Introduction à GIT

I Généralités

Définition : GIT est un logiciel qui permet de suivre les mises à jour d'un projet, il est stocké dans un dépôt local *(.git)*.

Vocabulaire : On appelle **repository** (ou *repo* pour les intimes) un dépôt GIT.

Fonctionnement : *(admis)*

Quand on modifie des fichiers dans VS Code *(i.e. dans le Working Directory)*, GIT détecte les changements (on les voit dans git status).

On peut alors choisir quels changements on veut enregistrer (staging area) avec la commande `git add <fichier>`.

Quand on a fait le tour des fichiers à sauvegarder, on peut faire un **commit** avec la commande `git commit -m "message"`. (on envoie les fichiers du staging area vers le dépôt local)

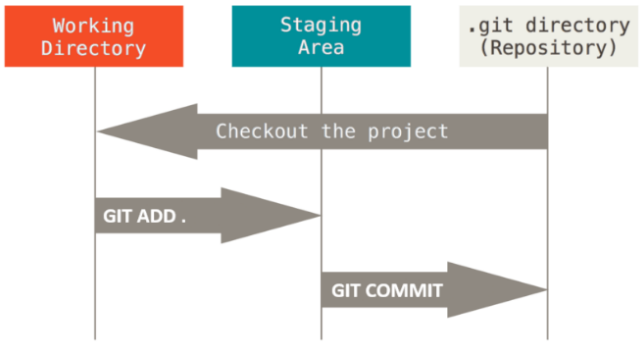

Figure 1: Cycle de vie d'un fichier dans GIT

Remarque : GIT stocke les changements, pas les fichiers entiers !

A Mise en commun

Définition : Pour partager son travail avec d'autres personnes, on utilise un dépôt distant (par exemple GitHub, GitLab, etc.).

Intuitivement, on peut imaginer que le dépôt distant est une copie du dépôt local. Il stocke les mêmes différences, sauf qu'on peut les partager.

Mise à jour du dépôt distant : *(admis)*

Pour envoyer les changements du dépôt local vers le dépôt distant, on utilise la commande `git push`.

Pour récupérer les changements faits par d'autres personnes sur le dépôt distant, on utilise la commande `git pull`.

✕ Attention ✕ Il faut toujours faire un `git pull` avant de faire un `git push`, sinon GIT va refuser d'envoyer les changements. (on doit avoir la dernière version du dépôt distant avant d'envoyer nos changements)

B Conflits et résolutions

Définition : Après avoir fait un `git pull`, il se peut que GIT ne puisse pas fusionner les changements automatiquement. On parle alors de **conflit**.

Exemple : Si Jean modifie la ligne 3 d'un fichier et que Paul modifie aussi la ligne 3 du même fichier, GIT ne sait pas quelle version garder. Il y a donc un conflit.
Le premier qui fait un `git pull` n'aura pas de problème, mais le deuxième aura un conflit à résoudre. (ici, Paul)

Remarque : Pour éviter d'avoir des conflits, il faut toujours faire un `git pull` avant de commencer à travailler sur un projet. Pour être celui qui devra traiter le moins de conflit, il faut faire des `git push` régulièrement, avant les autres ;).

Résolution de conflits : *(admis)*

Quand un conflit survient, GIT modifie les fichiers concernés et demande à l'utilisateur de résoudre le conflit manuellement.

Dans VS Code, les parties en conflit sont marquées avec des indicateurs «««, ====== et »»».

Il faut choisir quelle version garder (ou fusionner les deux) et supprimer les indicateurs, puis faire un commit.

Exemple :

```
<<<<<<< HEAD (Current Change)
def calcul_somme(a, b):
    # Version de Paul
    return a - b
=======
def calcul_somme(a, b):
    # Version de Paul
    return a * b
>>>>>>> feature/paul (Incoming Change)
```

Figure 2: Exemple de conflit dans VS Code

Remarque : VS Code propose une interface graphique pour résoudre les conflits, ce qui peut être plus simple que de le faire manuellement.

II Branches

Définition : Une branche est un ensemble de commits. Par défaut, GIT crée une branche appelée `main` (ou `master` dans les anciennes versions).

Utilisation des branches : *(admis)*

Les branches permettent de travailler sur des fonctionnalités ou des corrections de bugs sans affecter la branche principale.
C'est pratique pour tester des idées, ou pour développer une fonctionnalité et ne la rajouter à la branche principale que lorsqu'elle est terminée.

On peut créer une nouvelle branche avec la commande `git branch <nom_branche>`, puis se déplacer dessus avec `git checkout <nom_branche>`.

Remarque : `git checkout` est aussi utilisé pour revenir à une version précédente d'un fichier ou d'un commit.

Remarque : Depuis GIT 2.23, on peut utiliser la commande `git switch <nom_branche>` pour se déplacer sur une branche.

Fusion de branches : *(admis)*

Une fois le travail sur une branche terminé, on peut fusionner les changements dans la branche principale avec la commande `git merge <nom_branche> -m "message"`.

Il est recommandé de faire un `git pull` avant de fusionner pour éviter les conflits.
C'est comme un commit, mais pour une branche entière.

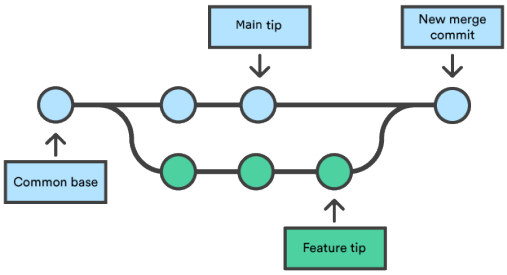

Figure 3: Fusion de branches