Conseils pour l'utilisation de git pour l'UE projet

Avertissement

Cette feuille essaye de répondre à une question posée sur le salon `sysadmin_git` du mattermost pour vous permettre de démarrer rapidement sur l'utilisation de `git` pour l'UE projet.

Elle est du type « recette de cuisine » mais ne vous permettra pas d'avoir suffisamment de hauteur pour pouvoir vous adapter à toutes les situations pouvant apparaître dans divers workflows. Pour cela, vous devez *a minima* être capable de visualiser les différents objets et leurs interactions (par exemple, étant donnée une situation concrète, être capable de dessiner l'état du worktree local, de la staging-area locale, du dépôt local, des dépôts distants (en: remote), ainsi que leurs divers pointeurs (e.g. telle branche de tel remote pointe sur tel commit), puis de prévoir l'action d'une commande sur ce système).

Quel hébergement de vos dépôts partagés ?

Le cours d'administration système devrait vous rendre capable d'héberger vous-même vos dépôts partagés.

En attendant, je vous suggère d'héberger vos dépôts partagés sur le gitlab de l'université, qui se trouve à l'adresse https://gitlab.sorbonne-paris-nord.fr/

N'hésitez pas à inviter vos enseignant·es si vous avez besoin de retours sur l'utilisation de git (par exemple problème de merge, cf plus bas).

Comment créer un dépôt partagé sur gitlab ?

- connectez-vous sur le gitlab de l'université avec vos identifiants.
- choisissez un nom pour votre groupe, et créez un groupe gitlab à ce nom (cherchez « New group » dans la barre du haut).
- ajoutez les membres de votre groupe de projet à votre groupe gitlab (cherchez le lien « Invite your colleagues »)
- créez un dépôt partagé, dont le nom correspond au code que vous allez y mettre, par exemple `parabox` :
  - cliquez sur « New repository » puis « Create blank project »
  - sélectionnez « Private » pour « Visibility Level » do sorte à ce que ce dépôt ne soit visible que par les membres de votre groupe.
  - décochez « Initialize repository with a README » dans « Project Configuration » de sorte à partir d'un dépôt vide.
- Une fois votre dépôt créé, cliquez sur le bouton « Clone » en bleu et notez l'URL indiquée sous « Clone with HTTPS » (elle est de la forme `https://gitlab.sorbonne-paris-nord.fr/<nom_groupe>/<nom_dépôt>.git`)

Quel workflow ?

Certaines personnes m'ont demandé quel workflow suivre. Il faut comprendre que git est un outil flexible, qui permet plusieurs façons de s'organiser collectivement. C'est cette organisation collective qu'on appelle « workflow », elle n'est pas prescrite par git.

Par exemple, l'exercice « cadavre exquis » du TP en autonomie propose un workflow ou chaque participant·es modifie et pousse sa propre branche sur un dépôt central, et c'est un·e « release manager » qui merge ces branches dans la branche `main` en s'assurant que tout se passe bien.

Pour commencer, je vous propose d'utiliser un workflow très simple :

- votre groupe de projet partage un unique dépôt commun, hébergé sur gitlab (voir section précédente). Ce dépôt a une unique branche nommée `main`.
- chaque étudiant·e du groupe possède une copie du dépôt commun sur sa machine personnelle, obtenue par la commande :

```
$ git clone <URL>
```

Le remote gitlab sera nommé automatiquement `origin` (observez le fichier `.git/config` dans votre répertoire de travail)

- ensuite, suivez la boucle suivante :

  1. avant de faire des changements sur le code, récupérez l'état courant de la branche `main` qui se trouve sur le remote `origin` :

     ```
     $ git pull origin main
     ```

  2. faites vos changements et commitez-les sur la branche `main` de votre dépôt local (cf slides « boucle locale ») :

     ```
     $ # éditer et compiler un <fichier>, tester le résultat
     $ git add <fichier>
     $ git commit -m "<commentaire>"

     $ # éditer et compiler un <fichier>, tester le résultat
     $ git add <fichier>
     $ git commit -m "<commentaire>"

     ...
     ```

  3. lorsque vous êtes satisfait·e de vos changements, partagez-les en poussant votre branche locale `main` sur le dépôt commun `origin` :

     ```
     $ git push origin main
     ```

Que faire en cas de problème ?

Si la commande `git push origin main` refuse de pousser votre branche `main` sur le dépôt `origin`, c'est en général parce que la branche `origin/main` a été modifiée depuis votre dernier pull (lisez le message d'erreur !), de sorte que la branche `origin/main` n'est plus un ancêtre de votre branche locale `main`.

La raison la plus probable est qu'un·e autre étudiant·e a poussé sa propre branche `main` entre-temps. Il faut donc :

1. récupérer la nouvelle branche `origin/main` :

   ```
   $ git fetch origin main
   ```

2. merger cette branche dans la branche courante `main` :

   ```
   $ git merge origin/main
   ```

   Si le merge ne demande pas d'arbitrage humain, git fait le merge automatiquement et ouvre un éditeur pour vous proposer de modifier le commentaire `"Merge remote-tracking branch ..."`. Sauvegardez ce commentaire pour finaliser le commit.

   Si le merge demande un arbitrage humain, il vous faut modifier les fichiers qui portent un conflit, les ajouter à la staging area, puis commiter. Pour savoir où vous en êtes du merge en cours, utilisez la commande :

   ```
   $ git status
   ```

3. pousser votre branche locale `main` vers le dépôt `origin` :

   ```
   $ git push origin main
   ```

Pensez à pousser vos changements régulièrement pour éviter ce problème.

Si l'activité de votre groupe de projet est telle que ce problème apparaît trop souvent, réfléchissez à un workflow mettant en jeu plusieurs branches.