Compte rendu JEBBARI MOUAD

Tuto GIT :

Commandes GIT :

- **Git config** : Permet de contrôler les différents aspects de l'apparence et du fonctionnement de Git via des vrariables de configuration.
  Les paramètres de configuration courants sont par exemple : l'email, le nom d'utilisateur …

  Utilisation : Création d'un nom de configuration qui affiche la valeur définie à ce nom. Les noms de configuration sont composés d'une ''section'' et d'une ''clé'' séparées par un point.
  « git config user.email » On a le bloc de configuration user (la section), et la propriété de ce bloc qui correspond à l'email (la clé).

- **Git init** : Permet de générer un nouveau dépôt Git vide ou réinitialiser un dépôt existant.
  L'execution de la commande implique la création d'un répertoire .git (Il contiendra notamment des sous-répertoires pour les objets et les fichiers modèles nécessaires pour générer un nouveau dépôt Git.

  Utilisation : Exécuter la commande git init dans le terminal lorsqu'on se situe dans le sous-repertoire du projet permet de créer un dépôt Git. C'est généralement la première commande à lancer lorsqu'on commence un nouveau projet.

- **Git status** : Cette commande permet de dresser ce qu'on peut appeler un rapport de situation en permettant de voir les modifications mises à jour et les fichiers non suivis (trackés) par Git.

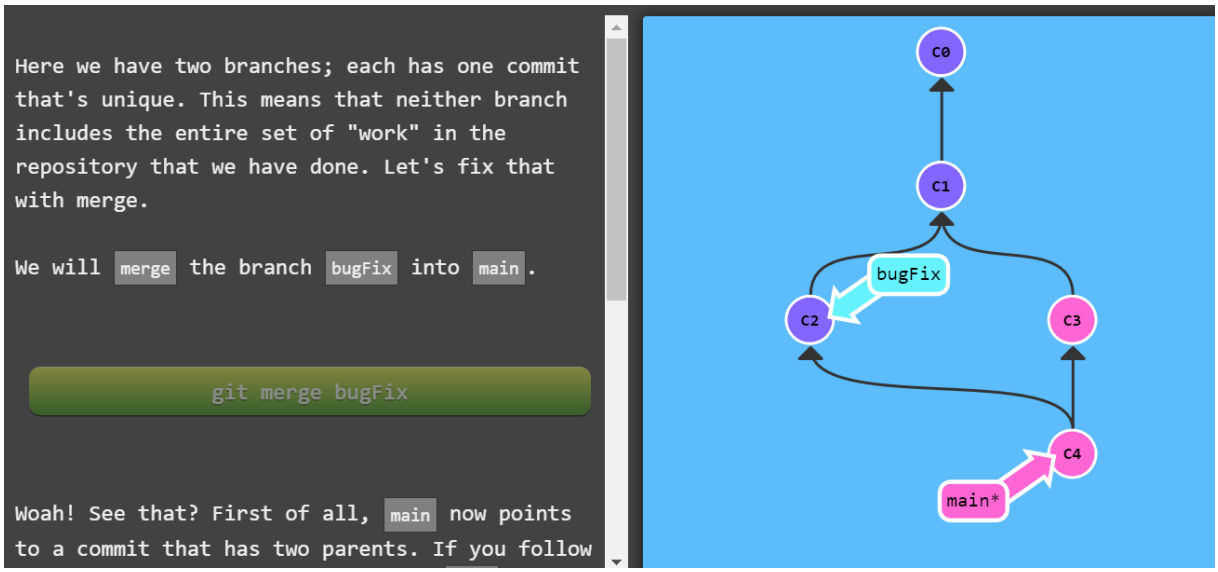
  Utilisation : Après exécution de la commande, on obtient une liste les fichiers qui sont répartis en trois catégories : staged, unstaged, et untracked.

```
git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   main.js

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    README.md
    index.html

no changes added to commit (use "git add" and/or "git commit -a")
```

- **Git add** : permet d'ajouter les modifications du répertoire de travail à la zone de transit.
  Utilisation : Lorsqu'on veut mettre à jour un fichier en particulier, la commande permet d'indiquer cela à Git. Mais les mises à jour ne seront enregistrés qu'après l'exécution de git commit.

- **Git push** : permet d'exporter le contenu du dépôt local vers le dépôt distant.
  Utilisation : Ainsi on peut effectuer des changements et les peaufiner sur notre dépôt local, et lorsqu'on le souhaite, git push permet de partager la modification avec les autres membres de l'équipe par exemple en exportant les modifications sur le dépôt distant.

- **Git merge** : permet la fusion de lignes indépendantes de développement en une seule branche.
  Utilisation : git merge permet de combiner deux branches ou même de fusionner plusieurs commits en un seul historique.
  Les commits générés avec git merge sont uniques car ils ont deux commits parents.
  Git fusionne automatiquement les historiques séparés lorsqu'un nouveau commit de fusion est créé.
  (Utilisé par exemple lorsqu'on veut exporter nos modifications sur le serveur distant mais que git refuse car on a pas mis à jour notre branche locale, git merge permet de récupérer des données du serveur distant et les combiner avec les changement apportés dans le serveur local. On peut ensuite exporter le commit sur le serveur distant)



- **Git diff** : permet d'afficher les modifications entre deux ensembles de données mis en entrée. Elle permet donc une comparaison.

- **Git blame** : peut être utilisée de différentes façons. Par exemple Elle peut être utilisée pour identifier le dernier auteur qui a modifié la ligne suite à l'exploration de l'historique du fichier. On peut également voir cela comme un tracker qui permettrait de suivre et identifier les lignes déplacées d'un fichier à un autre, ou celles qui ont été copiées et collées depuis un autre fichier…