

## TP2 : Gestion de branches et de conflits

Dans ce TP, nous allons aborder les notions suivantes :

- La création et le changement de branches.
- L'intégration des modifications, c'est-à-dire la fusion de branches.
- La réécriture d'historique.
- La gestion de conflits de fusion.

### Partie 1 : Créer des branches

- 1-
  - a. Placez-vous dans un répertoire spécifique à ce TP
  - b. Créez un dépôt Git vide `exo1` puis créez-y trois commits `$ git config --global core.autocrlf false`
  - c. Listez les branches locales de votre dépôt grâce à la commande `git branch` sans paramètres, qui devrait vous dire que seule la branche `master` existe pour l'instant.
  - d. Créez maintenant une branche nommée `exemple1` sur votre dernier commit grâce à la commande `git branch <branch>`. Visualisez votre historique Git grâce à la commande `git log` pour vérifier que la branche a été créée au bon endroit.  
*Remarquez que votre branche actuelle est toujours `master` puisque `HEAD` pointe vers `master` — ce que vous pouvez également vérifier avec `git status` ou `git branch`.*
  - e. Placez-vous maintenant dans votre branche nouvellement créée grâce à `git checkout exemple1`. Créez un nouveau commit, puis visualisez votre historique Git. Que s'est-il passé pour la branche `exemple1` ? Et pour `HEAD` ?
  - f. Nous venons de voir comment créer une branche puis comment s'y déplacer avec la combinaison de commandes `git branch <branch>` suivie de `git checkout`. Cette opération est tellement courante qu'elle est faisable en une seule commande : `git checkout -B <branch>`. Créez une branche `exemple2` et placez-vous y grâce à cette seule commande.
  - g. Enfin, il est aussi relativement courant de vouloir créer une branche ailleurs que sur `HEAD`. C'est ce que permet le paramètre optionnel `<start-point>` dans `git branch <branch> [<start-point>]` et `git checkout -B <branch> [<start-point>]`. Créez une branche `exemple3` à partir du commit vers lequel `master` pointe grâce à une de ces commandes, puis visualisez votre historique pour vérifier que votre branche pointe au bon endroit — l'option `--all` de `git log` permet de voir toutes les branches

### Partie 2 : Intégrer des modifications avec `git merge`

Placez-vous de nouveau dans un répertoire spécifique à ce TP (par exemple. Dans cet exercice et les suivants, nous allons travailler à partir d'un dépôt Git existant plutôt que de partir d'un dépôt vide.

- a. Récupérez une copie locale du dépôt grâce à la commande suivante : `git clone https://gitlab.com/git-course-mpoquet/exercice-branch1.git`
- b. Jetez un œil aux fichiers de dépôt (e.g., en lançant `ls -R`) et visualisez l'historique du dépôt avec `git log --all`. Il devrait contenir trois branches :
  - `master`, la version maintenue du dépôt à jour.
  - `doc-no-conflicts`, qui ajoute une information dans la documentation.
  - `usable-as-lib`, qui ajoute la fonctionnalité de se servir de `hello.py` en tant que bibliothèque python en plus d'être un script exécutable.Ici, j'ai fait tous ces commits mais dans un cas d'utilisation réel, vous pouvez vous dire que les branches `usable-as-lib` et `doc-no-conflicts` ont été faites par différents contributeurs. On souhaite intégrer ces branches dans `master`. Une différence entre ces deux branches est que `doc-no-conflicts` est dite `fastforward` par rapport à `master` puisqu'elle est placée directement au-dessus de `master`, ce qui n'est pas le cas de `usable-as-lib`.

Vous pouvez exécuter la commande « `git log --graph --oneline --decorate --all` » afin de voir visuellement comment les commits sont liés les uns aux autres et de repérer si une branche est directement en amont de la branche `master`.
- c. Fusionnons maintenant `doc-no-conflicts` dans `master`. Pour cela, assurez-vous d'être dans `master` puis lancez `git merge --no-ff origin/doc-no-conflicts`
- d. Observez l'historique résultant après cette commande : les commits de `doc-no-conflicts` ont été intégrés dans `master` grâce à un commit de fusion (merge commit). Ce commit de fusion devrait avoir pour parent principal son ancêtre direct dans `master` (`0aacc9`) et pour parent secondaire le commit vers lequel `doc-no-conflicts` pointe (`2a69873`) — ce que vous pouvez vérifier grâce à `git log --first-parent`, qui n'affiche que le parent principal de chaque commit.

### **Partie 3 : Intégrer des modifications avec git rebase**

Une autre commande très importante pour intégrer des modifications est `git rebase`, qui permet comme son nom l'indique de changer la base d'une branche.

- a. Repartez d'un dépôt Git frais comme indiqué dans l'exercice précédent : `git clone https://gitlab.com/git-course-mpoquet/exercice-branch1.git`
- b. Nous allons illustrer le cas le plus courant d'appel de `git rebase` en voyant une autre manière d'intégrer la branche `usable-as-lib` dans `master`. Nous allons pour cela faire passer `usable-as-lib` en `fast-forward` par rapport à `master`. Visualisez votre historique Git et notez que le commit dans la branche `usable-as-lib` est `a784b13`. Après vous être assurés d'être dans la branche `usable-as-lib`, lancez la commande `git rebase master`. Visualisez de nouveau votre historique Git. Que s'est-il passé pour la branche `usable-as-lib` ? Le commit précédent

(`a784b13`) est-il toujours accessible dans l'historique ? Vous pouvez ensuite fusionner votre branche dans `master` grâce à `git merge`.