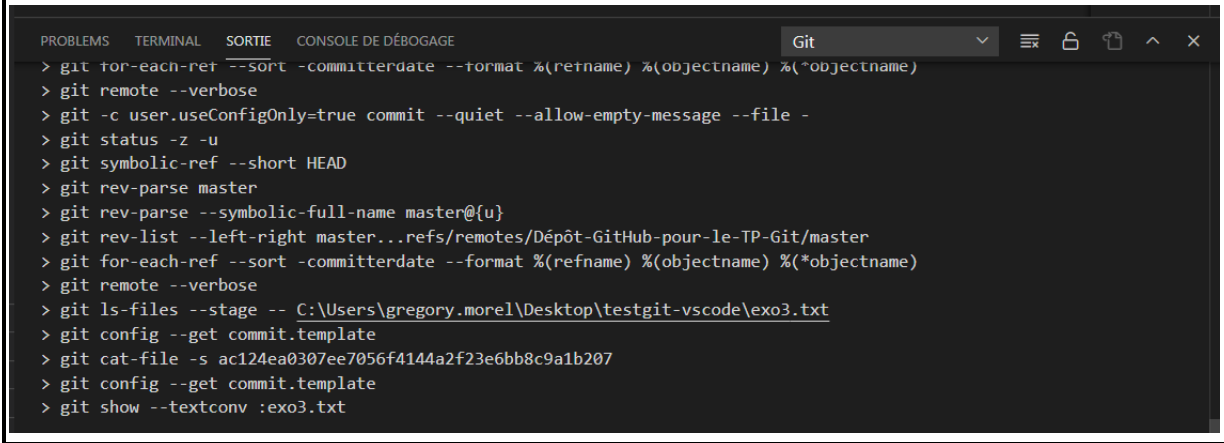


5. Acceptez la modification entrante, puis terminez le push et vérifiez qu'en ligne, vous obtenez bien un fichier contenant à la fois les premières lignes ajoutées localement, et la dernière ligne ajoutée « en ligne, par votre binôme ».

Exercice 4 : Git en ligne de commandes

Recommencez les exercices 1 et 2 en vous servant cette fois-ci de la ligne de commande exclusivement.

💡 Toutes les actions effectuées avec Git dans Visual Studio Code apparaissent dans la fenêtre *sortie*, ce qui peut vous aider à comprendre certaines commandes :



```
> git for-each-ref --sort -committerdate --format %(refname) %(objectname) %(^objectname)
> git remote --verbose
> git -c user.useConfigOnly=true commit --quiet --allow-empty-message --file -
> git status -z -u
> git symbolic-ref --short HEAD
> git rev-parse master
> git rev-parse --symbolic-full-name master@{u}
> git rev-list --left-right master...refs/remotes/Dépôt-GitHub-pour-le-TP-Git/master
> git for-each-ref --sort -committerdate --format %(refname) %(objectname) %(^objectname)
> git remote --verbose
> git ls-files --stage -- C:\Users\gregory.morel\Desktop\testgit-vscode\exo3.txt
> git config --get commit.template
> git cat-file -s ac124ea0307ee7056f4144a2f23e6bb8c9a1b207
> git config --get commit.template
> git show --textconv :exo3.txt
```