Projet dirigé

Git, pour le cours de KBE.

SALIHA YACOUB, LUC LEDOUX

GIT pour KBE, Projet Dirigé

Plan de la séance

- Définitions
- Avantages
- Git serveur et Git local
- Les dépôts
- Installation
- Créer le dépôt distant et locaux
- Les branches
- Application
- Résolution de conflits
- Retour à la version précédente

Saliha Yacoub, Luc Ledoux

Définition

- Qu’est-ce que la gestion de version?
  - La **gestion de version** (en anglais : version control ou revision control) consiste à maintenir **l'ensemble des versions d'un ou plusieurs fichiers** (généralement en texte) afin d’en **suivre l’évolution** au cours du temps. Essentiellement utilisée dans le domaine de la création de logiciels, elle concerne surtout la **gestion des codes source**.

  - Un gestionnaire de versions est un programme (logiciel) qui permet aux développeurs de conserver un historique des modifications et des versions de tous leurs fichiers..

Définition

- Un gestionnaire de versions permet de garder en mémoire :
  - chaque modification de chaque fichier ;
  - pourquoi elle a eu lieu ;
  - et par qui
- Un gestionnaire de version permet donc de:
  - Revenir à une version précédente de votre code en cas de problème.
  - Suivre l'évolution de votre code étape par étape.
  - Travailler à plusieurs sans risquer de supprimer les modifications des autres collaborateurs.

Avantages

- Ramener un fichier à un état précédent
- Ramener le projet complet à un état précédent
- Visualiser les changements au cours du temps
- Suivre l’évolution du développement du logiciel
- Développement collaboratif de logiciels
- Identifier qui a introduit un problème et quand

Git local et Git serveur

- Git est de loin le système de contrôle de versions le plus largement utilisé aujourd'hui. C’est un programme qui a une structure **décentralisée ou distribuée**.
- Git et GitHub sont deux choses différentes.
- **Git est un gestionnaire de versions**. Vous l’utiliserez pour créer un dépôt local et gérer les versions de vos fichiers.
- **GitHub est un service en ligne** qui va héberger votre dépôt. Dans ce cas, on parle de **dépôt distant** puisqu’il n’est pas stocké sur votre machine.
- Il existe d’autres outils de gestion de versions: GitLab, BitBusket

Les dépôts

- Un dépôt est comme un dossier qui conserve un historique des versions et des modifications d'un projet. Il peut être local ou distant.
- Un **dépôt local** est un entrepôt virtuel de votre projet. Il vous permet d'enregistrer les versions de votre code et d'y accéder au besoin.
- Le **dépôt distant** est un peu différent. Il permet de stocker les différentes versions de votre code afin de garder un **historique délocalisé**, c'est-à-dire un historique hébergé sur Internet ou sur un serveur d'un réseau. L'intérêt d'un dépôt distant est dans les suivants:
  - Si votre machine locale est en panne
  - **Si vous travaillez en équipe**

Les dépôts

- Le dépôt distant est donc un type de dépôt indispensable si vous travaillez à plusieurs sur le même projet, puisqu'il permet de centraliser le travail de chaque développeur. C'est aussi sur le dépôt distant que toutes les modifications de tous les collaborateurs seront fusionnées.

- Votre dépôt local est un clone de votre dépôt distant. C'est sur votre dépôt local que vous ferez toutes vos modifications de code.

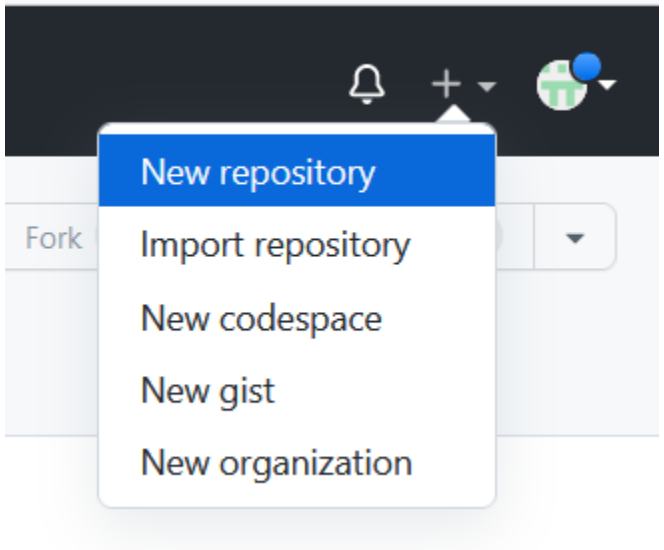
Installation

Les informations qui suivent vont vous permettre d'utiliser GitHub dans le cadre de votre cours de KBE, projet dirigé.

- Créer un compte GitHub. Aller sur https://github.com pour créer votre compte.
- Il est important de choisir la version gratuite si vous ne voulez pas avoir des frais. GitHub appartient à Microsoft.
- Installer GitHub Desktop.
- Lancer l'installation. Suivez les informations par défaut.

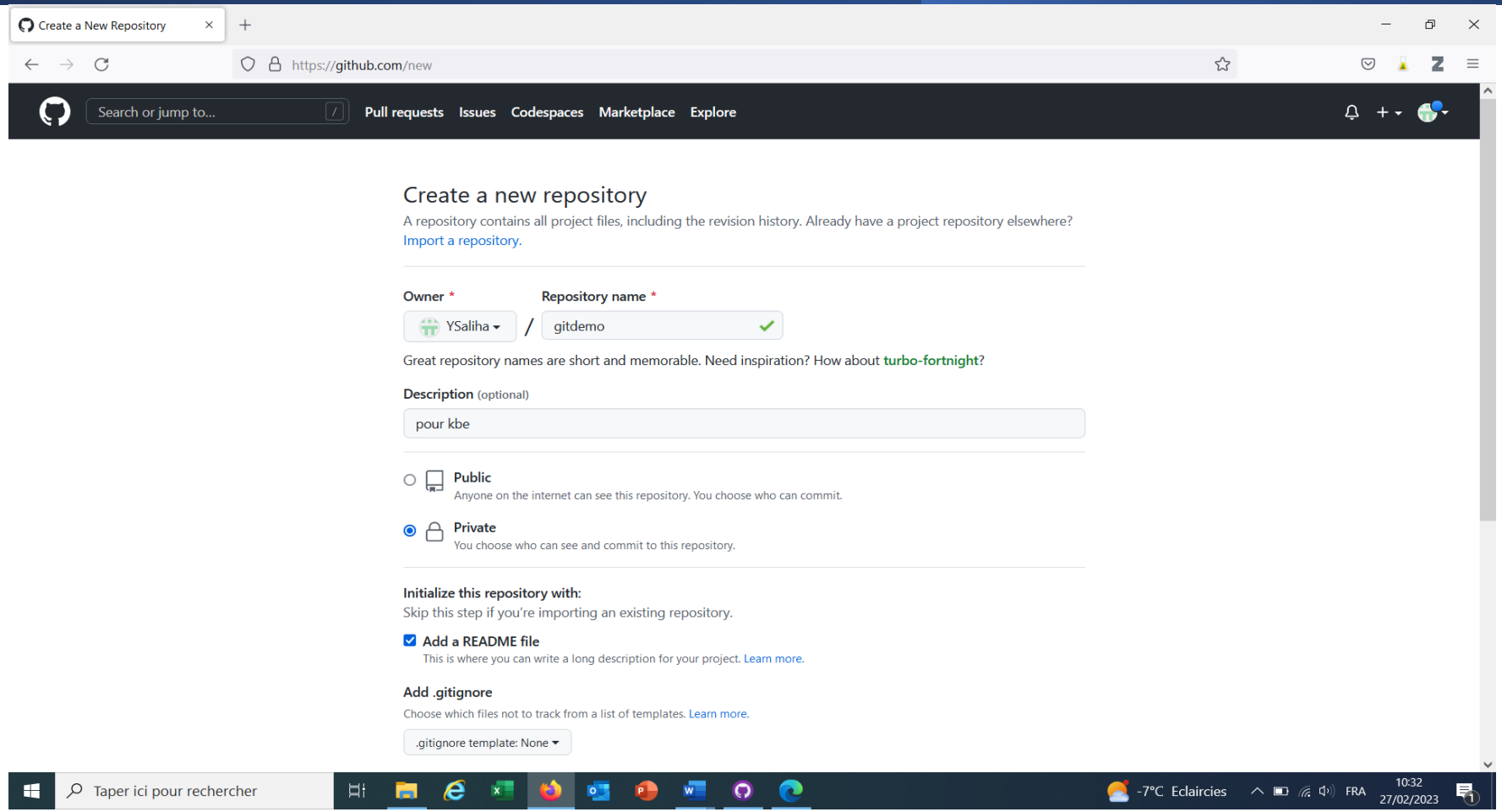
Créer le dépôt distant (sur le Web→GitHub)

- Rendez-vous sur votre GitHub, pour la première fois, vous allez voir un bouton **New**, puis créer votre repository
  - Donner un nom significatif : Exemple **projet101.**
  - Il faut qu'il soit "private"
  - Mettre une petite description
  - Ajouter un fichier READEME (peut vous servir à ajouter des commentaires plus tard)
- Sinon par le bouton **+ , New repository**
- **gitignore** est un fichier qui permet d'ignorer certains fichiers de votre projet Git. Nous reviendrons là-dessus plus tard.



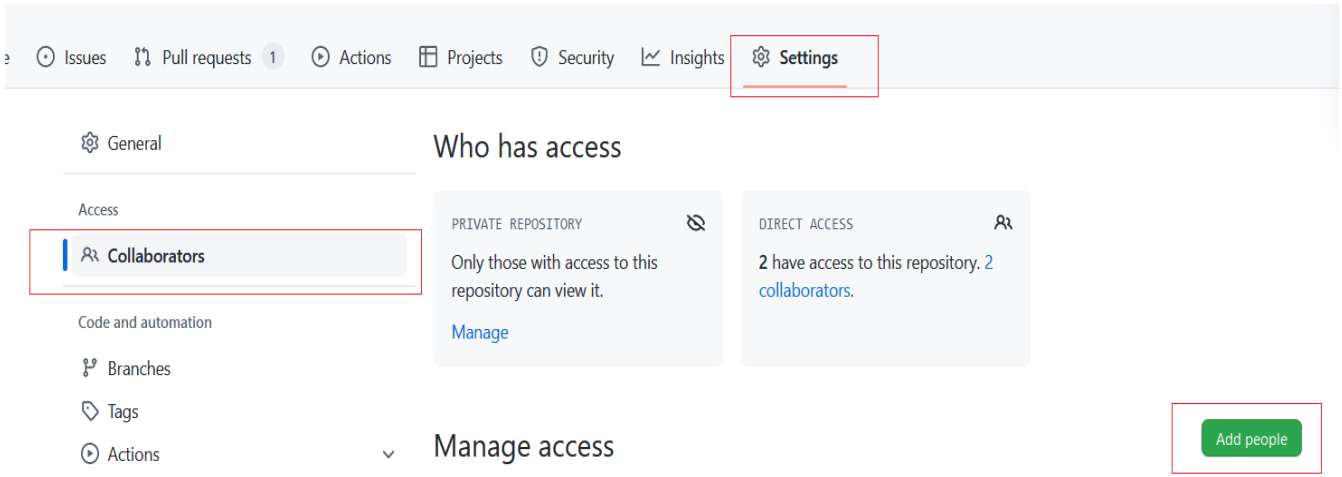
Exemple

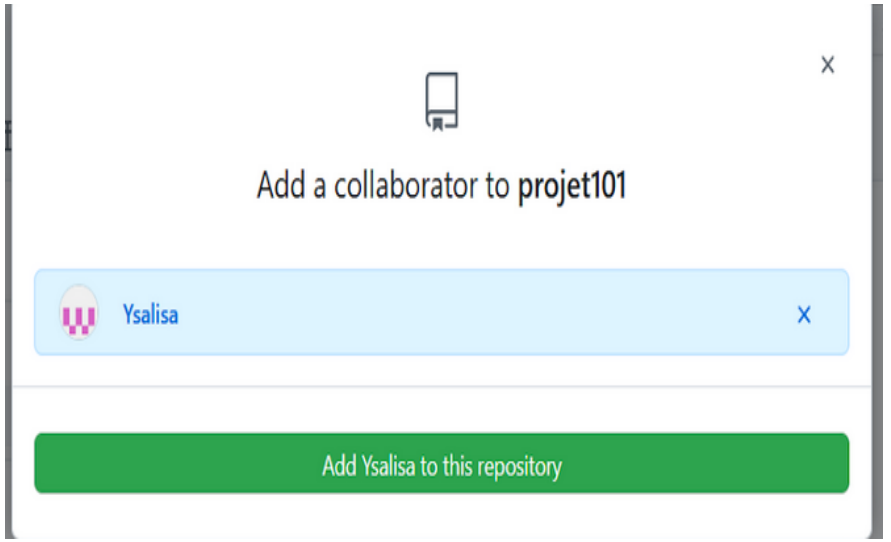
- Exemple



Inviter des collaborateurs: Vos équipiers

- Pour être un collaborateur, il faut avoir un compte GitHub.
- Par le menu : Settings-->Collaborators-->Add people.

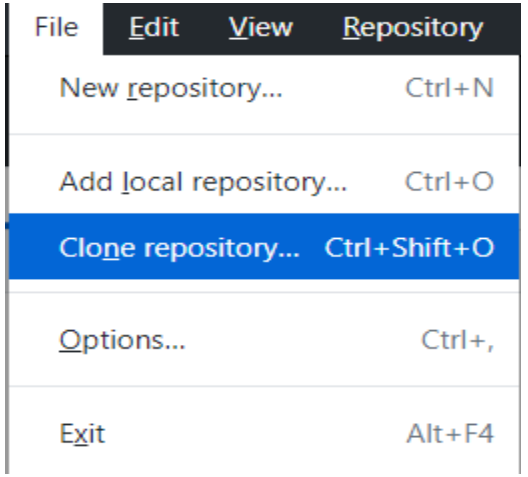




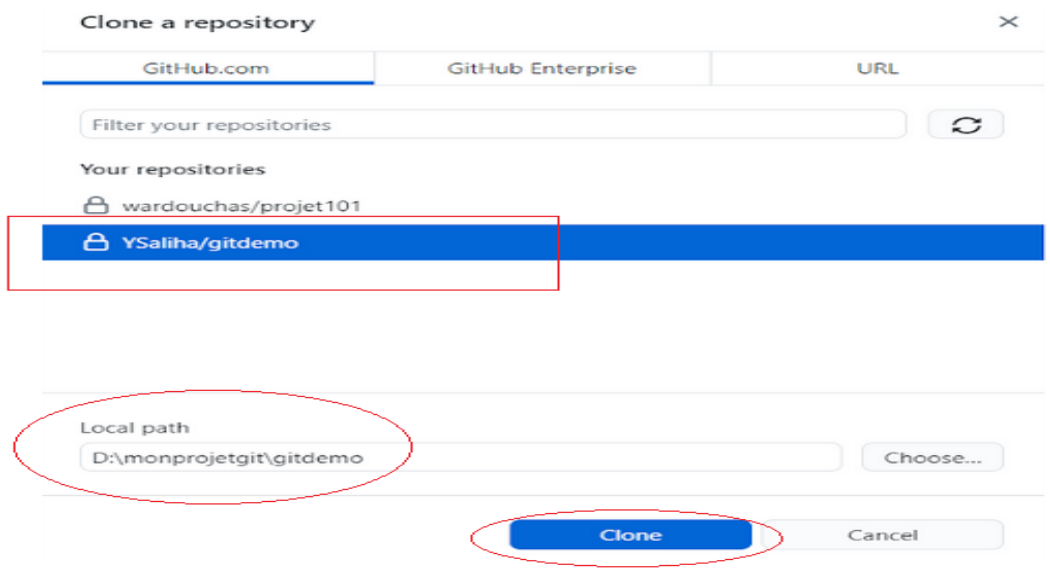
- Ceux qui reçoivent l'invitation doivent accepter l'invitation pour faire partie de l'équipe de collaborateurs.
- Vous devez faire une invitation à votre enseignant(e)

Le dépôt local →GitHub Desktop

- Nous pouvons créer un dépôt local normalement, nous allons voir les étapes plus loin. Nous pouvons aussi « cloner » GitHub (à partir de l'internet) pour que tous les membres de l'équipe aient la même structure du dépôt.
- Rendez-vous sur votre espace disque (endroit où vous travailler pour le projet), puis créer un dossier qui va contenir votre dépôt. Donnez un nom significatif: **projetskbe**
- Ouvrir l'application GitHub Desktop, puis choisir: **Clone a repository from the internet ou File Clone repository**



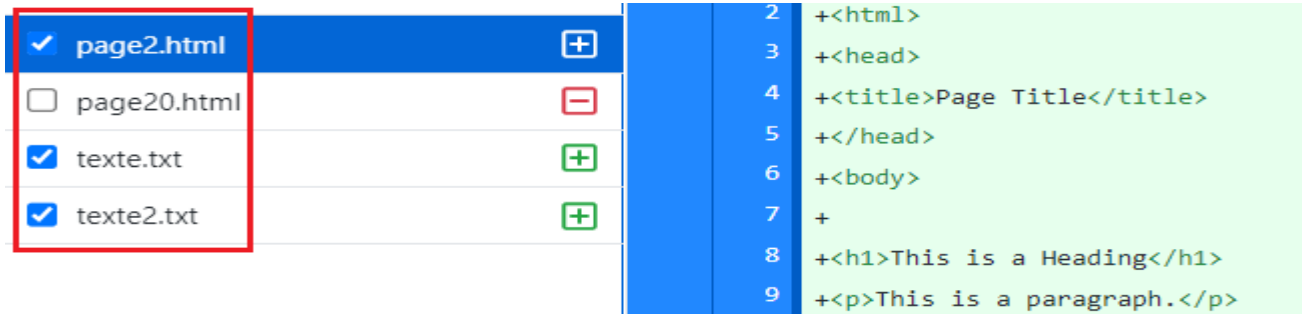
Le dépôt local →GitHub Desktop

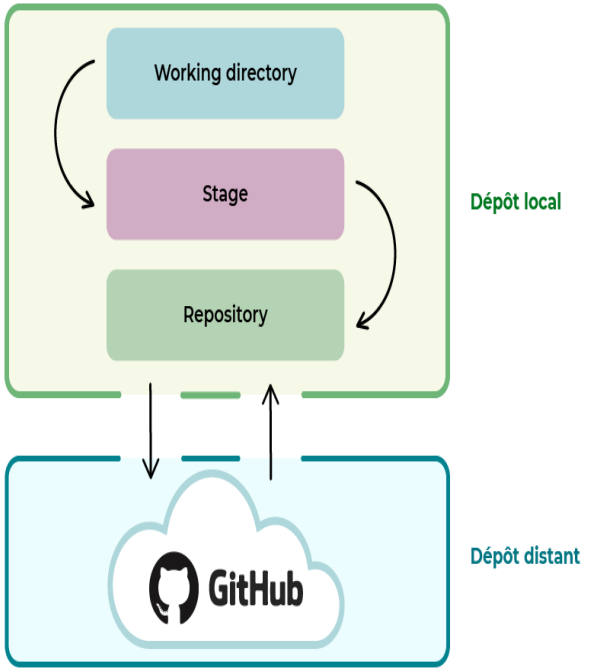


- Le repository Ysaliha/**gitdemo** est le repository à cloner
- **gitdemo** est cloné dans le répertoire monprojetgit de votre espace de travail local.
- Vérifier que le repository est bel est bien dans votre répertoire local.

État des fichiers

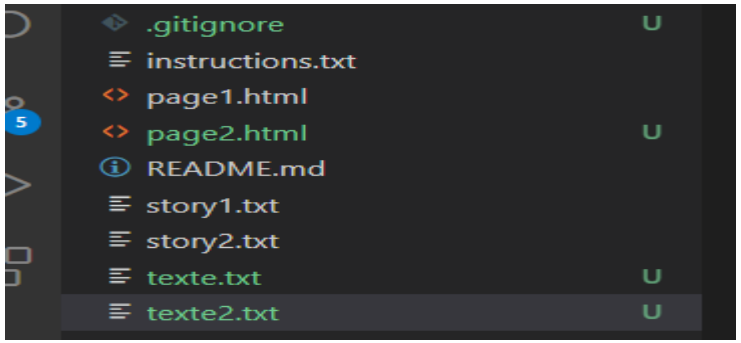
- Les fichiers actuels de votre projet se retrouvent dans le répertoire de travail (Working Directory) sur votre disque dur.
- Lorsque vous modifiez un fichier, GitDeskop le détecte et l'indique à son interface.
- Pour sauvegarder une version de votre projet, vous devez d'abord ajouter les fichiers concernés dans le "Staging Area". Ce qui correspond à la case cochée dans la figure ci-après.





État des fichiers

- Vous pouvez également le voir dans votre Visual studio code, les fichiers en verts ne sont pas encore comité



- Enfin, l'opération "Commit" permet de transférer les fichiers du "Staging Area" vers votre dépôt Git (Repository) afin d'ajouter un instantané de cette nouvelle version à l'historique et ainsi rendre permanent vos modifications.

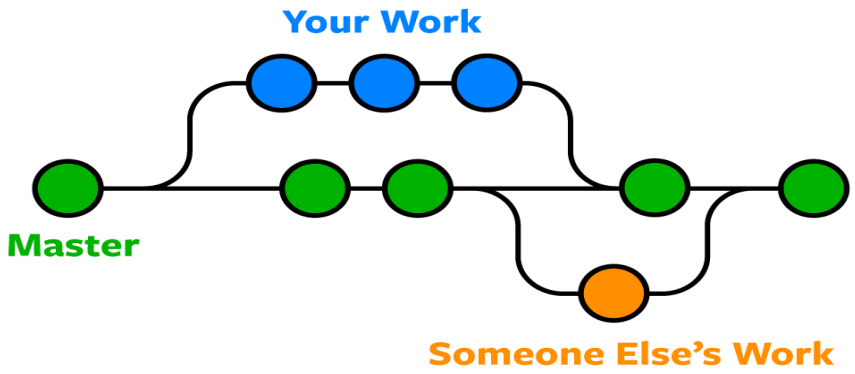
Attention: les modifications non sauvegardées dans Git pourraient être perdues.

Les branches, définitions

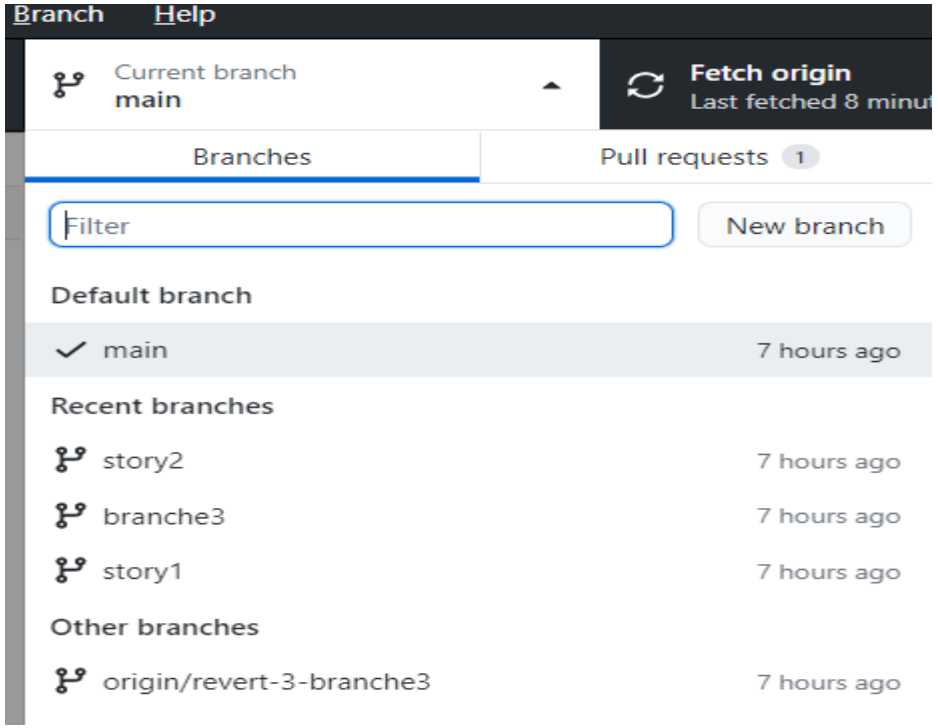
- Lorsque le projet est important, il est conseillé d'utiliser des branches pour le développement.
- Les différentes branches correspondent à des copies de votre code principal à un instant T, où vous pourrez tester toutes vos idées cela impacte votre code principal.
- Sous Git, la branche principale est appelée la ***branche main. (ou master avant 2020)***
- La branche principale (main) portera l'intégralité des modifications effectuées. **Le but n'est donc pas de réaliser les modifications directement sur cette branche**, mais de les réaliser sur d'autres branches, et après divers tests, de les intégrer sur la branche principale.

Les branches, Exemple

- La branche Master est la branche **main (depuis 2020 pour GitHub)**
- Les commit se feront sur la branche et sera fusionnée (merge) sur main une fois que votre branche est OK.
- La figure ci-dessous montre , deux autres branches en plus du main. La votre et celle de quelqu'un d'autre.

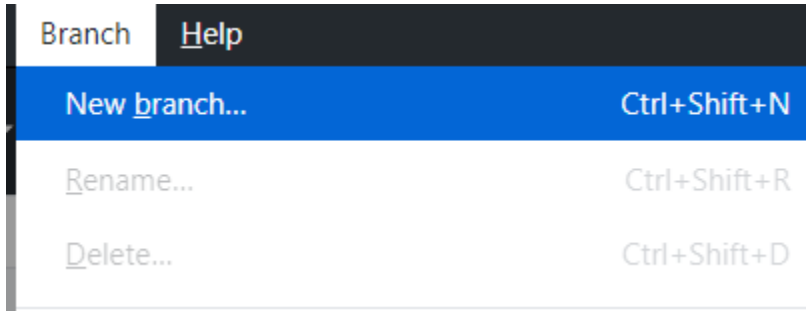


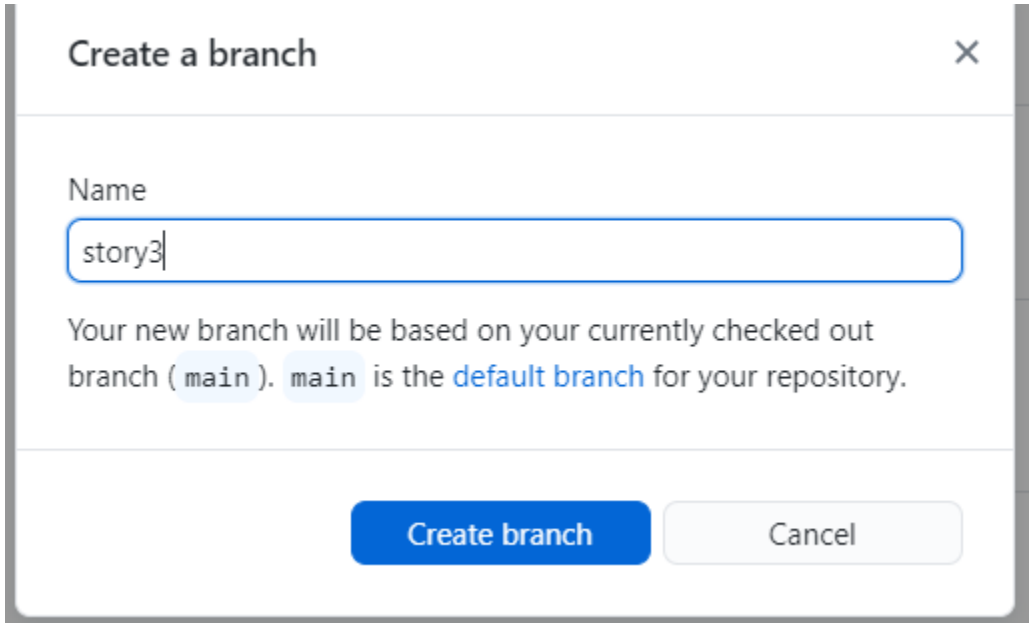
- Dans GitHub Desktop, voici la vision des branches (ci-dessous)



Les branches, création

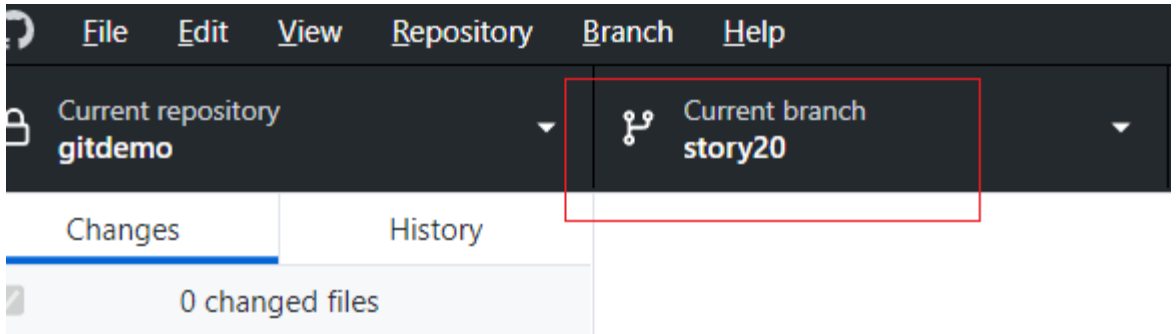
- Allez dans dans GitHub DeskTop, puis
- Par le menu Branch, faire New branch (choisir Main comme branche source)
- Donnez un nom significatif, puis faire Create branch.
- Nous vous conseillons et recommandons une brache par story.
- Une branche peut contenir plusieurs fichiers





GitHub Desktop, application

- Dans votre GitHub Desktop, assurez-vous d'être dans la bonne branche: (ici sotry20)



- Ouvrir votre Visual Studio Code
- Ouvrir le dossier de votre repository local
- Faire nouveau fichier (txt ou html). Écrire quelques lignes puis sauvegarder.
- Dans GitHub Desktop vérifier que votre page html (ou txt) est là.

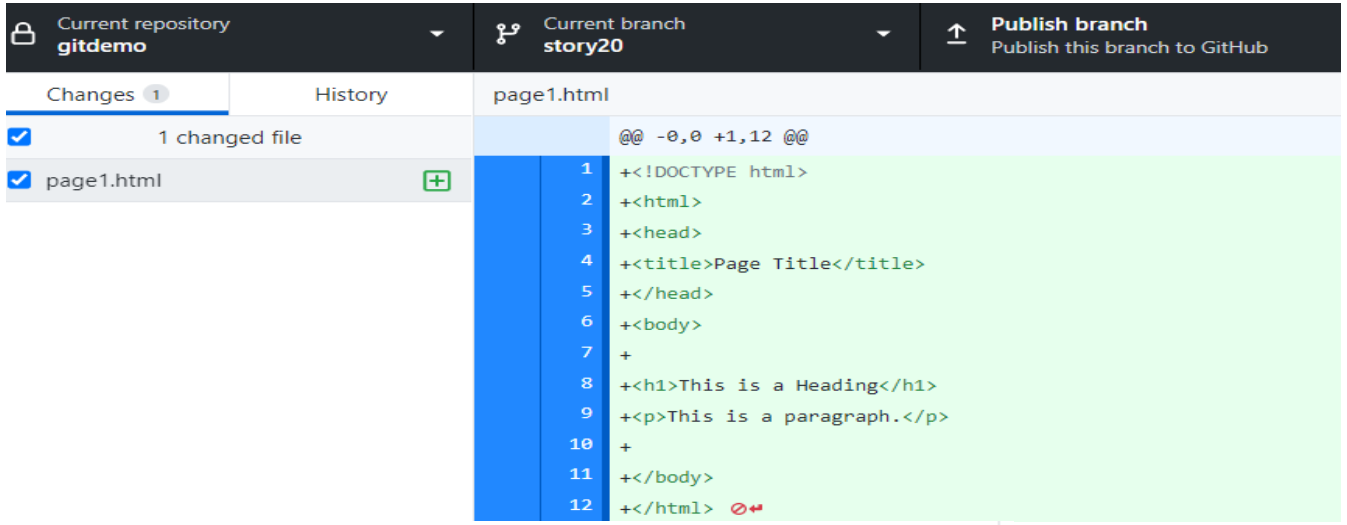
GitHub Desktop, application

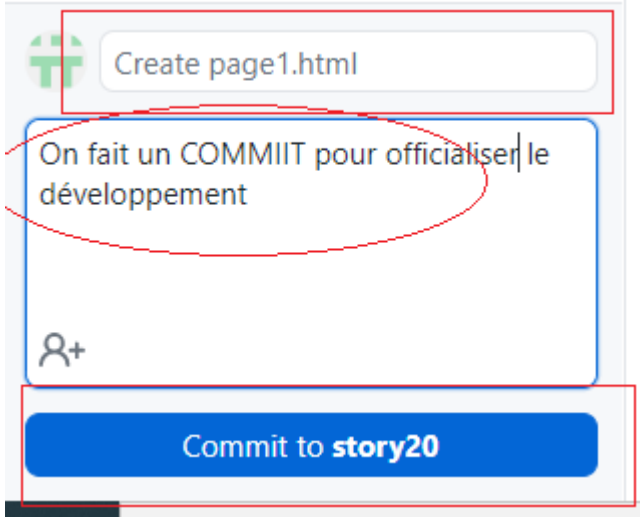
La figure suivante montre la page1.html.

Pour l'instant, le développement n'est pas officialisé.

Pour officialiser il faut faire un **commit local**

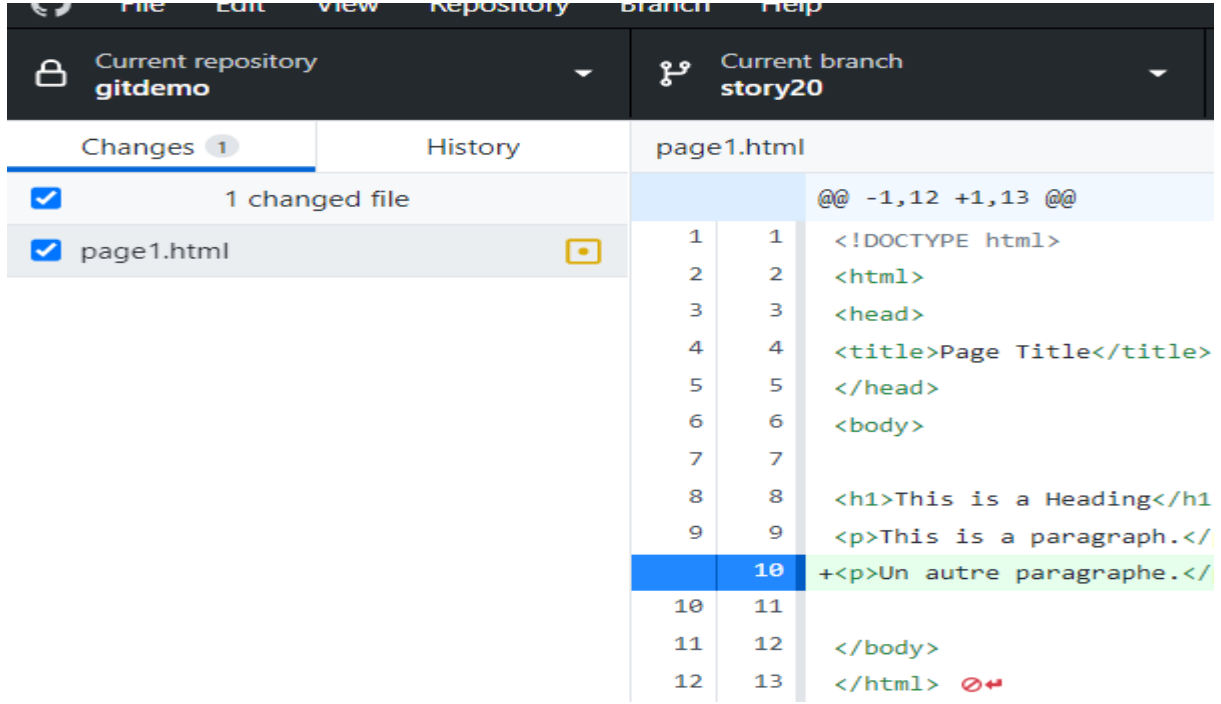
Écrire un commentaire utile qui reflète la description des modifications, puis cliquer sur Commit to **votrebranche**

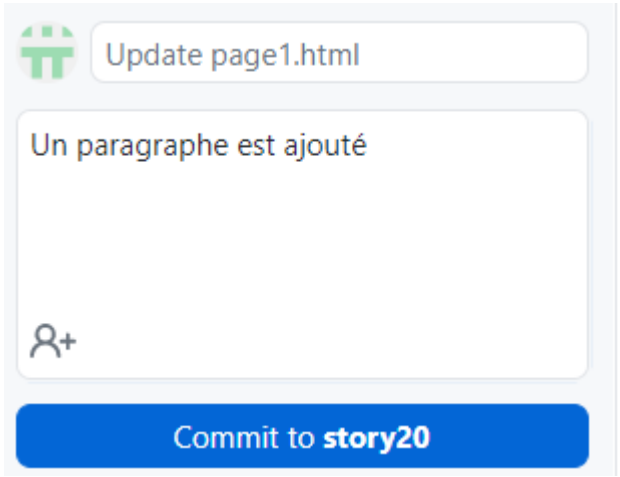




GitHub Desktop, application

Dans Visual Studio Code, modifier votre fichier. Puis allez dans GitHub Desktop pour voir que la page a été modifiée.

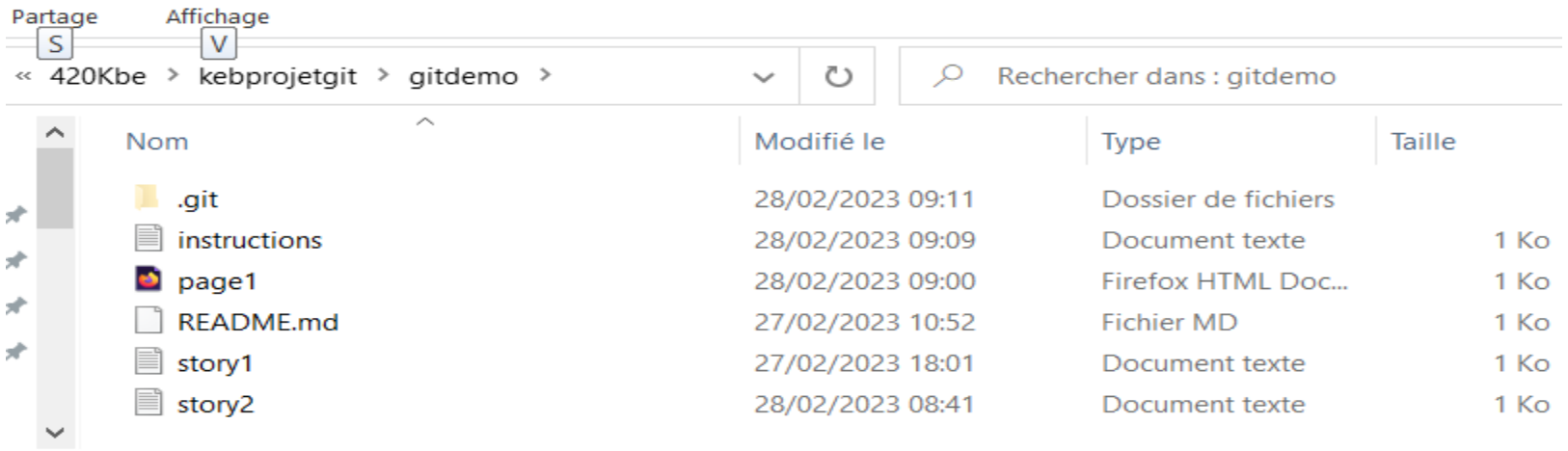




Ajouter votre commentaire, puis COMMIT

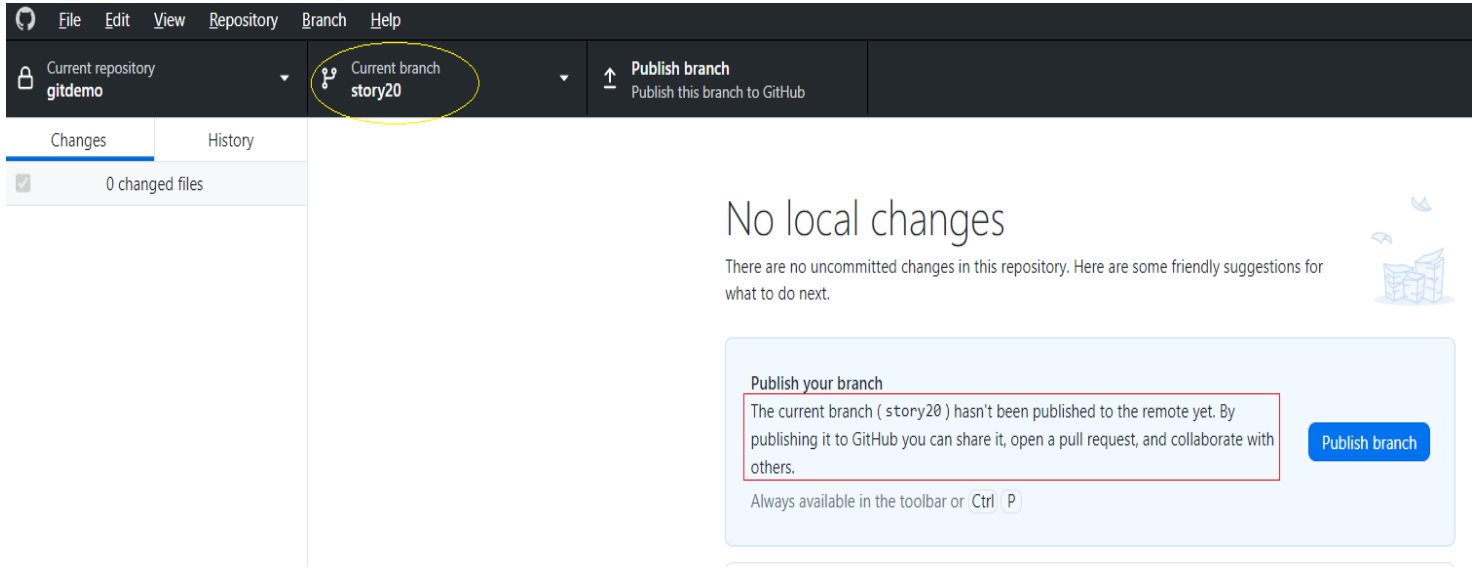
GitHub Desktop, application

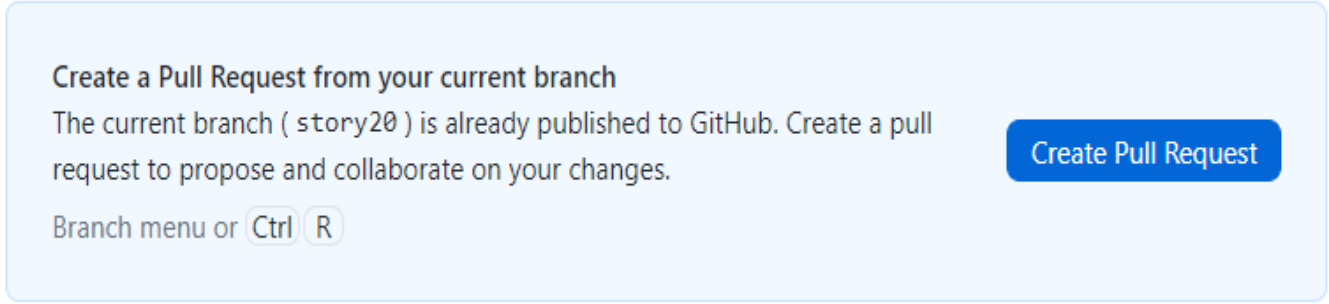
- Dans votre GitHub Desktop, assurez-vous d'être dans la bonne branche: (ici sotry20)
- Nous allons créer un 2eme fichiers dans cette branche.
- Dans GitHub Desktop, on fait le commit.
- Dans votre dossier local, tous vos fichiers sont là



Faire un Push sur GitHub

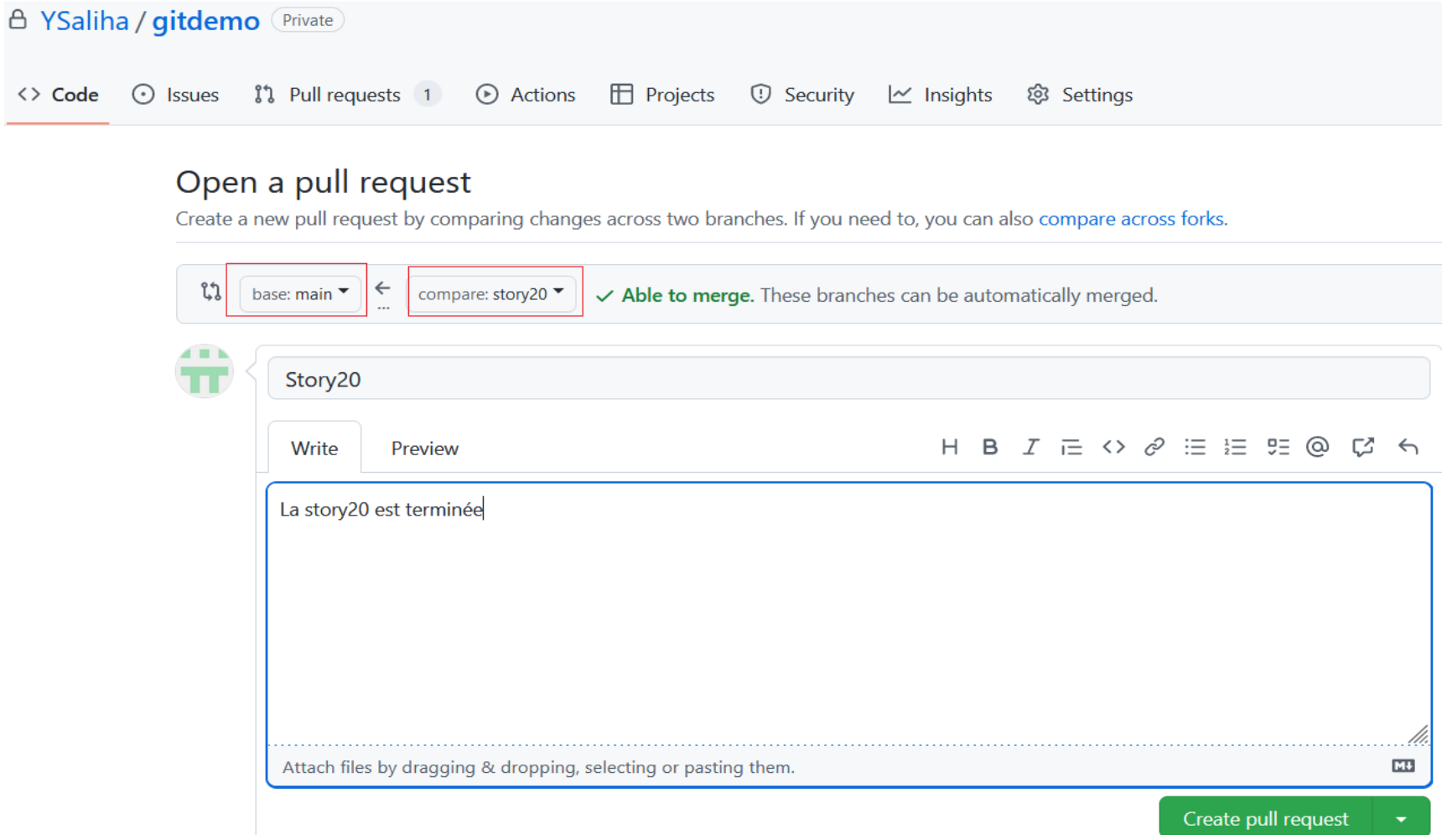
- Dans votre GitHub Desktop, assurez-vous d'être dans la bonne brache, Cliquer sur Publish branch
- Create Pull Request ce qui va permettre de vérifier et de résoudre les conflits
- Les pull requests vous permettent d'informer les autres collaborateurs des modifications que vous avez appliquées à une branche d'un repository sur GitHub, et que vous voulez fusionner avec le code principal





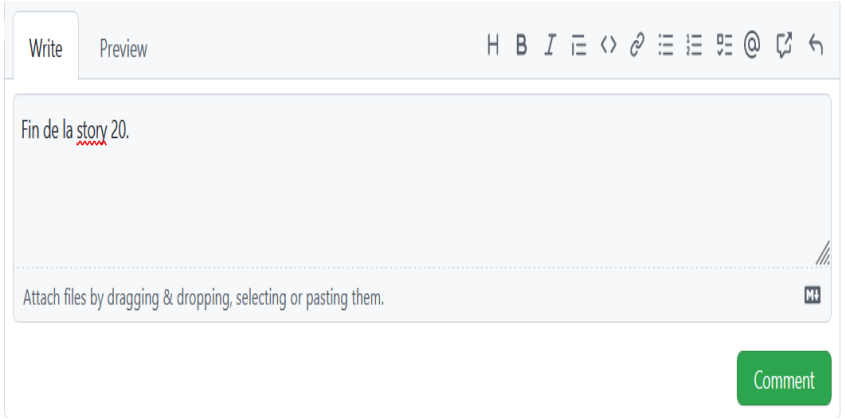
Faire un Push sur GitHub

- Laisser un commentaire utile pour vous et vos collaborateurs , puis Create pull request

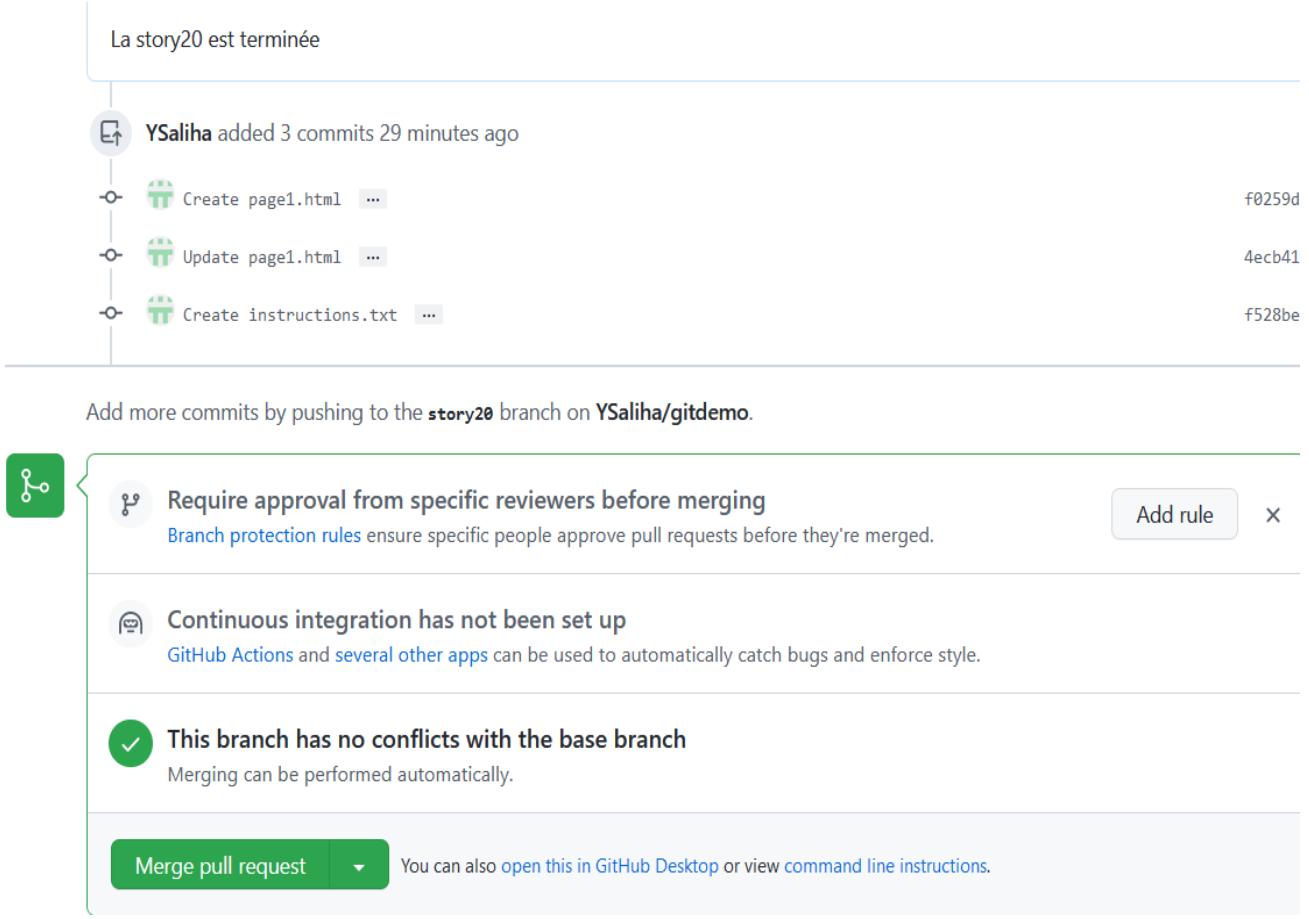


Faire un Push sur GitHub

- Pour finaliser, remarquez que la branche n’a aucun conflit.
- Laisser un commentaire

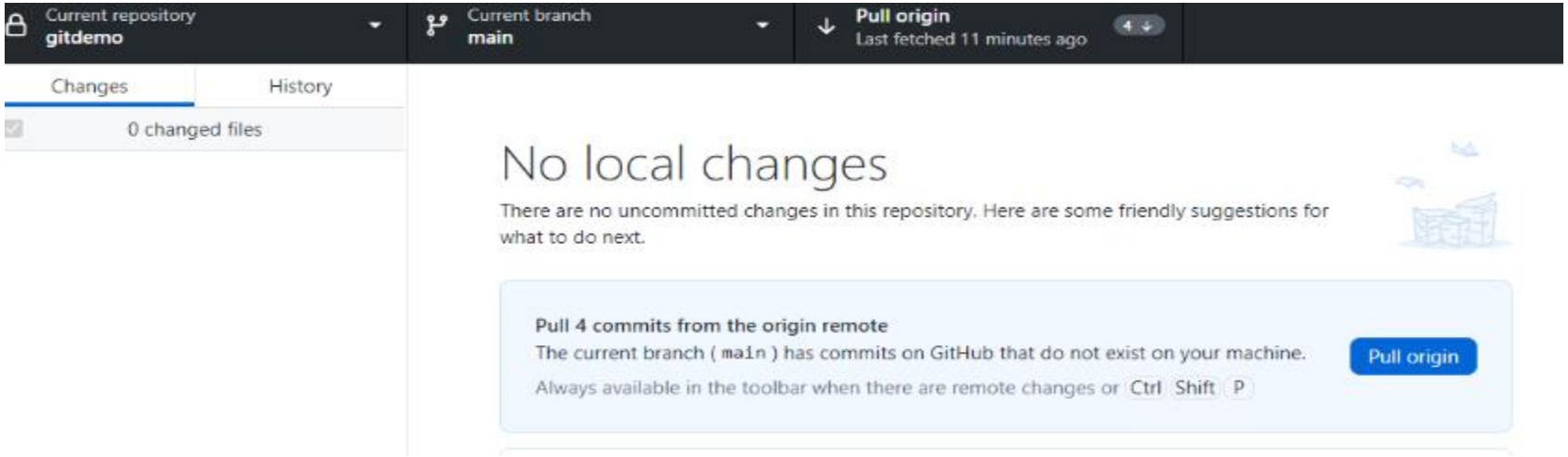


- Cliquer Merge pull request



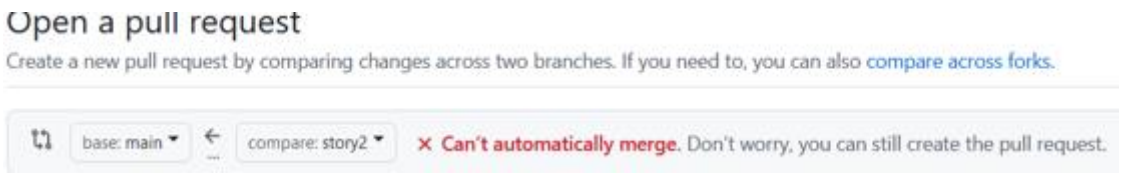
Faire un Pull origin

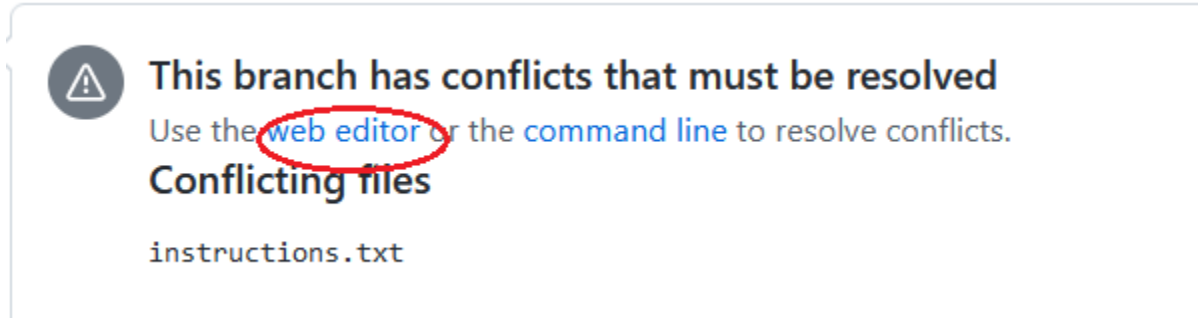
- Vous pouvez faire un Push sans merge
- **Vous ne faîtes un merge que si vous êtes certains que tout est OK.**
- Si vous êtes un collaborateur et que vous voulez avoir tous les changements faits sur le main dans GitHub, faite Pull Origin. Puis vérifier que dans votre dossier local, les changements sont présents



Gérer un conflit

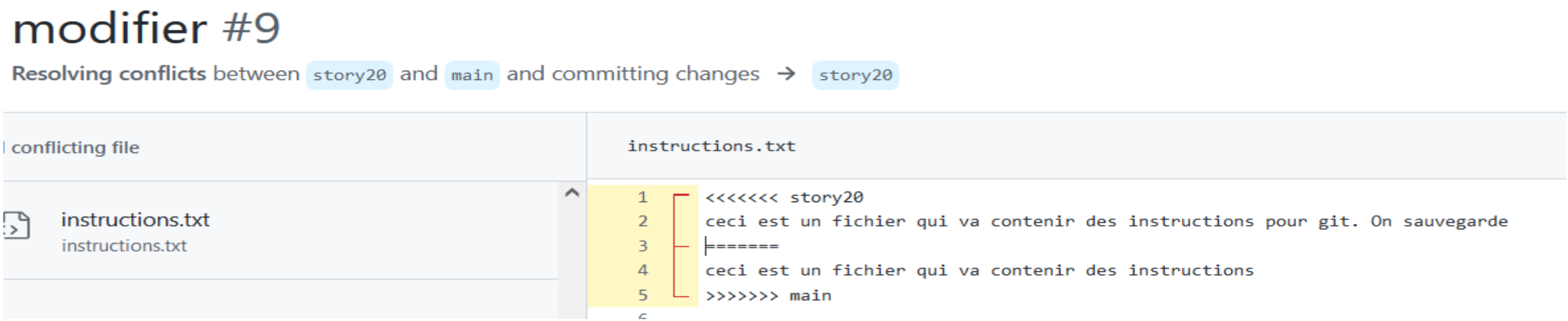
- Dans certains cas, il arrive que le merge ne se fasse pas car il y a un conflit.
- Pour le régler soit vous revenez dans votre code original et essayer de voir le problème. Soit vous choisissez: web editor pour voir le contenu du fichier





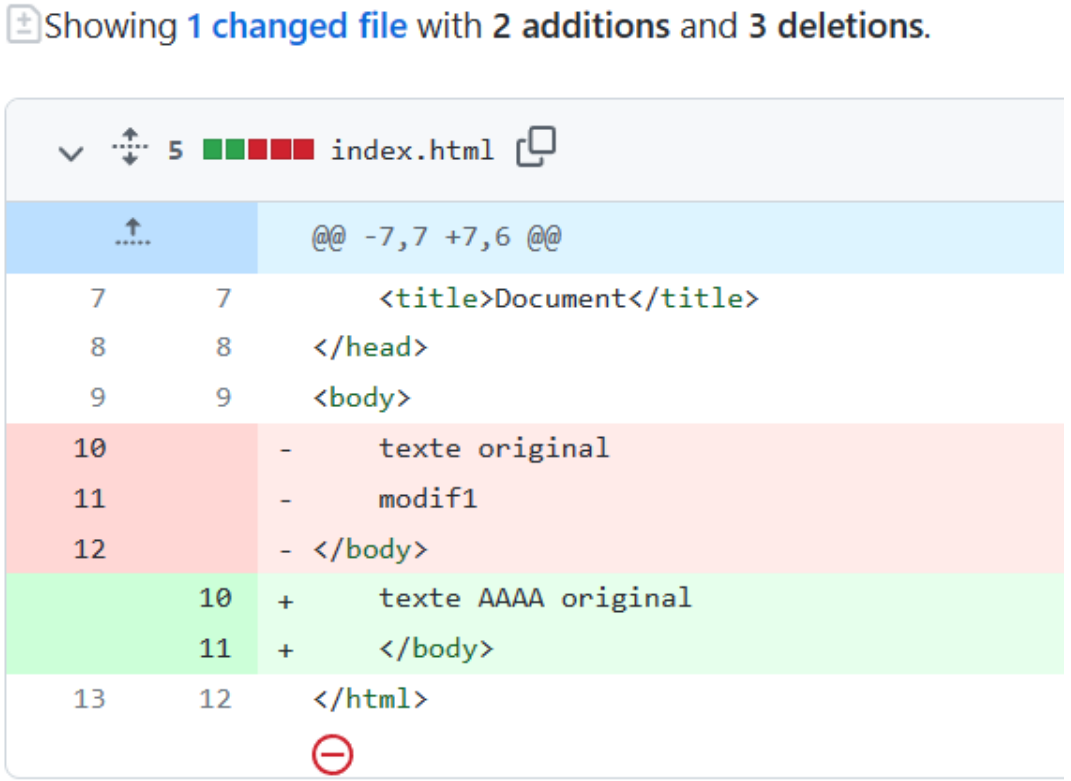
Gérer un conflit

- Le conflit ici est que le main et la branche story 20 ont modifié la même ligne différemment.
- On peut avoir un conflit si on essaie de modifier un fichier supprimé.
- On peut avoir un conflit si deux collaborateur donnent le même nom de fichiers à des fichiers différents.



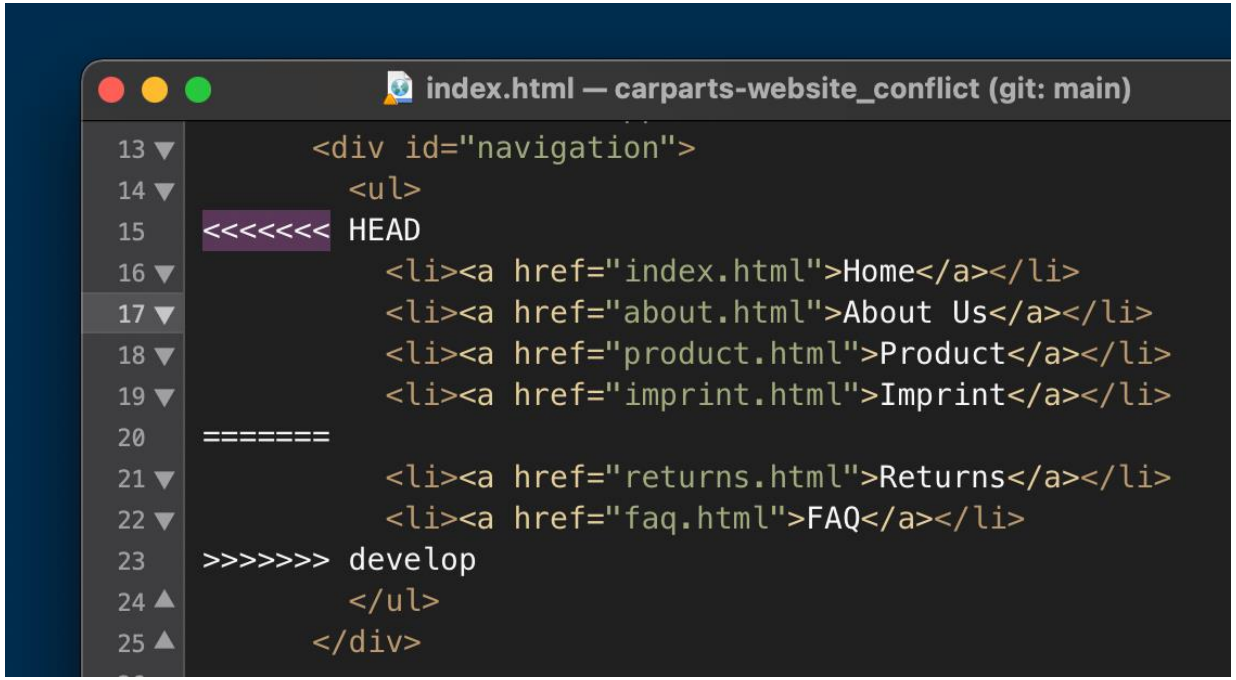
Gérer un conflit

- Quand un conflit est généré vous devez décider la version finale du code afin que la fusion puisse s'effectuer.
- Le cas typique est quand 2 personnes ont modifié la même ligne, comme dans l'exemple présenté ici.



Gérer un conflit

- Quand on édite un conflit on nous montre le contenu actuel de la destination (HEAD) et le contenu qui sera utilisé pour le changement (le nom de la branche).
- À la fin de la modification manuelle vous devez avoir modifié entre le <<<<<< et le >>>>>> ainsi que retirer ces symboles.

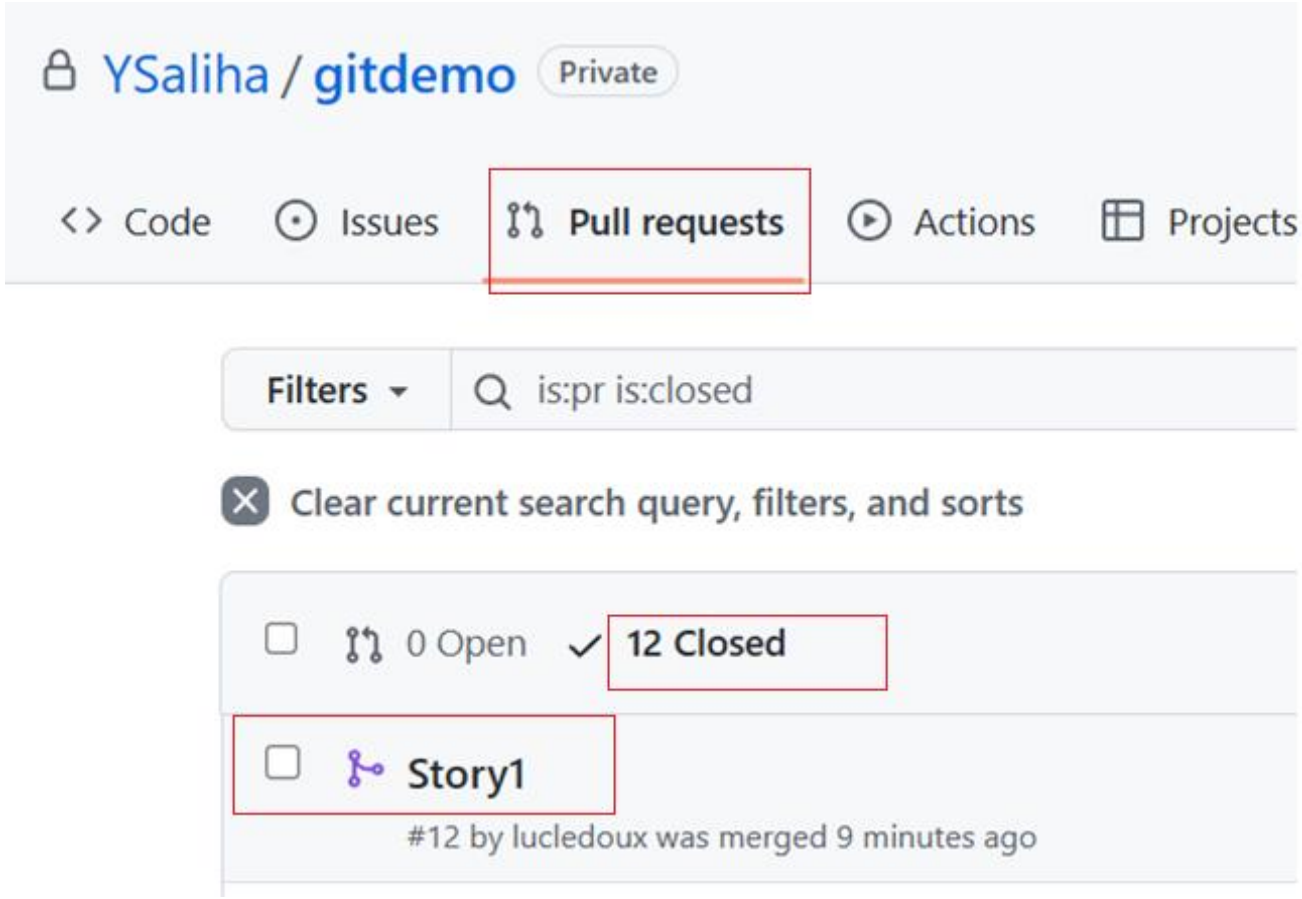


Revert, revenir en arrière

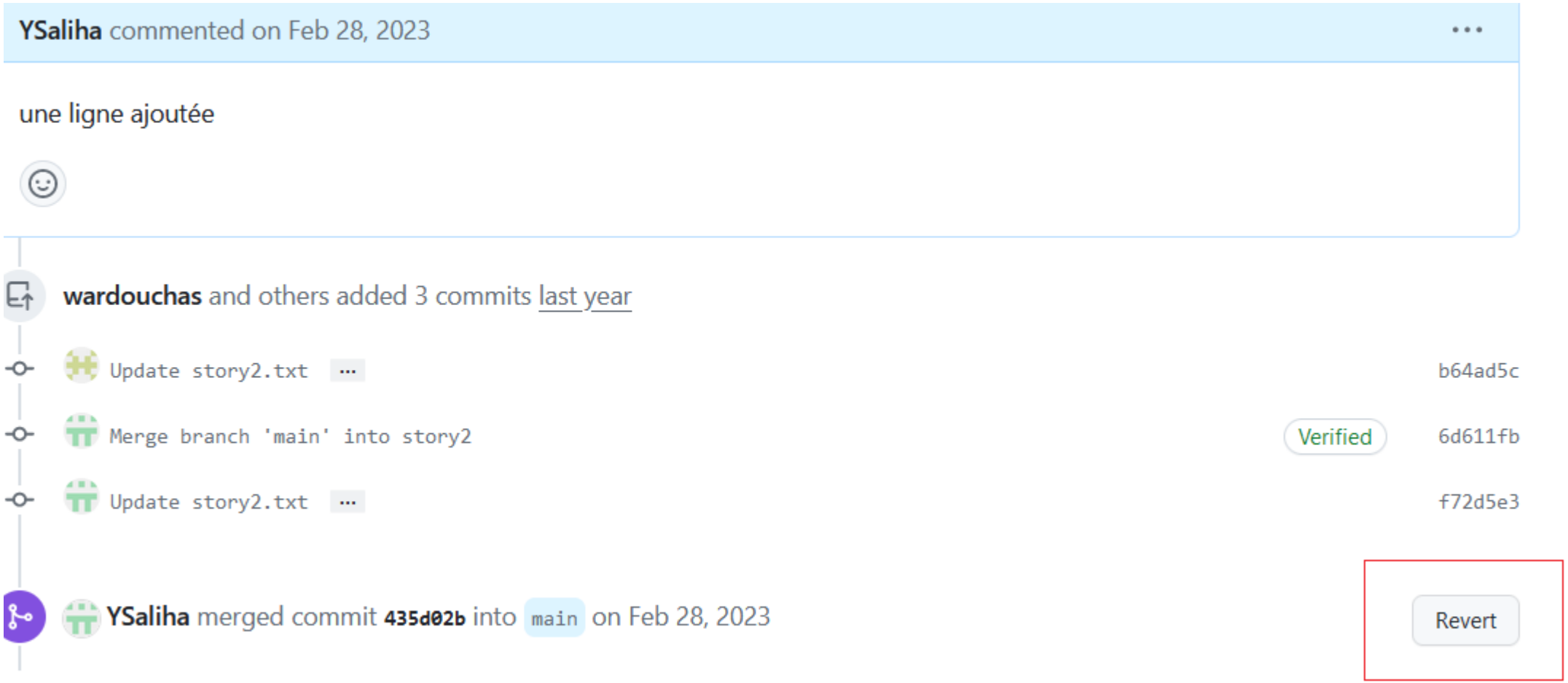
- Il est possible de revenir en arrière et d'**annuler une fusion de branche** (merge).
- Ceci pourrait être fait si on réalise qu'une erreur majeure a été commise au moment où on a fusionné la branche d'une story et ça cause le Main de cesser de fonctionner.
- Cette action doit être faite en cas de problème majeur, dans la mesure du possible faire une vérification complète avant de faire une fusion.
- On fait une fusion si on estime la story complétée, donc testée.
- Pour faire le Revert nous devons aller dans la section **Pull Requests-->Closed**

Revert, revenir en arrière

- Sélectionner la fusion à renverser.
- Appuyez sur le bouton **Revert**.
- Cette action va créer une nouvelle "Pull Request", appuyez sur le bouton **Create pull request**.
- Une vérification de conflits va s'effectuer, appuyez sur le bouton **Merge pull request** et **Confirm merge.**



Revert, revenir en arrière



Revert, revenir en arrière

- Une fois le **Revert** terminé vous verrez la création d'une nouvelle branche ayant comme source Main et sans les modifications de la fusion originale.
- Vous pouvez alors continuer votre travail sur la branche d'origine afin de régler les problématiques.
- Vous pouvez supprimer la nouvelle branche Revert si elle n'est pas utilisée.
- git **reset** va revenir à l'état précédent sans créer un nouveau commit, alors que git **revert** va créer un nouveau commit.
- git reset est accessible par des commandes git
- On peu faire un revert d’un revert

Conclusion

- Utiliser un outil de gestion de version a beaucoup d'avantages en particulier lorsque vous travaillez en équipes
- L'utilisation d'un outil de gestion des version et de collaboration peut être parfois complexe, mais nous vous recommandons fortement dans le cadre du cours de Projet Dirigé.
- Nous utiliserons GitHub pour voir la participation des membre de l'équipe à la réalisation de votre projet.

Sources:

- https://openclassrooms.com/fr/
- https://www.nobledesktop.com/learn/git/git-branches
- https://docs.github.com/fr/get-started/onboarding/getting-started-with-your-github-account

CONCLUSION

QUESTIONS ??