Travail collaboratif

UE12 P24 - Python

Contexte

- Vous travaillerez rarement seul·e·s sur un projet
- Il est plus fréquent de contribuer à un projet existant que de créer son propre projet
- Un même code source va donc exister en plusieurs endroits, en plusieurs versions
- On a besoin d’un logiciel de versionnage : le plus commun est Git

Hébergement du code

- Les 2 plateformes d’hébergement de code les plus courantes sont GitHub et GitLab
- Une instance publique existe : https://github.com et https://gitlab.com
- Vous pouvez également les installer chez vous
  - Ça permet de garder la maitrise de ses données

Organisation d’un projet

- Un projet est géré par des mainteneurs
- La référence officielle est la branche main ou master présente sur la plateforme
  - Elle peut être protégée : par exemple on peut empêcher qu’elle soit modifiée autrement que par une pull request

Règles classiques

Lorsqu'on est pas mainteneur d'un projet :

- On ne pousse pas de code sur une branche qui n'est pas la sienne
  - Et surtout pas sur main
- Pour proposer des modifications sur main, on passe par une pull request

Organisation des dossiers

> ⚠ **Warning**
>
> Évitez de mettre un dossier *gité* dans un autre dossier *gité*.
>
> - C’est possible mais ça prête à confusion

Le cas classique : vous êtes dans un dossier *gité* et vous faites un *git clone*.

Organisation des dossiers

> ⚠ **Warning**
>
> Lorsque vous faites `git clone`, le répertoire distant est cloné dans un nouveau dossier, pas directement dans le dossier courant

Organisation des dossiers

```
arnaud@MSI:D:\projets\mines\mon-cours-préféré\le-meilleur-projet(master)> git clone git@github.com:albanehasbroucq/Messenger.git
Cloning into 'Messenger'...
remote: Enumerating objects: 45, done.
remote: Counting objects: 100% (45/45), done.
remote: Compressing objects: 100% (29/29), done.
remote: Total 45 (delta 12), reused 42 (delta 10), pack-reused 0 (from 0)
Receiving objects: 100% (45/45), 981.84 KiB | 10.00 KiB/s, done.
Resolving deltas: 100% (12/12), done.
arnaud@MSI:D:\projets\mines\mon-cours-préféré\le-meilleur-projet(master)> git checkout -b add-argument-parser
Switched to a new branch 'add-argument-parser'
arnaud@MSI:D:\projets\mines\mon-cours-préféré\le-meilleur-projet(add-argument-parser)> ls

    Directory: D:\projets\mines\mon-cours-préféré\le-meilleur-projet

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d----          15/12/2024    18:58                Messenger

arnaud@MSI:D:\projets\mines\mon-cours-préféré\le-meilleur-projet(add-argument-parser)> cd .\Messenger\
arnaud@MSI:D:\projets\mines\mon-cours-préféré\le-meilleur-projet\Messenger(main)>
```

`.git`

Toutes les informations liées à un dépôt git donné sont stockées dans un dossier `.git`.

Lorsque vous utilisez une commande git :

- elle cherche le plus proche dossier parent contenant un dossier `.git`
- elle modifie le contenu de ce dossier `.git`
- éventuellement, elle modifie le reste du dossier parent

`.git`

Le dossier `.git` est entièrement géré par Git, il est donc souvent caché

- VS Code ne l'affiche pas
- `ls` ne l'affiche pas

Pour l'afficher :

Windows

```
ls -Hidden
# ou
ls -Force
```

Unix

```
ls -a
```

`.git`

Toutes les infos liées à Git y sont :

- l’historique des commits
- les branches
- les URL des dépôt distants (GitHub ou GitLab)

Organisation des dossiers

Évitez :

```
- mon-messenger
    - .git
    - messenger.py
    - messenger-14alarro
        - .git
        - messenger.py
```

Préférez :

```
- mon-messenger
    - .git
    - messenger.py
- messenger-14alarro
    - .git
    - messenger.py
```

Préparer un ajout de code

1. Demander accès au dépôt
2. Aller dans un dossier qui n’est pas gité
3. Cloner le dépôt
4. Aller dans le dossier nouvellement créé
5. Créer une nouvelle branche

Pendant un ajout de code

- Le mainteneur continue à faire des modifications
- Il faut donc regarder régulièrement si la branche `main` évolue
- Si elle évolue, des conflits peuvent apparaître
- Dans ce cas, vous devrez intégrer les modifications de `main` dans votre branche :

```
1 git merge main
```

et résoudre les conflits

Finaliser un ajout de code

1. Pousser votre branche sur le dépôt distant
2. Faire une pull request
3. Vérifier qu'il n'y a pas d'erreurs
4. S'il y en a, les corriger
5. Demander à un mainteneur de relire votre code
6. Prendre en compte ses retours

Recevoir une demande d’ajout de code

1. Relire toutes les modifications
2. Vérifier qu’il n’y a pas d’erreur
3. En cas de doute, récupérer la branche et faire des tests en local
4. Si besoin, demander des modifications au développeur
5. Fusionner la PR
6. Ramener les modifications sur votre machine