

Travailler en équipe avec Git & GitHub

Sommaire

1. Préambule	2
2. Création du projet	3
3. Mise en place sur Github	4
4. Récupération du projet par les autres collaborateurs	5
5. Créer sa branche pour développer sa partie	5
6. Au quotidien : Travailler sur sa branche	6
7. Intégrer son travail : créer une Pull Request	7
8. Prise en charge de la Pull Request pour validation	10
9. Résolution des conflits	14
10. Éviter les conflits	16
11. Fin de projet	16
12. Commit et messages de commit	17

1. Préambule

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

Configuration git

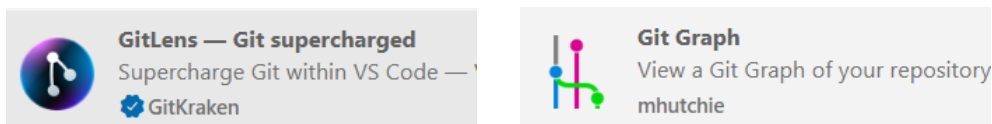
Avant toute chose, et surtout pour ceux qui ont changé de poste, il faut configurer git :

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
```

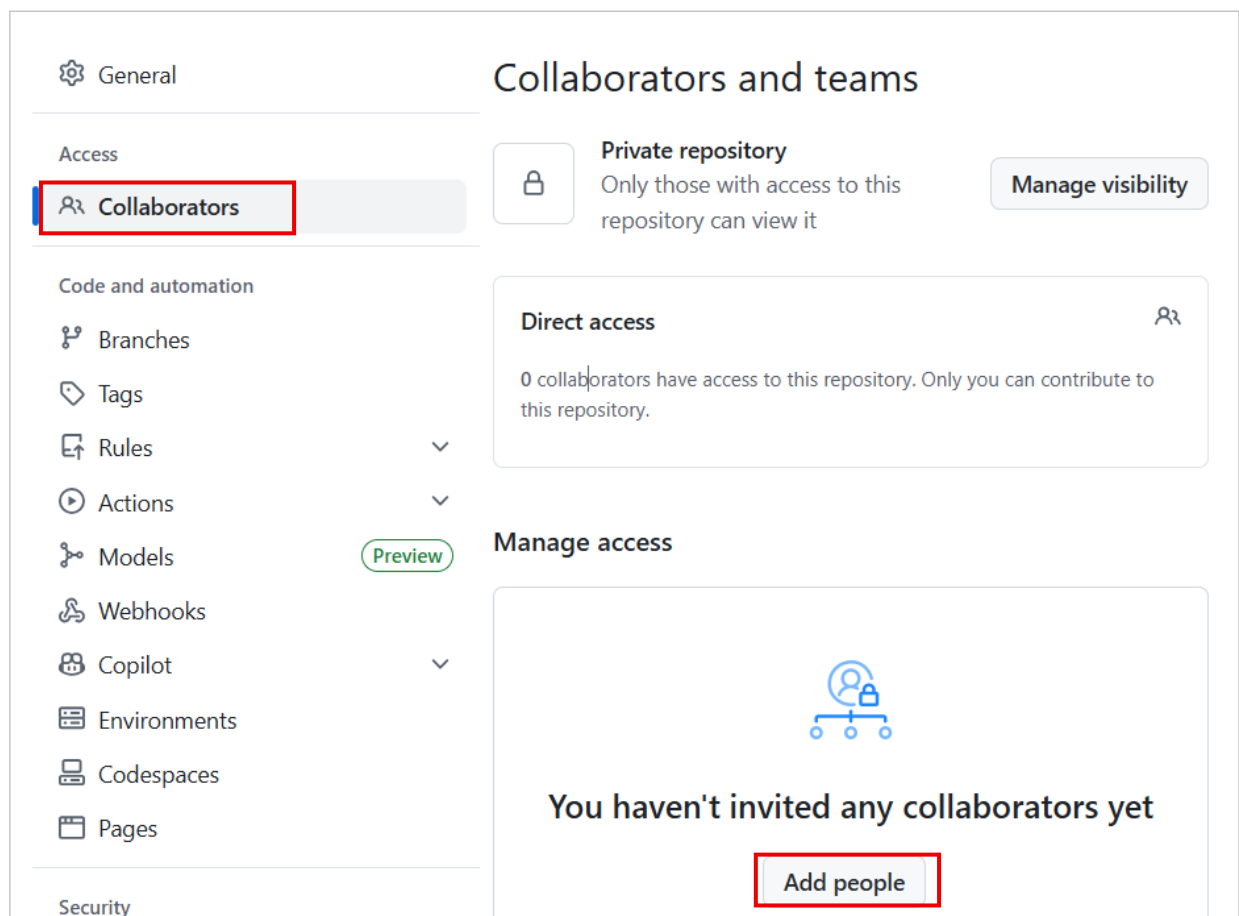
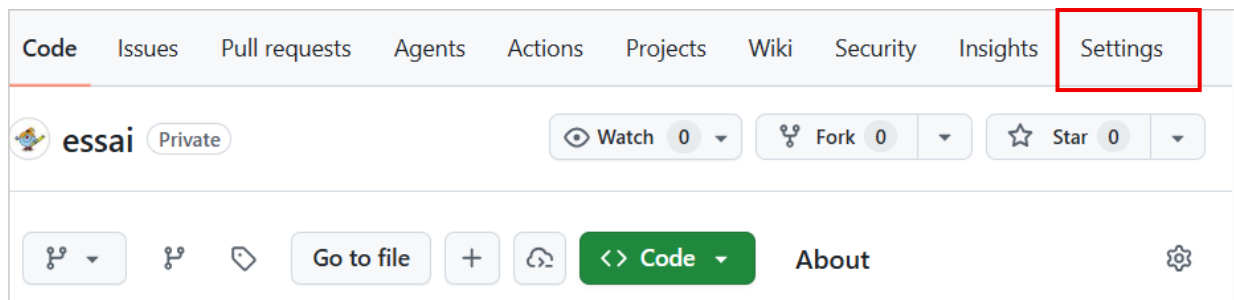
3. Mise en place sur Github

- 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

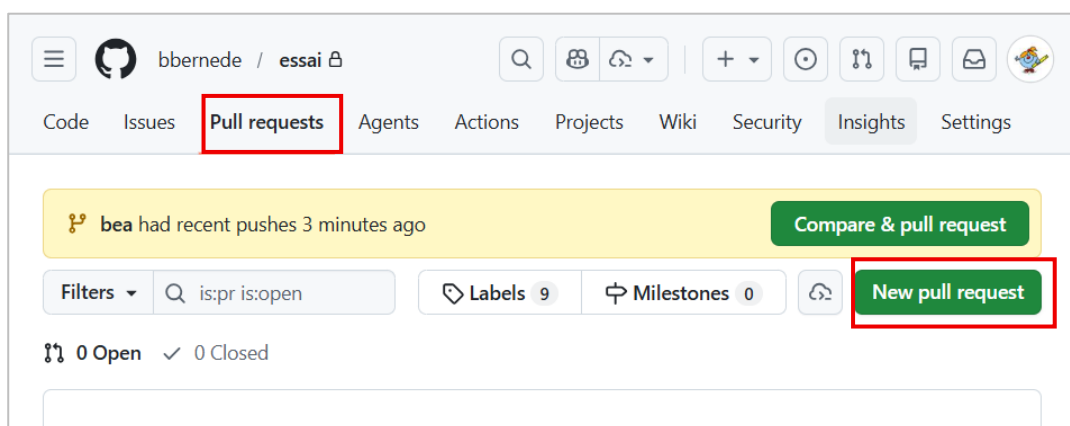
```
git switch nomBranche
```

```
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 :

The screenshot shows the GitHub interface for a repository named 'bbernede / essai'. The 'Pull requests' tab is selected. The main heading is 'Comparing changes'. Below it, there's a note: 'Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#) or [learn more about diff comparisons](#).' Below this, there's a comparison bar with 'base: dev' and 'compare: bea'. A green checkmark indicates 'Able to merge. These branches can be automatically merged.' Below the comparison bar, there's a blue box with the text 'Discuss and review the changes in this comparison with others. [Learn about pull requests](#)' and a green 'Create pull request' button. Below this, there's a summary bar showing '1 commit', '2 files changed', and '1 contributor'. Below the summary bar, there's a commit list showing a commit by 'bbernede' titled 'modif fic1.txt et ajout fic3.txt' committed 14 minutes ago. Below the commit list, there's a diff view showing changes to 'fic1.txt' with 2 additions and 1 deletion. The diff view shows a line of text: 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc sed diam sit amet ipsum tincidunt luctus ut ac massa. Pellentesque sapien ligula, interdum in facilisis nec,'.

Attention ici, on fait la Pull Request de la branche collaborateur vers la branche dev

Ici, on visualise les modifications qui vont être apportées à dev

The screenshot shows the 'Open a pull request' form in GitHub. The title is 'modif fic1.txt et ajout fic3.txt'. The description is 'Ici une description des modifications/ajouts apportés au projet'. The 'Assignees' section is open, showing a list of users to assign the pull request to. The list includes 'bbernede ProfSIO', 'copilot-pull-request-reviewer Copilot', 'Copilot Your AI pair programmer', and 'slamjr slamjr'. A red box highlights the 'Assignees' section, and a red arrow points to the 'Assignees' button in the 'Assignees' section.

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

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#). [Learn more about diff comparisons here](#).

 base: dev < ... compare: bea < ✓ **Able to merge.** These branches can be automatically merged.

**Add a title ***
modif fic1.txt et ajout fic3.txt

Add a description
Write Preview       ...
Ici une description des modifications/ajouts apportés au projet
Markdown is supported Paste, drop, or click to add files

Reviewers
Suggestions
 Copilot [Request](#)

Assignees
 slamjr

Labels
None yet

Projects
None yet

Milestone
No milestone

Development
Use [Closing keywords](#) in the description to automatically close issues

Create pull request

- 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

modif fic1.txt et ajout fic3.txt #1 Edit <> Code

Open bbernede wants to merge 1 commit into dev from bea

Conversation 0 Commits 1 Checks 0 Files changed 2 +2 -1

**bbernede** commented 2 minutes ago
Ici une description des modifications/ajouts apportés au projet
Mention @copilot in a comment to make changes to this pull request.
modif fic1.txt et ajout fic3.txt 43e82c6
bbernede assigned slamjr 2 minutes ago

Reviewers
Suggestions
 Copilot [Request](#)
Still in progress? [Convert to draft](#)

Assignees
 slamjr

Labels
None yet

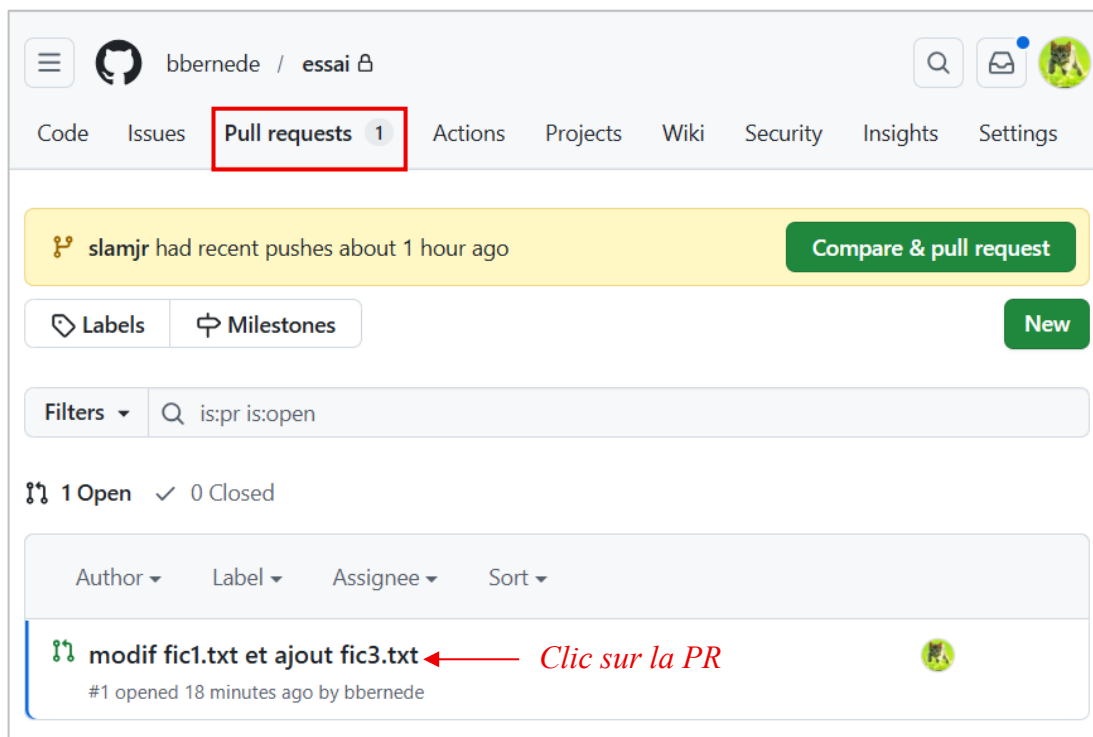
Projects
None yet

Milestone
No milestone

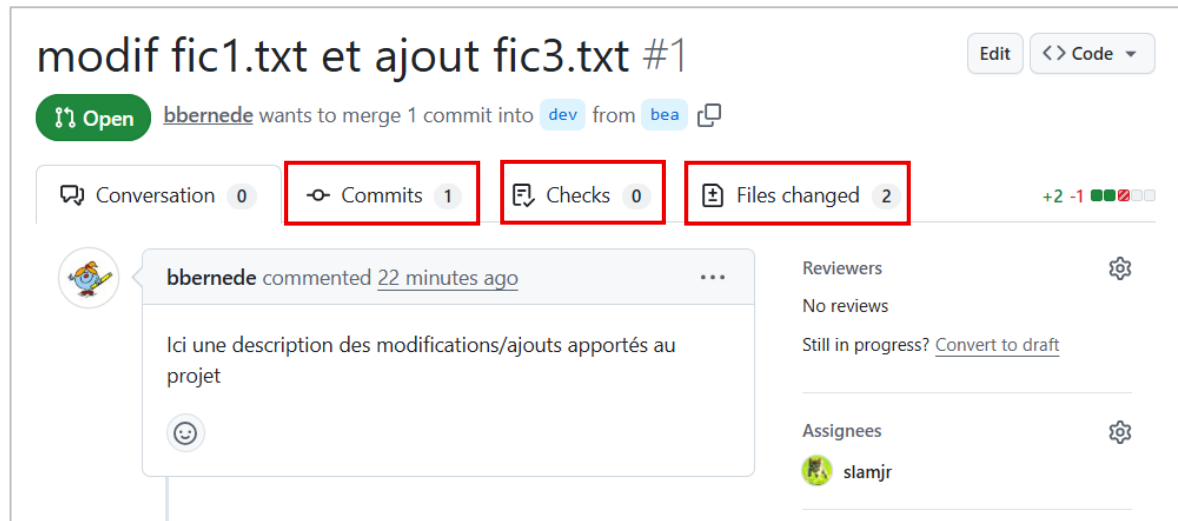
 **No conflicts with base branch**
Merging can be performed automatically.

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 :

The screenshot shows a pull request interface. At the top, a green box indicates 'No conflicts with base branch' and 'Merging can be performed automatically.' Below this is a green button labeled 'Merge pull request' with a dropdown arrow. To the right of the button, text says 'You can also merge this with the command line.' and a link 'View command line instructions.' Below the button, it says 'Still in progress? Convert to draft'.

The 'Add a comment' section is highlighted with a red box. It includes a user profile picture, the text 'Add a comment', and a text area with the content 'Revue de code OK'. Above the text area are tabs for 'Write' and 'Preview', and a set of formatting tools (H, B, I, list, link, code, etc.). Below the text area, it says 'Markdown is supported' and 'Paste, drop, or click to add files'.

On the right side of the interface, there are sections for 'Milestone' (No milestone), 'Development' (Successfully merging this pull request may close these issues. None yet), 'Notifications' (Unsubscribe button), and '2 participants' (with two user avatars).

- Enfin, il approuve ou refuse la demande de fusion (merge) :

- **Pour REFUSER** la demande de merge, cliquer sur **Close** pull request.

The screenshot shows the same pull request interface as before, but with the 'Close with comment' button highlighted with a red box. The 'Merge pull request' button now has a dropdown arrow, and a tooltip 'Select merge method' is visible. The 'Add a comment' section is still present with the text 'Revue de code OK'.

At the bottom of the 'Add a comment' section, there is a red box around the 'Close with comment' button, which has a red icon of a person with a red X. Next to it is a green 'Comment' button.

The right side of the interface remains the same, showing 'Milestone', 'Development', 'Notifications', and '2 participants'.

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

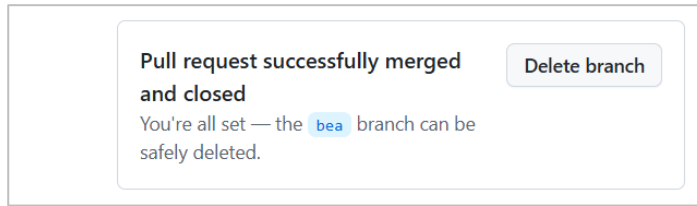
Le choix « **Squash and merge** » permet de réduire le nombre de commit transférés dans la branche dev.

The screenshot shows a GitHub pull request interface. At the top, a green box indicates 'No conflicts with base branch' and 'Merging can be performed automatically.' Below this, a green button labeled 'Squash and merge' is highlighted with a red rectangular box. To the right of the button, text says 'You can also merge this with the command line.' and a link 'View command line instructions.' is provided. Below the button, it says 'Still in progress? Convert to draft'. On the right side of the interface, there are sections for 'Milestone' (None yet), 'Development' (Successfully merging this pull request may close these issues.), 'Notifications' (Unsubscribe button), and '2 participants' (with two avatars). At the bottom, there is a 'Comment' button and a 'Close with comment' button.

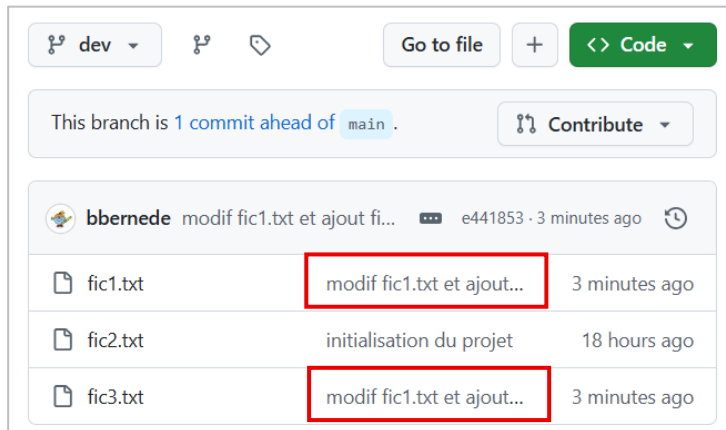
Il faut alors confirmer après avoir indiqué une description précise des modifications apportées au projet :

The screenshot shows a GitHub pull request interface. At the top, a green button labeled 'Open' is visible. Below it, the pull request title is 'modif fic1.txt et ajout fic3.txt #2' and the description is 'bbernede wants to merge 1 commit into dev from bea'. The commit hash '43e82c6' is also shown. Below the title, there is a section for 'Commit message' with the text 'modif fic1.txt et ajout fic3.txt (#2)'. Below this, there is a section for 'Extended description' with the text 'Ici une description précise des modifications apportées au projet'. At the bottom, there is a green button labeled 'Confirm squash and merge' highlighted with a red rectangular box, and a 'Cancel' button next to it. On the right side of the interface, there are sections for 'Labels' (None yet), 'Projects' (None yet), 'Milestone' (None yet), 'Development' (Successfully merging this pull request may close these issues.), 'Notifications' (Unsubscribe button), and '2 participants' (with two avatars). At the bottom, it says 'Still in progress? Convert to draft'.

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 :



- **Prévenir les autres membres de l'équipe** 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
```


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	<code>git switch dev</code>
2. Récupérer la version distante de la branche dev	<code>git pull origin dev</code>
3. Aller sur votre branche	<code>git switch <i>nomBranche</i></code>
4. Fusionner dev dans votre branche	<code>git merge dev</code>

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 ?)
```