Travailler en équipe avec Git & GitHub

Sommaire

1. Préambule

Ce document résume la marche à suivre pour travailler en équipe sans casser le code des autres.

Configuration git

Avant toute chose, et surtout pour ceux qui ont changé de poste, il faut configurer git :

Ouvrir une console git bash puis exécuter les commandes suivantes :

```
git config --global user.name "VotreUserNameGitHub"
git config --global user.email "VotreEmailAssociéAVotreCompteGitHub"
git config --global http.proxy http://10.1.2.5:8080
```

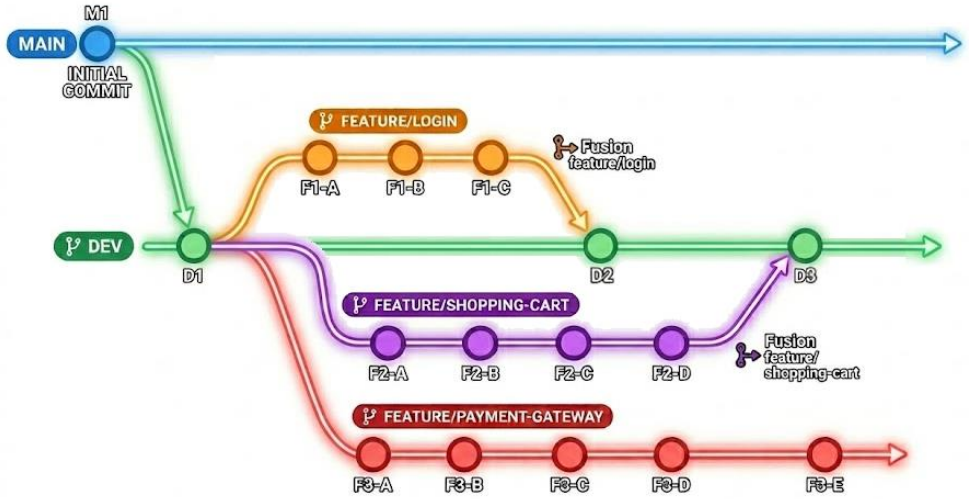
Extensions Visual Code

Dans Visual Code, installer les extensions suivantes :

Principe des branches

Lorsque l'on collabore à plusieurs sur un projet, il est indispensable que chacun travaille dans un espace qui lui est propre. Pour ce faire on travaille avec des **branches**.

Illustration :



La branche `main` correspond à la dernière version stable d'un projet.

La branche `dev` est utilisée pour livrer les fonctionnalités des collaborateurs puis effectuer des tests d'intégration (s'assurer que tout fonctionne correctement après ajout des modifications de chacun).

Une fois que tout est livrée, testé et fonctionnel, la branche `dev` est fusionnée dans la branche `main` pour donner une nouvelle version du projet.

2. Création du projet

Un membre de l'équipe crée le projet sur son poste :

- Le dossier du projet
- L'arborescence
- Les fichiers de base (navbar.php, footer.php, index.php) ainsi qu'une page template.php qui contient la structure de base de toutes les pages : entête du document (balise head avec les différents liens (css, font, ...) conteneur, h1, ..., liens vers les fichiers javascript si besoin, ainsi qu'un modèle de formulaire (pour l'emplacement, la disposition, les css, ...)
- Un fichier css (même s'il est vide au départ)

Vous pouvez vous répartir le travail mais il faut qu'à la fin tout soit installé sur le même poste

Une fois que c'est validé par l'équipe, le dépôt git doit être initialisé :

- Initialiser le dépôt local en nommant explicitement la branche principale

  ```
  git init --initial-branch=main
  ```

- Ajouter les fichiers à l'index (staging)

  ```
  git add .
  ```

- Valider les changements

  ```
  git commit -m "initialisation du projet"
  ```

Puis une nouvelle branche `dev` doit être créée :

```
git branch dev
```

On vérifie l'état du dépôt local :

```
git status
```

```
PS C:\Users\BEA\Desktop\testgit\moi> git status
On branch main
nothing to commit, working tree clean
```

On vérifie les branches :

```
git branch -a
```

```
PS C:\Users\BEA\Desktop\testgit\moi> git branch -a
  dev
* main
```

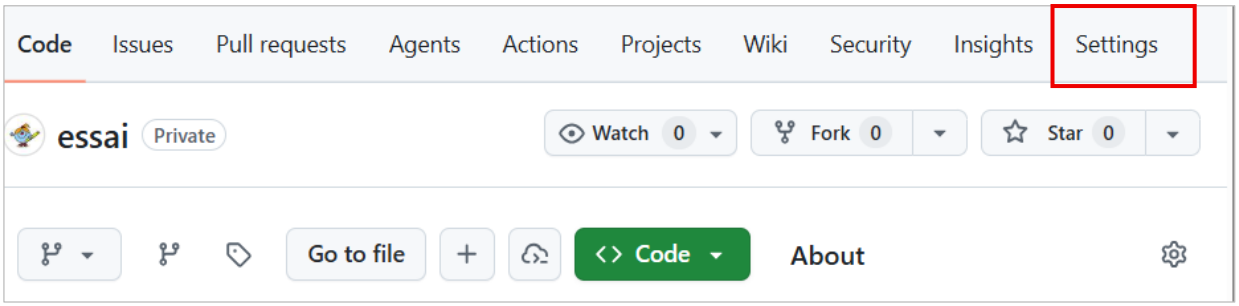
3. Mise en place sur Github

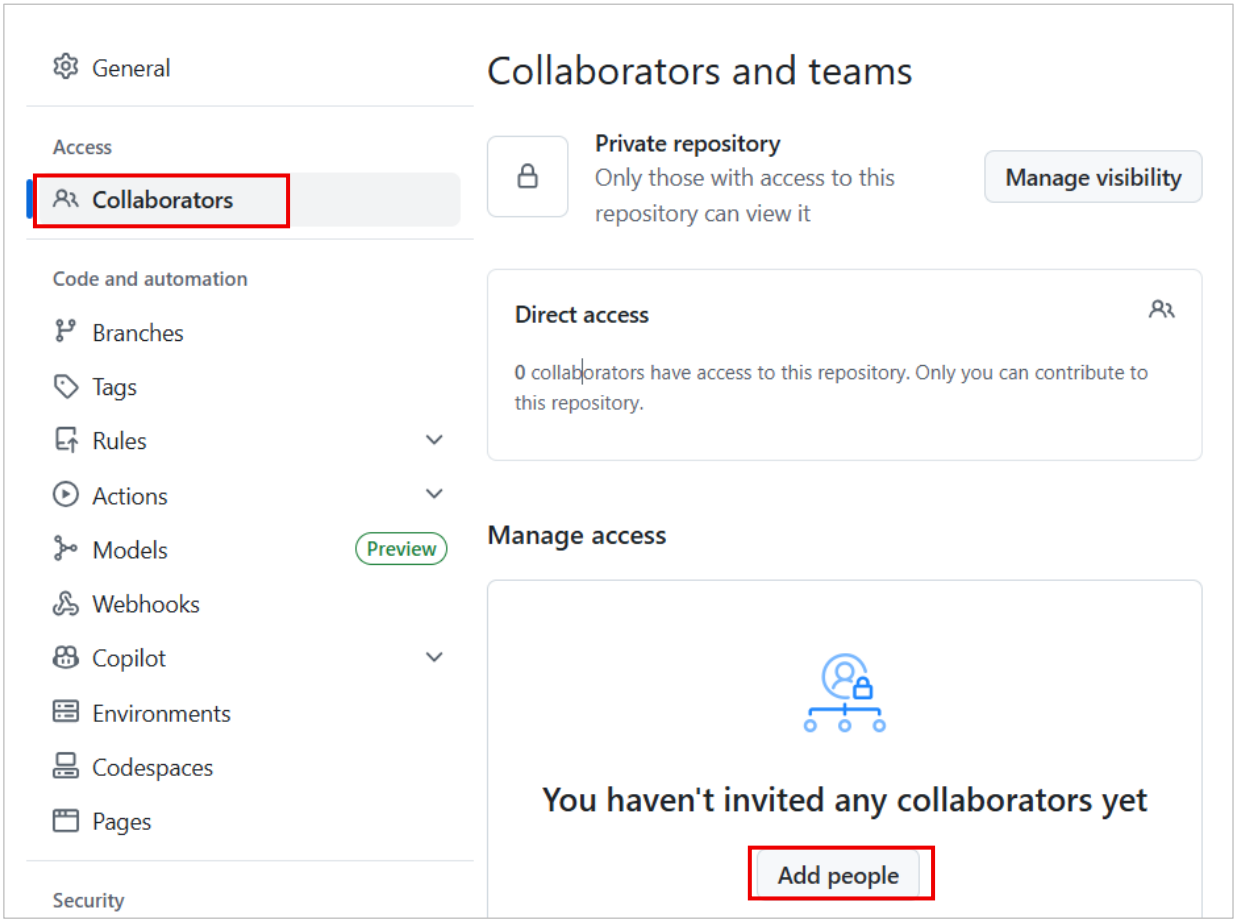
- Celui qui dispose du projet sur son poste crée un dépôt vide privé sur GitHub.
- Il associe le dépôt local au dépôt Github :

  `git remote add origin` *URL_du_dépôt_en_https*
- Puis il pousse les branches `main` et `dev` vers le dépôt Github :

  `git push -u origin main`

  `git push -u origin dev`
- Enfin il ajoute les membres de l'équipe + le prof : onglet « settings » :





4. Récupération du projet par les autres collaborateurs

- Cloner le projet :

```
git clone URL_du_dépôt_en_https
```

- Créer la branche `dev` locale par copie de la branche distante :

```
git switch dev
```

- Vérification :

```
git branch        # pour voir les branches locales
git branch -a     # pour voir toutes les branches locales et distantes
```

5. Créer sa branche pour développer sa partie

- Récupérer la dernière version de la branche dev de GitHub (par précaution) :

```
git switch dev
git pull
```

- Créer une nouvelle branche par copie de la branche dev ; le nom de la branche est le prénom du membre de l'équipe (tout en minuscules)

```
git checkout -b nomBranche dev
```

6. Au quotidien : Travailler sur sa branche

Chaque membre de l'équipe travaille sur sa propre branche. **Ne travaillez jamais directement sur main ou dev.**

En DEBUT de séance :

- Récupérer la dernière version de la branche dev de GitHub :

```
git switch dev
git pull
```

- Aller sur votre branche et faire un merge (fusion) de la branche dev pour récupérer les éventuelles modifications des autres membres de l'équipe

```
git switch nomBranche
git merge dev
```

- Résoudre les conflits si besoin (Cf. § 9. Résolution des conflits)

En COURS de séance, régulièrement, et en FIN de séance :

Enregistrer son travail :

```
git add .
git commit -m "Description claire du changement"
```

Envoyer sa branche sur GitHub :

```
git push origin nomBranche
```

Consigne : faire des commits fréquents avec des messages clairs (Cf. § 12. Commit et messages de commit)

7. Intégrer son travail : créer une Pull Request

Une fois la fonctionnalité terminée, testée et opérationnelle en local :

- Récupérer la dernière version de la branche dev de GitHub :

  ```
  git switch dev
  git pull
  ```

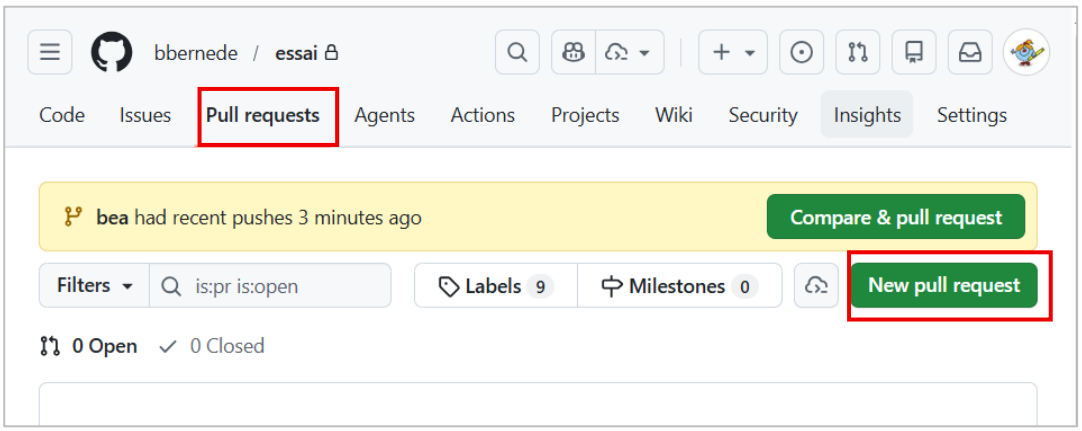
- Aller sur votre branche et faire un merge de la branche `dev` pour récupérer les éventuelles modifications des autres membres de l'équipe

  ```
  git switch nomBranche
  git merge dev
  ```

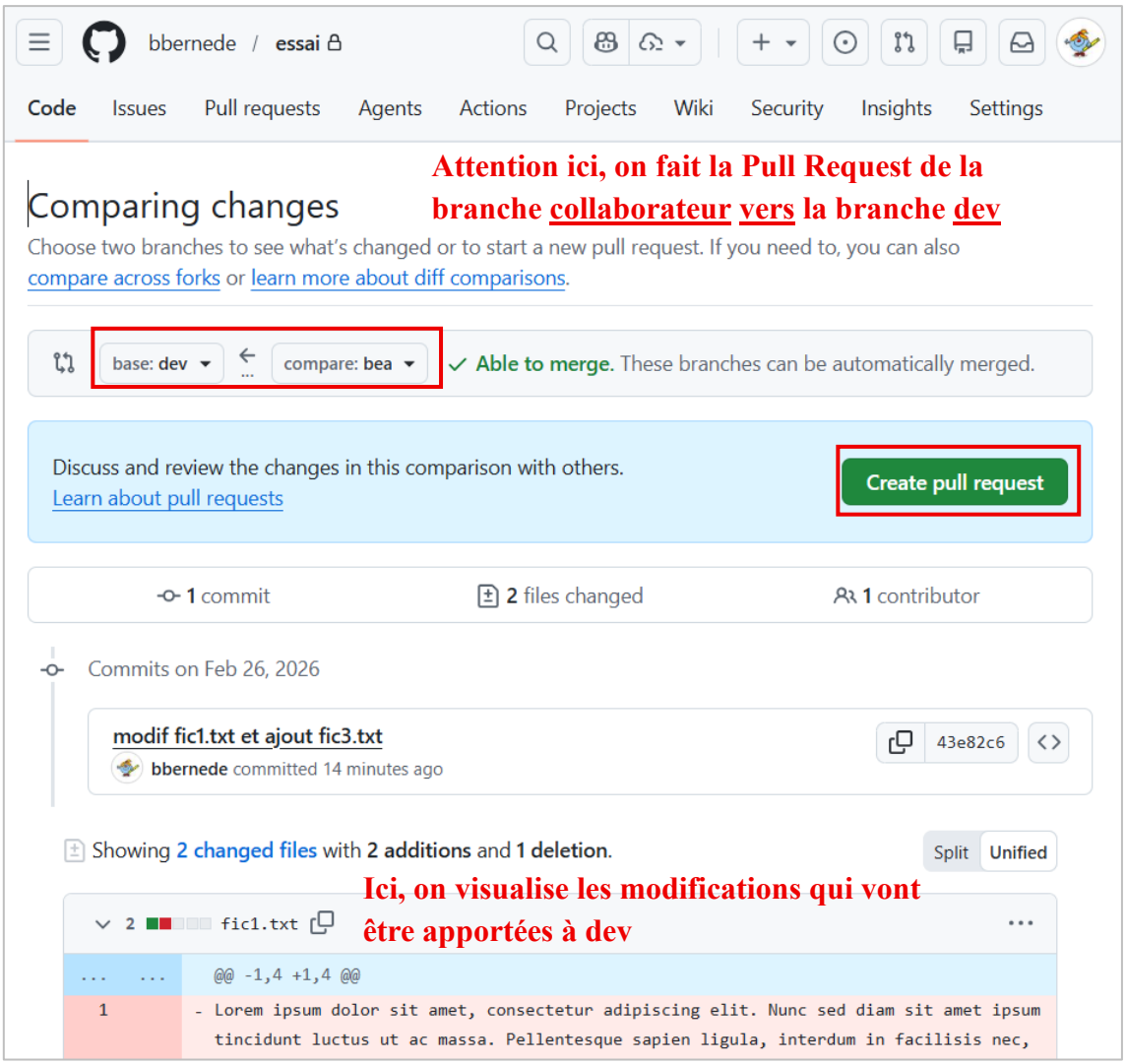
- Résoudre les conflits si besoin (Cf. § 9. Résolution des conflits)
- Envoyer votre branche sur GitHub :

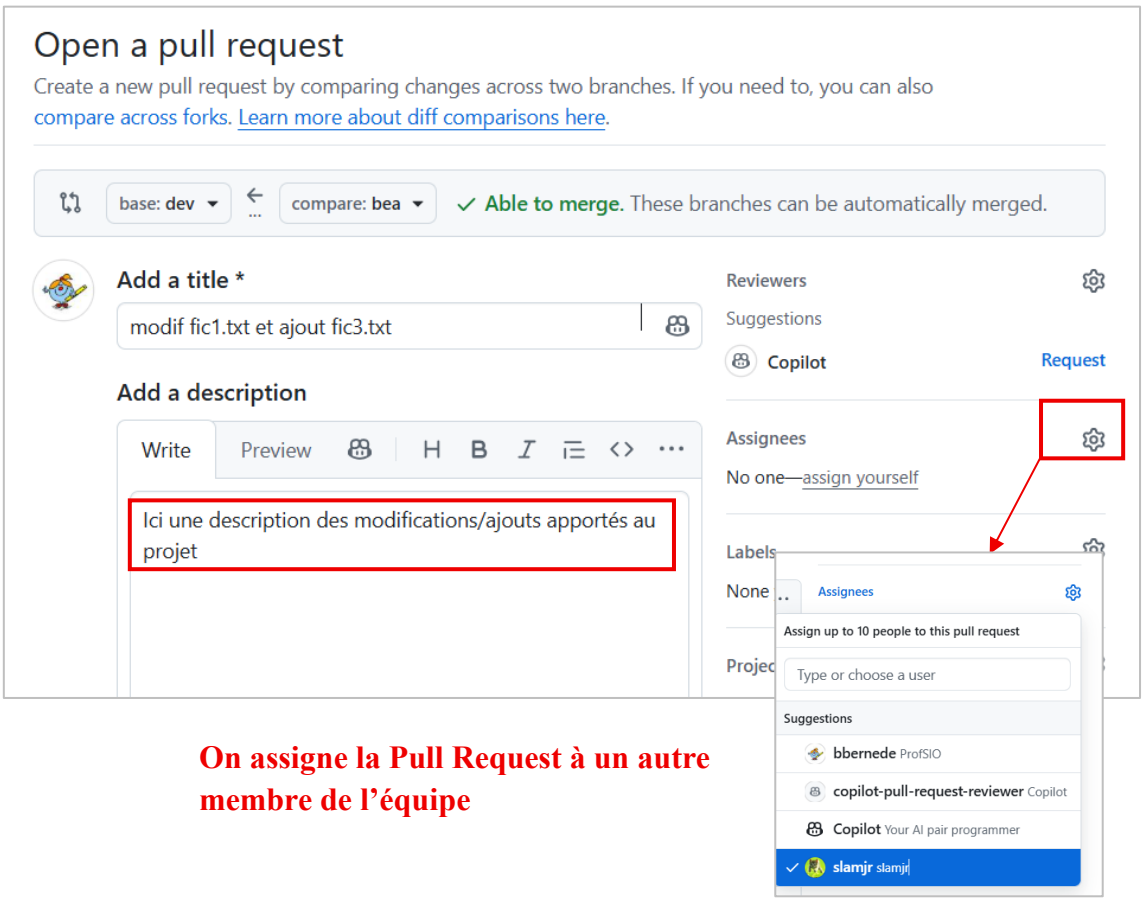
  ```
  git push origin nomBranche
  ```

- Allez sur GitHub.
  - Depuis l'onglet Pull Request, demandez à créer une nouvelle **Pull Request (PR)** :

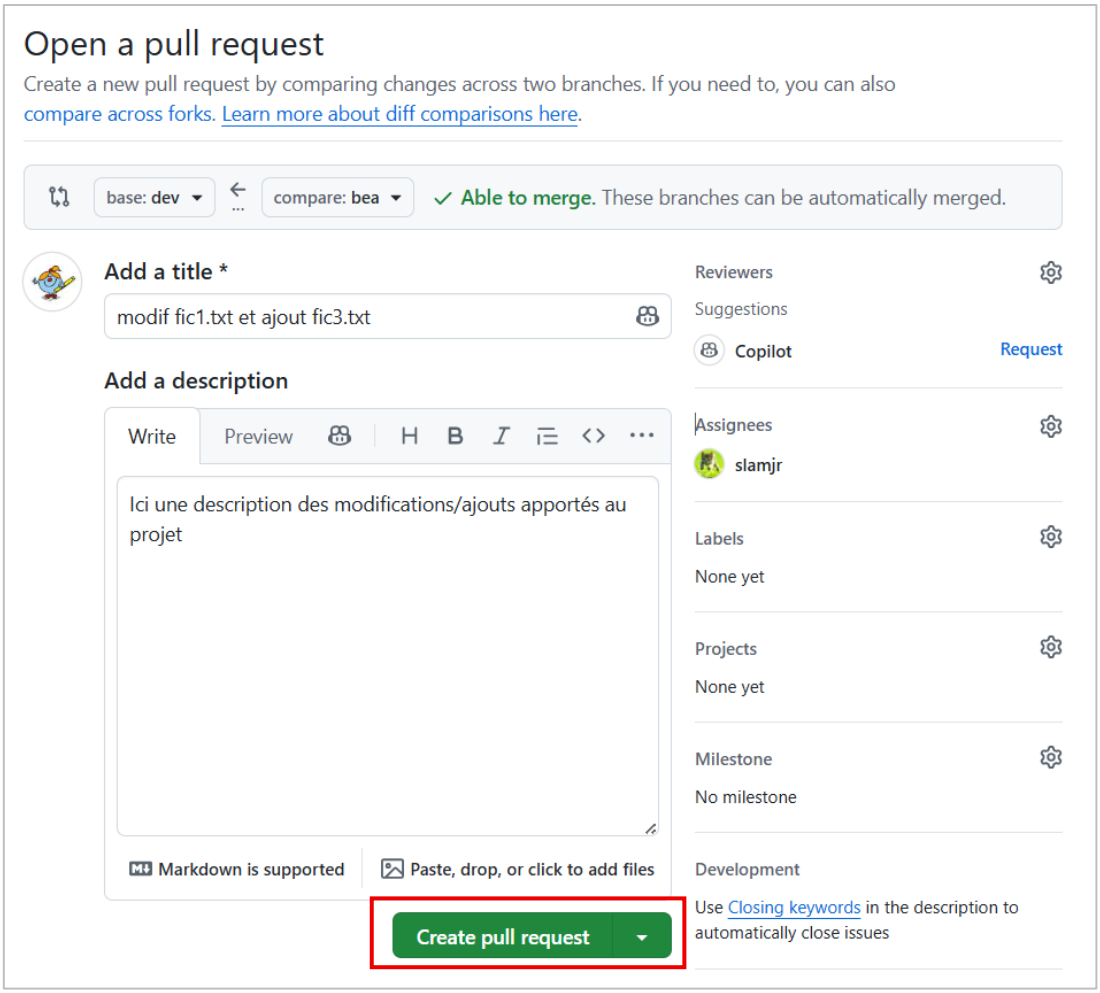


- Créer la **Pull Request (PR)** de la branche collaborateur vers dev :



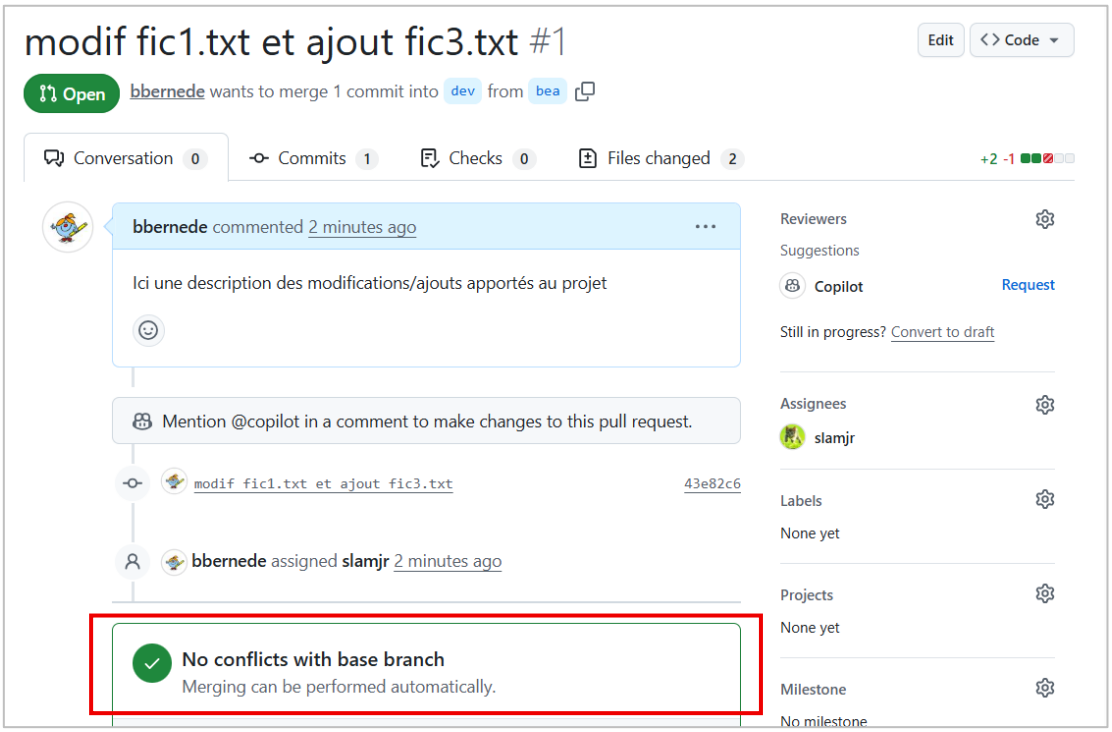


On assigne la Pull Request à un autre membre de l'équipe



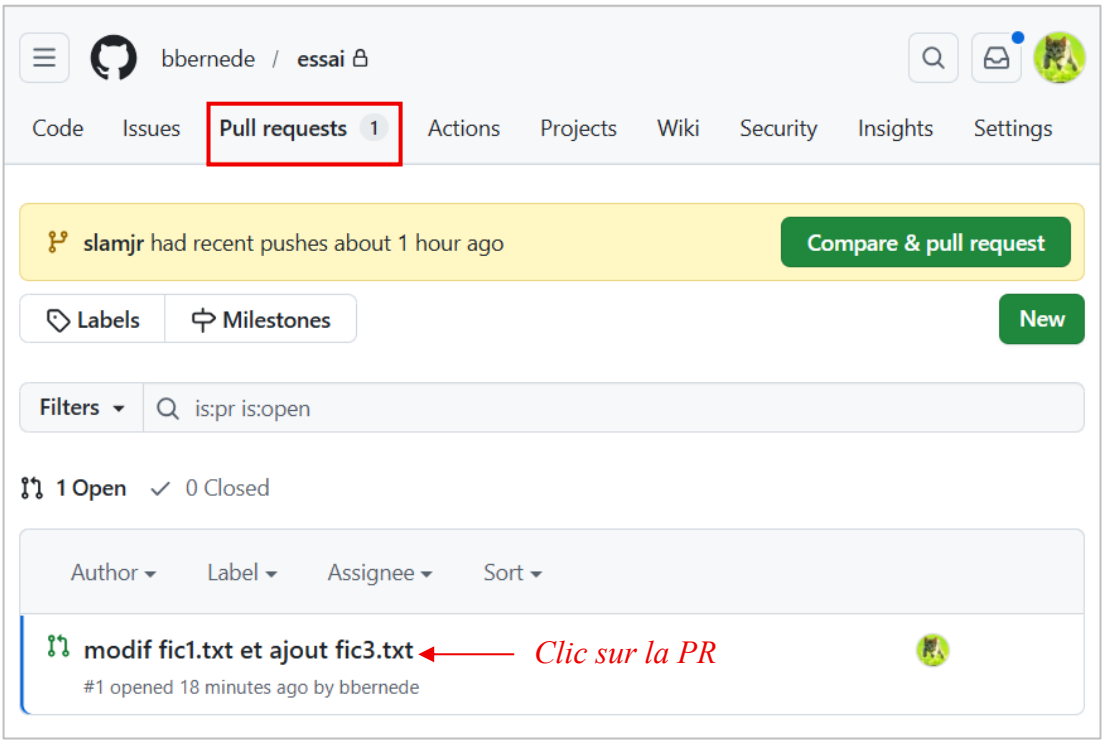
- La Pull Request est créée :

  Normalement **aucun conflit ne doit apparaître** à ce stade, si vous avez bien suivi le processus. Dans le cas contraire, reprendre le développement afin d'adapter le code pour résoudre les problèmes soulevés

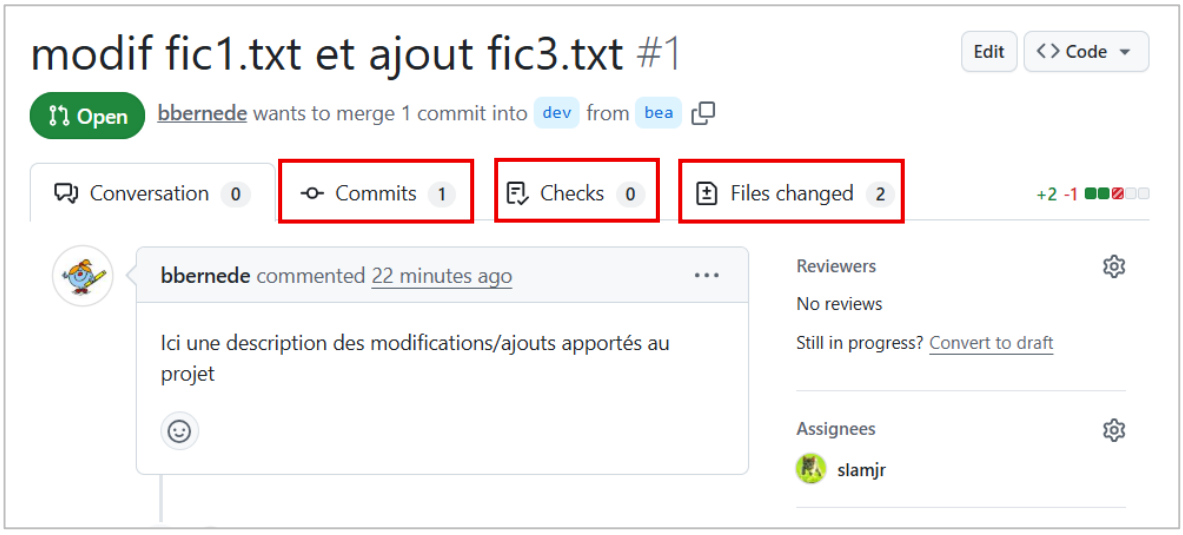


8. Prise en charge de la Pull Request pour validation

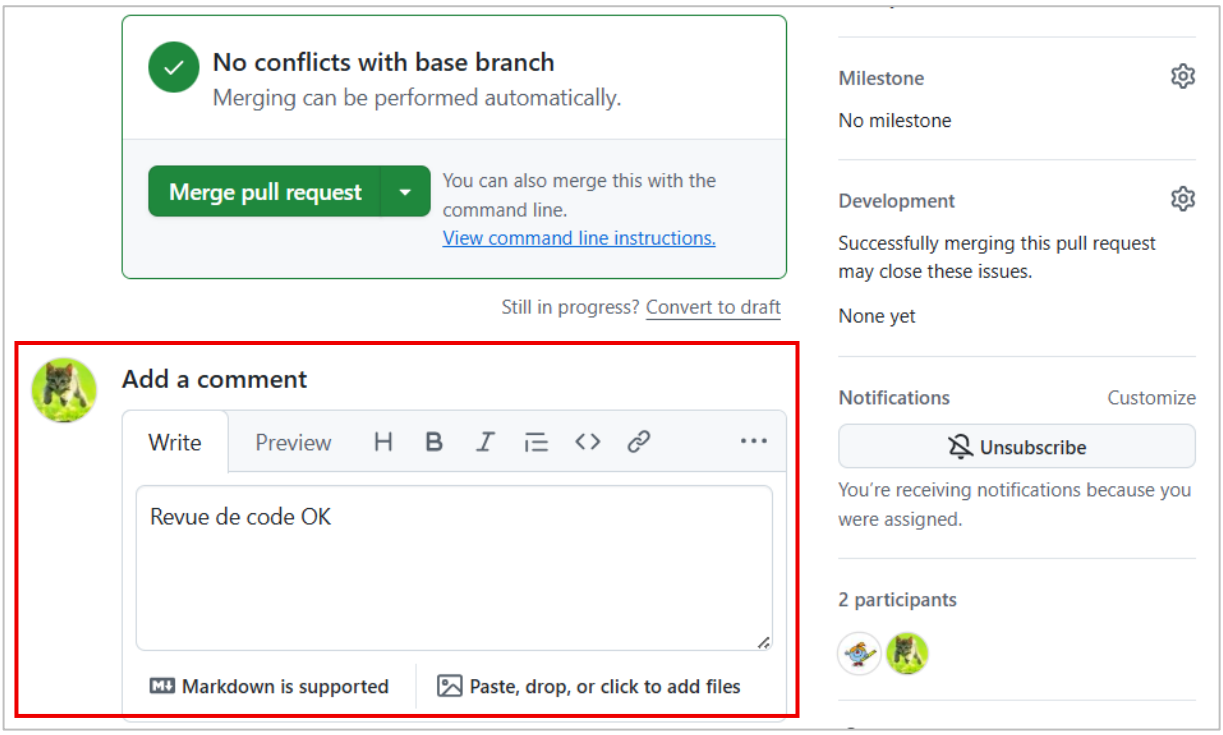
- Le membre de l'équipe qui s'est vu assigner la Pull Request, l'ouvre depuis l'interface GitHub :



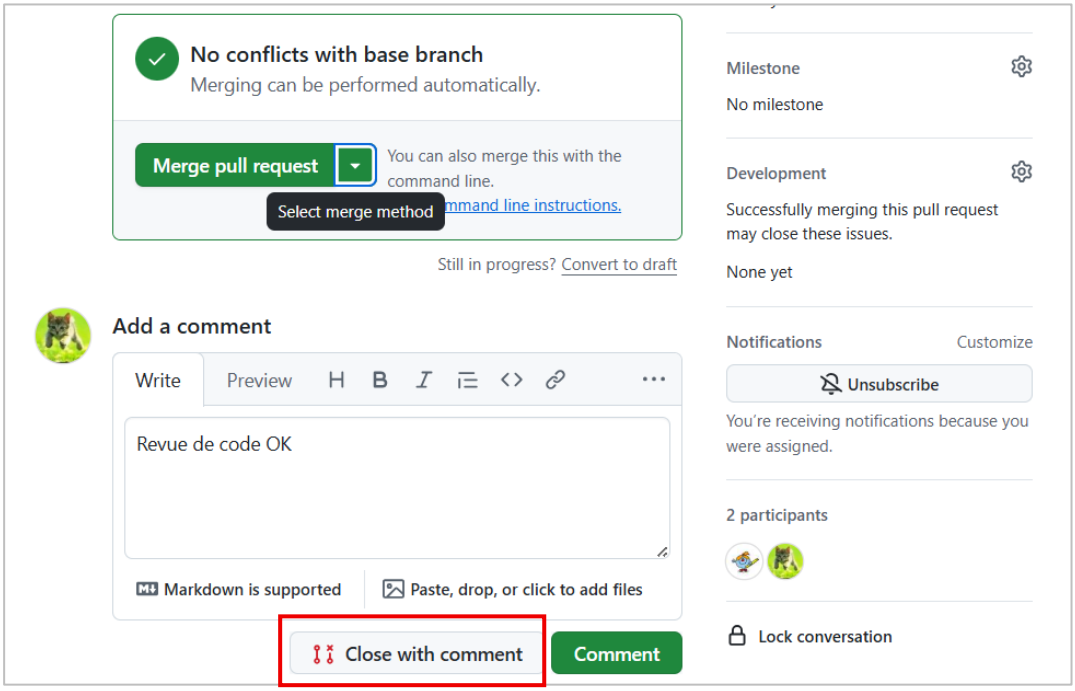
- Il réalise **une revue de code** en étudiant les informations relatives à la Pull Request : Commits, Checks, Files changed ...



- A ce stade des commentaires peuvent être ajoutés :

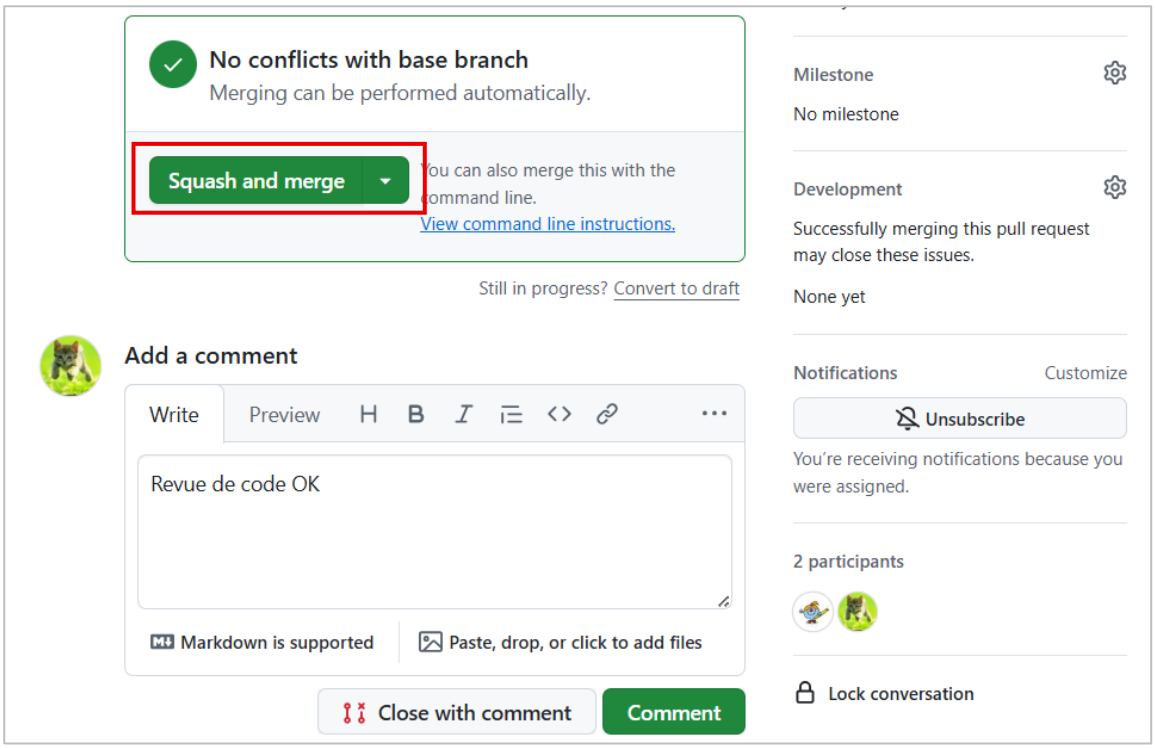


- Enfin, il approuve ou refuse la demande de fusion (merge) :
  - **Pour REFUSER** la demande de merge, cliquer sur **Close** pull request.

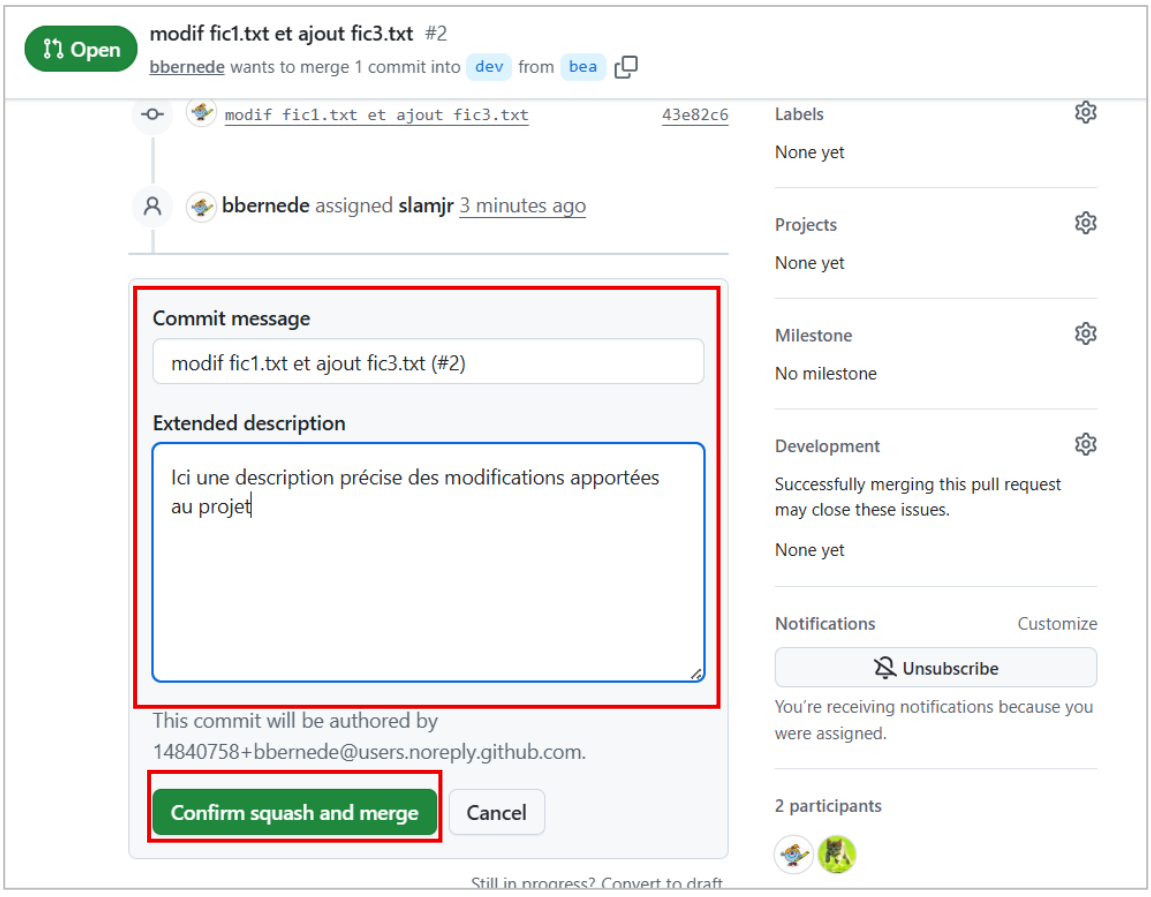


- **Pour APPROUVER** la demande de merge :

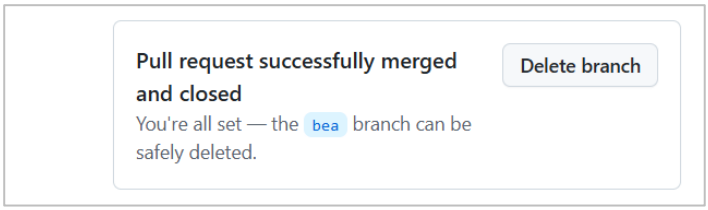
  *Le choix « **Squash and merge** » permet de réduire le nombre de commit transférés dans la branche dev.*



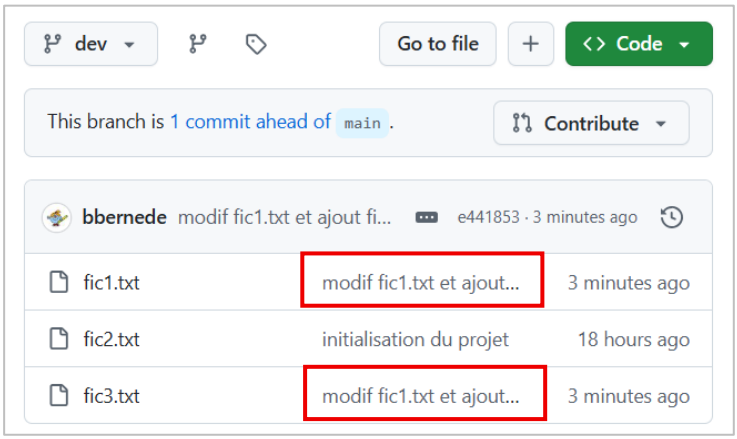
Il faut alors confirmer après avoir indiqué une description précise des modifications apportées au projet :



A ce stade, la branche du collaborateur peut être supprimée selon le choix de conserver ou non l'ensemble des branches :



- Si la Pull Request a été approuvée, la branche dev de GitHub contient les nouvelles évolutions :



- **<u>Prévenir les autres membres de l'équipe</u>** que la branche dev a été mise à jour afin qu'ils récupèrent la dernière version en local et la fusionne dans leur branche :

```
git switch dev
git pull
git switch nomBrancheCollaborateur
git merge dev
```

9. Résolution des conflits

Note : Si Git signale un conflit, ouvrez les fichiers concernés, choisissez le code à conserver, sauvegardez, puis faites un `git add` et `git commit`.

Exemple sur une branche locale slamjr :

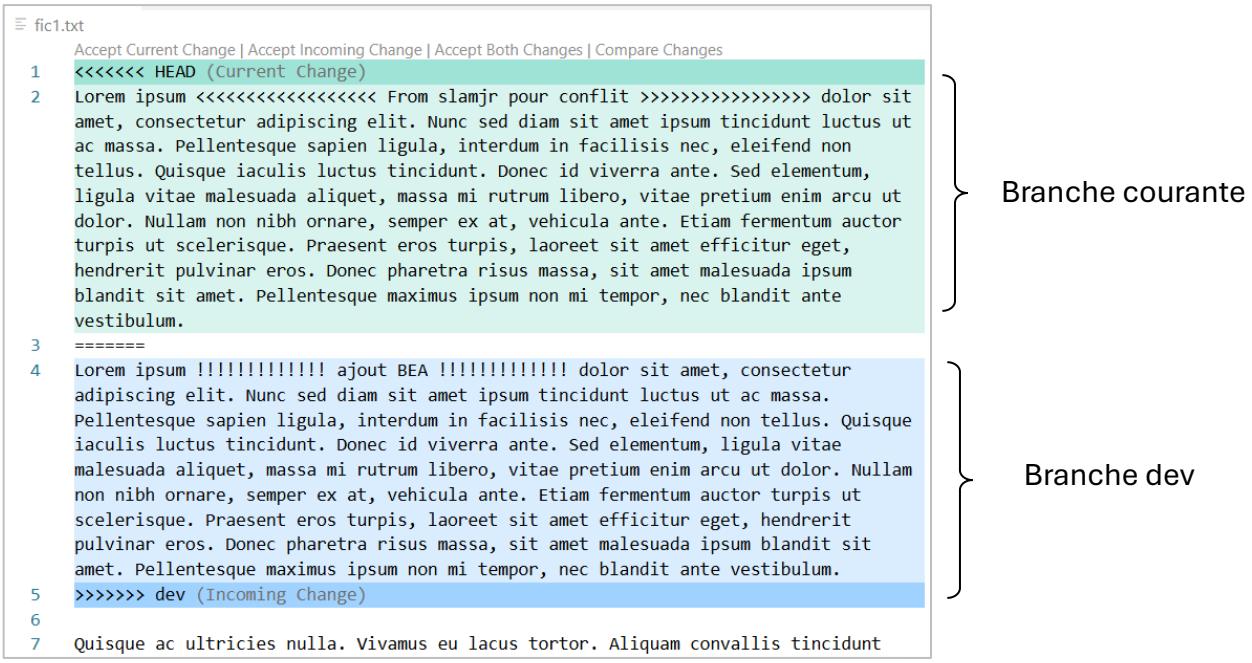
- Lorsque que vous tentez de fusionner la branche dev dans votre branche personnelle, git vous signale un conflit :

```
PS C:\Users\Prof\Desktop\testgit\essai> git merge dev
Auto-merging fic1.txt
CONFLICT (content): Merge conflict in fic1.txt
Automatic merge failed; fix conflicts and then commit the result.
PS C:\Users\Prof\Desktop\testgit\essai>
```

Pour visualiser les conflits :

```
PS C:\Users\Prof\Desktop\testgit\essai> git diff slamjr dev
diff --git a/fic1.txt b/fic1.txt
index d6751e4..d94d549 100644
diff --git a/fic1.txt b/fic1.txt
index d6751e4..d94d549 100644
--- a/fic1.txt
+++ b/fic1.txt
@@ -1,4 +1,4 @@
-Lorem ipsum <<<<<<<<<<<<<<<<<< From slamjr pour conflit >>>>>>>>>>>>>>>> dolo
ut ac massa. Pellentesque sapien ligula, interdum in facilisis nec, eleifend no
a vitae malesuada aliquet, massa mi rutrum libero, vitae pretium enim arcu ut d
s ut scelerisque. Praesent eros turpis, laoreet sit amet efficitur eget, hendre
t. Pellentesque maximus ipsum non mi tempor, nec blandit ante vestibulum.
+Lorem ipsum !!!!!!!!!!!!! ajout BEA !!!!!!!!!!!!! dolor sit amet, consectetur
e sapien ligula, interdum in facilisis nec, eleifend non tellus. Quisque iaculi
t, massa mi rutrum libero, vitae pretium enim arcu ut dolor. Nullam non nibh or
```

- Traiter les conflits manuellement en éditant les fichiers indiqués par git. Il faut adapter le code et retirer les marques git. Vous pouvez également utiliser les liens au-dessus du code :



Ici, pour l'exemple, j'ai repris les modifications provenant des 2 fichiers :

fic1.txt

```
1   Lorem ipsum !!!!!!!!!!!!! ajout BEA !!!!!!!!!!!!! <<<<<<<<<<<<<<<<<< From slamjr
    pour conflit >>>>>>>>>>>>>>>>>> dolor sit amet, consectetur adipiscing elit. Nunc
    sed diam sit amet ipsum tincidunt luctus ut ac massa. Pellentesque sapien ligula,
    interdum in facilisis nec, eleifend non tellus. Quisque iaculis luctus tincidunt.
    Donec id viverra ante. Sed elementum, ligula vitae malesuada aliquet, massa mi
    rutrum libero, vitae pretium enim arcu ut dolor. Nullam non nibh ornare, semper ex
    at, vehicula ante. Etiam fermentum auctor turpis ut scelerisque. Praesent eros
    turpis, laoreet sit amet efficitur eget, hendrerit pulvinar eros. Donec pharetra
    risus massa, sit amet malesuada ipsum blandit sit amet. Pellentesque maximus ipsum
    non mi tempor, nec blandit ante vestibulum.
```

- Enregistrer
- Vérifier que le code est fonctionnel en réalisant TOUS LES TESTS NECESSAIRES
- Vérifier l'état de la branche :

```
PS C:\Users\Prof\Desktop\testgit\essai> git status
On branch slamjr
Your branch is up to date with 'origin/slamjr'.

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Changes to be committed:
        new file:   fic3.txt

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   fic1.txt
```

- On ajoute le(s) fichiers à l'index :

```
git add fic1.txt
```

```
PS C:\Users\Prof\Desktop\testgit\essai> git status
On branch slamjr
Your branch is up to date with 'origin/slamjr'.

All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

Changes to be committed:
        modified:   fic1.txt
        new file:   fic3.txt
```

- Il ne reste plus qu'à valider la fusion :

```
git commit -m "fusion branche slamjr dans dev"
```

- Puis à pousser la branche locale vers github :

```
git push
```

10. Éviter les conflits

Pour éviter au maximum les conflits, il est crucial de récupérer ce que les autres ont fusionné dans `dev` avant de continuer à coder sur votre branche, vous devez récupérer ce que les autres ont fusionné dans `dev`.

Action	**Commande**
1. Aller sur dev	`git switch dev`
2. Récupérer la version distante de la branche dev	`git pull origin dev`
3. Aller sur votre branche	`git switch` *`nomBranche`*
4. Fusionner dev dans votre branche	`git merge dev`

11. Fin de projet

Quand le projet est terminé et que la branche `dev` est stable, on fusionne `dev` dans `main` pour la livraison finale à l'aide d'une Pull Request sur GitHub.

💡 Les 3 règles d'or

1. **Faites des commit souvent :** De petits changements sont plus faciles à fusionner.
2. **Communiquez :** Prévenez vos collègues quand vous allez fusionner une grosse partie.
3. **Pull avant Push et Merge :** Assurez-vous d'avoir la version la plus récente de `dev` avant de proposer vos modifications.

12. Commit et messages de commit

Un commit doit idéalement représenter **une seule chose**. Si on modifie le menu, le pied de page et la base de données en même temps, le commit sera impossible à suivre en cas d'erreur.

Un bon message de commit doit être court et explicite. La structure recommandée est la suivante :

```
type: description courte
```

Les types à utiliser :

- **feature:** Une nouvelle fonctionnalité
- **fix:** Une correction de bug
- **docs:** Modification de la documentation
- **style:** Changement de design ou formatage
- **refactor:** Amélioration du code sans changer le fonctionnement

Exemples

```
git commit -m "feature: ajout de la barre de recherche"
git commit -m "feature: ajout du formulaire de contact"
git commit -m "fix: correction du lien vers la page profil"
git commit -m "fix: alignement du bouton menu"
git commit -m "style: passage des titres en bleu"
git commit -m "style: modification de la police"
git commit -m "docs: ajout des consignes d'installation"
git commit -m "docs: modification du fichier README"
git commit -m "refactor: nettoyage des fonctions JS"
```

À bannir

`git commit -m "update"` (Trop vague)

`git commit -m "eufhzieu"` (Inutile)

`git commit -m "ça marche enfin"` (Pas professionnel)

`git commit -m "fix bug"` (Quel bug ?)