Tutoriel atelier MAIA GIT

Exercices pratiques 1, 2 et 3

Animation : Nicolas B., François G., Philippe V., Rémi V., Ghislain V.

Table des matières :

Exercice 1, utilisation d'un dépôt GIT individuel

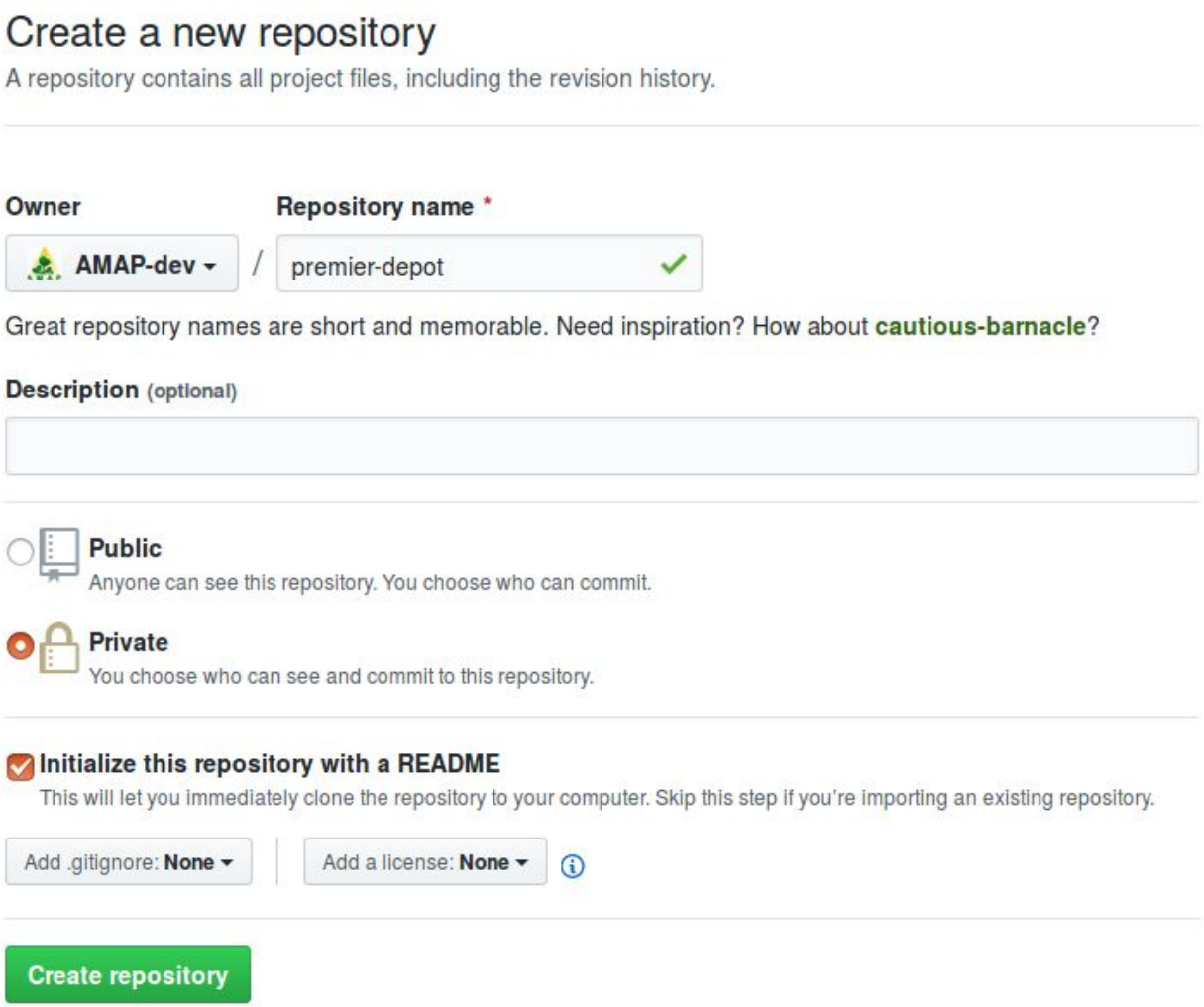
Cet exercice apprend à créer un nouveau dépôt sur Github, à récupérer une copie de travail en local, à manipuler les commandes de base localement en mode invite de commande et enfin à synchroniser son travail en local avec le dépôt distant.

Connexion à son espace personnel sur Github

https://github.com/login

Création d'un dépôt distant sur Github

https://github.com/new



Créer un nouveau dépôt privé "premier-depot" par exemple et cocher l'option "README".

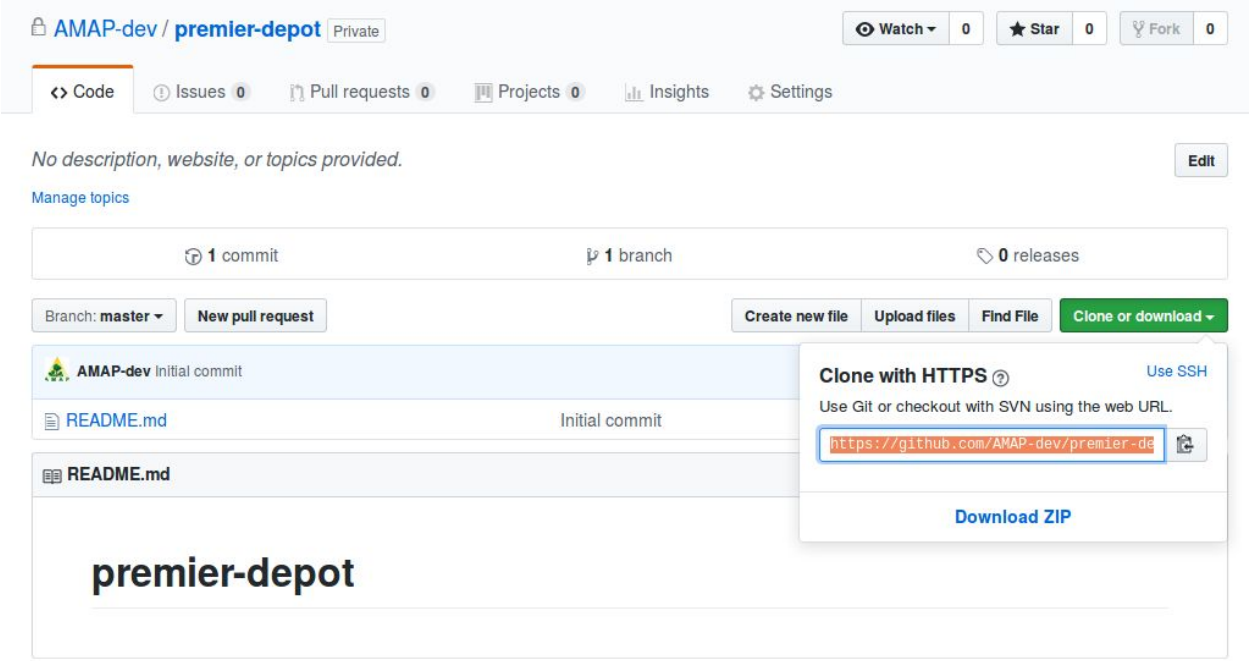
Valider avec un clic sur le bouton “Create repository”.

Cloner le dépôt distant en local

Ouvrir un invite de commande GIT en local.

> Invite de commande GIT git clone https://github.com/*votre-login-github*/premier-depot.git

Pour trouver l’adresse exacte de votre dépôt, aller sur la page du dépôt dans github et cliquer sur le bouton “Clone or download”.



Vous devrez saisir vos login et mot de passe github.

Ouvrir l’explorateur de fichier et constater la création d’un nouveau dossier premier-depot qui contient un dossier .git et un fichier README.md.

Modification du fichier README.md

Le fichier README.md est une commodité de github qui n’est pas nécessaire à un dépôt GIT (contrairement au dossier .git). Il permet toutefois de fournir une description de votre dépôt qui sera automatiquement mise en forme sur la page web du dépôt sur github.

Editer le fichier README.md avec un éditeur “simple” (type wordpad ou Gedit mais pas de LibreOffice ou Word) et le compléter comme suit :

```
# premier-depot
```

Premier dépôt GIT créé dans le cadre de l'atelier MAIA du 04 avril 2019

Contenu

A ce stade nous avons appris à cloner le dépôt :

```
https://github.com/mon-login-github/premier-depot.git
```

Ensuite nous allons rajouter un fichier texte avec du latin dedans. On en profite pour découvrir les deux trois premiers rudiments de MarkDown. Les listes par exemple :

* item 1
* item 2
* item 3

Une ligne verticale c'est --- avec une ligne vide au-dessus et en dessous :

Remerciement

Merci !

Sauver et fermer README.txt

Invite de commande GIT `git status`

GIT devrait vous dire que le fichier README.txt est modifié.

Invite de commande GIT `git diff` pour voir les modifications apportées.

Pour valider cette modification :

Invite de commande GIT `git add README.txt`

Invite de commande GIT `git commit -m “Modification du readme”`

Pour voir la liste des commits :

Invite de commande GIT `git log`

Pour envoyer cette modification sur le dépôt distant :

Invite de commande GIT `git push`

Il vous faudra saisir vos login et mot de passe github. Si vous retournez sur la page github de votre dépôt, vous devriez voir que le fichier README.md est automatiquement interprété et mis en forme par la page web.

Ajout d'un fichier texte dans le dépôt et premier commit

Créer en local un nouveau fichier texte latin.txt à côté de README.md et y copier le texte suivant : https://www.faux-texte.com/lorem-ipsum-3.htm. Si vous êtes allergiques au latin vous pouvez choisir un autre langage : https://randomtextgenerator.com et copier 3 paragraphes. Enregistrer et fermer l'éditeur de texte.

Constater que GIT a détecté un nouveau fichier :

> Invite de commande GIT `git status`

Ajouter le nouveau fichier :

> Invite de commande GIT `git add latin.txt`
>
> Invite de commande GIT `git commit -m "Ajout du nouveau fichier latin.txt"`

Modifications du fichier texte et nouveaux commits

Editer le fichier latin.txt et apporter une modification quelconque au premier paragraphe.

De la même façon que pour README.md, une fois la modification apportée vous allez dans l'ordre :

- Constater la modification apportée : `git status`, `git diff`
- Valider la modification : `git add`, `git commit -m "Modification du paragraphe 1 dans latin.txt"`
- Lister les modifications apportées : `git log`

Répéter l'opération avec une modification quelconque du deuxième paragraphe.

Récupération temporaire d'une version antérieure

Imaginons que vous vouliez revenir à l'état du dépôt avant la modification du deuxième paragraphe de latin.txt.

Tout d'abord lister les modifications apportées :

> Invite de commande GIT `git log`

Noter le hash du commit sur lequel vous souhaitez revenir puis

> Invite de commande GIT `git checkout` *`hash-du-commit`*

Alternativement si vous voulez simplement revenir à l'avant-dernier commit :

Invite de commande GIT git checkout HEAD~1

Pour revenir au dernier commit :

Invite de commande GIT git checkout master

Création d'une branche et travail sur la branche

Vous souhaitez maintenant réaliser un travail en profondeur sur le paragraphe trois de latin.txt, sans que cela interfère avec d'autres modifications que vous pourrez apporter dans le même temps aux paragraphes 1 ou 2. Pour cela nous utilisons une branche.

Tout d'abord créer la branche de nom latin-section3 :

Invite de commande GIT git branch latin-section3 crée la branche

Invite de commande GIT git checkout latin-section3 déplace sur la nouvelle branche

Version condensée des deux commandes précédentes :

Invite de commande GIT git checkout -b latin-section3

Pour confirmer que vous êtes bien dans la nouvelle branche :

Invite de commande GIT git branch

Comme précédemment apporter par exemple deux modifications au troisième paragraphe du fichier latin.txt et en faire deux commits.

Quand les modifications sont commitées, retourner à la branche principale :

Invite de commande GIT git checkout master

Constater que le fichier latin.txt ne présente pas les modifications apportées sur la branche latin-section3. Apporter une modification sur le premier ou le deuxième paragraphe et la commiter.

Quand les modifications sur le master sont commitées, retourner sur la branche de travail :

Invite de commande GIT git checkout latin-section3

Constater que le fichier latin.txt ne présente pas la dernière modification apportée sur la branche master dans le paragraphe 2.

Fusion d'une branche à une autre

Nous souhaitons fusionner les modifications de la branche latin-section3 dans la branche master.

> Invite de commande GIT git checkout master
>
> Invite de commande GIT git merge latin-section3
>
> Invite de commande GIT git status
>
> Invite de commande GIT git log

Pour aller plus loin. Que ce serait-il passé au moment de la fusion si à l'étape précédente, sur la branche master, nous avions apporté des modifications sur le troisième paragraphe du fichier latin.txt ? (réponse à l'exercice 2)

Envoi des changements locaux vers le dépôt distant

Lister les commits qui sont en local mais pas sur le dépôt distant :

> Invite de commande GIT git log origin/master..HEAD

Envoyer les modifications sur le dépôt distant :

> Invite de commande GIT git push

Retourner sur la page github de votre dépôt, rafraîchir l'affichage si besoin et observer l'historique des modifications.

Reprendre tout ou partie de l'exercice 1 avec un client GIT avec interface graphique (Github Desktop, Git Kraken, etc.)

Fin de l'exercice 1.

Exercice 2, gestion d'un conflit

L'objectif de cet exercice est de créer une situation de conflit en local au moment de fusionner deux branches et de résoudre ce conflit.

Créer une situation de conflit en local

Comme pour l'exercice 1, créer une branche, par exemple *latin-conflit* (git checkout -b latin-conflit) et apporter quelques modifications au fichier latin.txt avec des nouveaux commits pour chacune de ces modifications.

Revenir sur la branche principale (git checkout master) et apporter de la même manière une ou des modifications au fichier latin.txt en veillant à ce qu'au moins une des modifications concerne un même paragraphe que les modifications apportées sur la branche *latin-conflit*.

Imaginons maintenant que vous vouliez fusionner la branche latin-conflit avec la branche master. Il y a des modifications concurrentes et contradictoires du fichier latin.txt et cela crée une situation de conflit.

Fusion à blanc et annulation

Fusionner "à blanc" la branche latin-conflit pour vérifier présence / absence de conflit :

> Invite de commande GIT git merge --no-commit --no-ff

La commande devrait indiquer le conflit :

```
CONFLICT (content): Merge conflict in latin.txt
Automatic merge failed; fix conflicts and then commit the result.
```

> Invite de commande GIT git status

La commande git status renvoie normalement both modified: latin.txt pour faire état du conflit.

Pour annuler l'essai de fusion :

> Invite de commande GIT git merge --abort

Évaluer la situation de conflit

Vous savez maintenant que vous avez un conflit pour fusionner les branches. Avant de résoudre le conflit vous voulez évaluer clairement les commits en jeu dans les deux branches.

Lister les commits de la branche latin-conflit qui ne sont pas communs avec la branche master :

Invite de commande GIT git log master..latin-conflit

Lister les commits de la branche master qui ne sont pas communs avec la branche latin-conflit :

Invite de commande GIT git log latin-conflit..master

Lister l'ensemble des commits impliqués dans la fusion des deux branches :

Invite de commande GIT git log master...latin-conflit

Visualiser clairement la provenance des commits impliqués dans la fusion des deux branches

Invite de commande GIT git log --oneline --decorate --left-right --graph master...latin-conflit

Les logs informent sur les commits en jeu dans le conflit mais ne permettent pas de voir les modifications contradictoires du ou des fichiers concernés. Pour cela il faut utiliser la commande diff.

Visualiser les modifications apportées à la branche latin-conflit :

Invite de commande GIT git diff master...latin-conflit

Visualiser les modifications apportées à la branche master :

Invite de commande GIT git diff latin-conflit...master

Visualiser les différences entre les deux branches :

Invite de commande GIT git diff master latin-conflit

Pour aller plus loin dans la compréhension des "foo..bar" et des "foo...bar" une recherche "git diff three dots" vous renverra vers des discussions stackoverflow éclairantes.

La commande git show permet de combiner la visualisation des commits et des modifications. Vous pouvez l'essayer pour aller plus loin.

Résoudre le conflit

Vous avez les idées claires sur la nature du conflit et vous êtes prêt à le résoudre le conflit manuellement. GIT propose un outil de résolution des conflits.

Invite de commande GIT git mergetool

L'outil de résolution s'appuie sur les outils dont vous disposez en local pour éditer des versions de fichiers conflictuelles. Pour Windows ou Mac nous recommandons d'installer Atom et pour Linux l'utilitaire Meld (disponible aussi pour Windows). A défaut d'outils dédiés GIT proposera d'utiliser le bloc note ou tout autre éditeur de texte rudimentaire. Le principe est toutefois

similaire quelque soit l'outil : GIT présente les modifications conflictuelles du fichier et il faut les fusionner manuellement en une version satisfaisante.

Quand la résolution du conflit est terminée il ne reste plus qu'à faire un commit classique.

Terminer cette partie en supprimant la branche locale latin-conflit

```
Invite de commande GIT git branch -d latin-conflit
```

Fin de l'exercice 2.

Exercice 3, utilisation d'un dépôt GIT partagé

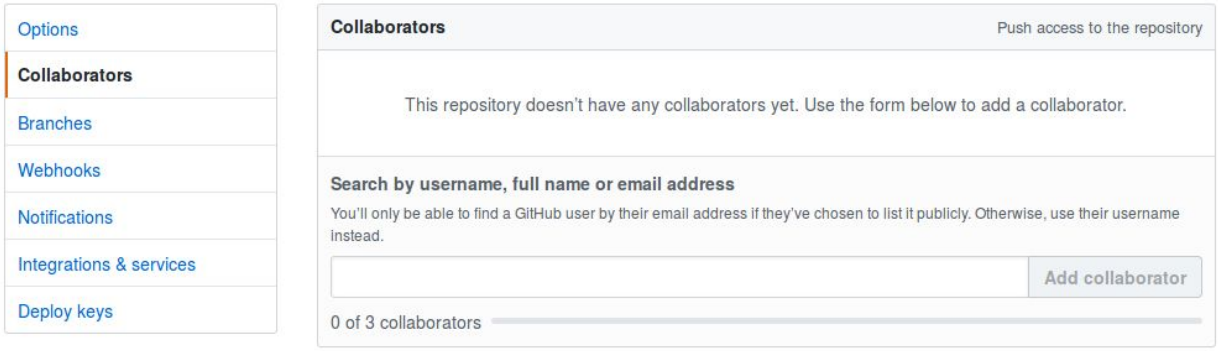
L'objectif de cet exercice est multiple :

- récupérer un dépôt GIT créé par un tiers (ou un pair !);
- récupérer et mettre à jour des modifications apportées de façon concomitantes par d'autres collègues et vous-même;
- gérer les situations conflictuelles.

Cet exercice doit être fait en binôme (ou trinôme). A la suite les deux personnes du binôme seront mentionnées comme P1 et P2.

P1 autorise P2 à accéder à son dépôt

Aller sur github.com, la page des paramètres du dépôt privé "premier-depot" dans la rubrique "Collaborators". P1 ajoute P2 comme collaborateur.



P2 clone le dépôt de P1

> P2 : Invite de commande GIT `git clone https://github.com/*login-p1*/premier-depot.git premier-depot_p1` (remplacer *login-p1* par le login github de P1)

P1 et P2 apportent des modifications indépendantes

P1 : créer un fichier charabia_p1.txt, faire un commit et un push pour l'envoyer.

P2 : créer un fichier charabia_p2.txt, faire un commit et un push pour l'envoyer.

Pour générer du texte aléatoire https://randomtextgenerator.com

Faire quelques modifications sur les fichiers respectifs charabia_p1.txt et charabia_p2.txt, les commiter et les envoyer avec un push sur le dépôt distant de P1.

Récupérer les modifications envoyées par l'un et l'autre sur le dépôt distant de P1 :

P1&P2 : Invite de commande GIT git pull

P1 : Invite de commande GIT git log --author=*login-p2*

P2 : Invite de commande GIT git log --author=*login-p1*

Pour voir le détail d'un commit du binôme utiliser la commande git show *hash-du-commit*.

P1 et P2 travaillent sur un même fichier appartenant à P1

P1 : apporter deux nouvelles modifications sur deux paragraphes différents du fichier charabia_p1.txt, chacune faisant l'objet d'un commit.

P2 : apporter deux nouvelles modifications sur charabia_p1.txt, le fichier de P1. Se concerter avec P1 pour qu'une modification touche un paragraphe modifié par P1 et l'autre un paragraphe non modifié par P1. Créer deux commits.

P1 : envoyer les deux nouveaux commit sur le dépôt distant avec git push.

P2 : quand P1 a terminé son push, essayez vous aussi de faire un git push. Il devrait être rejeté au motif "because the tip of your current branch is behind".

P2 : récupérer les derniers commits de P1 avec un git pull. De façon prévisible cela devrait générer une situation de conflit puisque un même fichier (charabia_p1.txt) est modifié de façon concomitante et contradictoire par différents commits.

P2 : annuler la tentative de fusion git merge --abort

P2 récupèrent les modifications de P1 en toute sécurité

Git permet de récupérer les modifications distantes sans interférer avec votre travail en local :

Invite de commande GIT git fetch

S'informer des commits distants récupérés par la commande fetch :

Invite de commande GIT git log origin pour voir l'intégralité de la séquence des commits du dépôt distant.

Invite de commande GIT git log ..origin (raccourci de git log HEAD...origin/HEAD) pour ne voir que les nouveaux commits distants par rapport à votre dépôt local.

> Invite de commande GIT `git log origin..` (raccourci de git log origin/HEAD..HEAD) pour ne voir que les commits de votre dépôt en local.
>
> Invite de commande GIT `git log --oneline --decorate --left-right --graph HEAD...origin/HEAD` pour voir la synthèse des commits faits sur les dépôts local et distant.

Comparer le dépôt local et le dépôt distant :

> Invite de commande GIT `git diff ...origin` (raccourci de git diff HEAD...origin/HEAD)

Fusionner “à blanc” les dépôts local et distant pour voir les changements et éventuels conflits :

> Invite de commande GIT `git merge --no-commit --no-ff`

Si pas de conflit, vous pouvez faire un commit de la fusion et l’envoyer sur le dépôt distant avec un push. Si conflit, voir section suivante.

P2 résout le conflit avec les modifications de P1

Au moment de la fusion GIT a identifié une situation conflictuelle sur le fichier charabia_p1.txt (git status renvoie both modified: charabia_p1.txt). Il vous faut résoudre le conflit manuellement.

Une première situation consiste à dire que vous renoncez à votre commit local (par exemple parce que le commit distant apporte déjà les modifications que vous visez avec votre commit local). Pour supprimer un commit qui n’a pas été envoyé sur le serveur :

> Invite de commande GIT `git log origin..` pour lister vos commits non “pushés” sur le dépôt distant.
>
> Invite de commande GIT `git reset --hard` *`hash-du-commit`* pour supprimer tous les commits locaux jusqu’à celui identifié par *hash-du-commit*, ce dernier inclus.
>
> Invite de commande GIT `git reset --hard HEAD~1` pour supprimer le dernier commit local.

La situation la plus courante est la résolution du conflit manuellement parce que les modifications locales et distantes sont importantes et il faut les accommoder. Le conflit se résout de la même façon que fait précédemment en local dans l’exercice 2 avec l’outil dédié de résolution des conflits.

> Invite de commande GIT `git mergetool`

Une fois la résolution du conflit terminée vous pouvez commiter les changements et envoyer les modifications sur le dépôt distant avec un push. P1 fait de son côté un `git pull` pour récupérer le commit de la fusion.

P1 et P2 inversent les rôles

Reprendre l'exercice à partir de la section "P1 et P2 travaillent sur un même fichier appartenant à P1" en inversant les rôles. C'est à dire que P1 et P2 travaillent sur le fichier charabia_p2.txt. Dans le détail :

P2 : apporter deux nouvelles modifications sur deux paragraphes différents du fichier charabia_p2.txt, chacune faisant l'objet d'un commit.

P1 : apporter deux nouvelles modifications sur charabia_p2.txt, le fichier de P2. Se concerter avec P2 pour qu'une modification touche un paragraphe modifié par P2 et l'autre un paragraphe non modifié par P2. Créer deux commits.

P2 : envoyer les deux nouveaux commit sur le dépôt distant avec `git push`.

P1 : quand P2 a terminé son push, essayez vous aussi de faire un `git push`. Il devrait être rejeté au motif "because the tip of your current branch is behind".

P1 : récupérer les derniers commits de P2 avec un `git pull`. De façon prévisible cela devrait générer une situation de conflit puisque un même fichier (charabia_p2.txt) est modifié de façon concomitante et contradictoire par différents commits.

P1 : annuler la tentative de fusion `git merge --abort`.

P1 : récupérer les modifications distantes sans interférer avec le travail local avec `git fetch`.

P1 : utiliser judicieusement `git log` et `git diff` pour comprendre la nature du conflit.

P1 : décider si abandonner les modifications locales avec `git reset` ou résoudre le conflit manuellement avec `git mergetool`. Terminer par un `git push` pour envoyer les éventuelles modifications sur le dépôt distant.

P2 : récupérer les modifications de P1 avec `git pull`.

Les deux dépôts locaux de P1 et P2 sont maintenant au même niveau.

Fin de l'exercice 3.