# Apprendre Git en créant un objet de référence SQL

## Qu'est-ce que c'est Git ?

- Git est un outil opensource.
- Git est un système de versioning
- Git est une référence dans le domaine
- Git, c'est la vie !

## Setup

L'installation de Git ne demande que peu d'efforts, après c'est tout un monde que vous avez dans votre ordinateur à travers lequel vous pouvez voyager.

N'oubliez pas de configurer votre identité pour vos futurs commits:

```
git config --global user.name <nom>
git config --global user.email <email>
```

## Vocabulaire

- Repository
- Branch
- Commit
- Pull Request

## Créons notre référence SQL

### 1. Créer un dossier

Créer un dossier `learn_sql` dans votre repertoire personnel et déplacez-vous dedans:

```
cd ~
mkdir learn_sql
cd learn_sql
```

## 2. Initialisez un repository

Dans votre dossier `learn_sql` , exécuter la commande

```
git init
```

Cette commande initialise un repository dans le dossier courant. Cela permet de traquer les changements qui s'opèrent dans ce dossier à partir de l'intant.

Pour vérifier, exécutez la commande:

```
git status
```

Cette commmande permet de recenser les fichiers modifiés non encore versionnés.

## 3. Notre premier commit

Nous allons commencer par créer un fichier `README.md` .

```
touch README.md
```

Vérifions l'état de notre repository:

```
git status
```

Nous remarquons que le fichier `README.md` est présent dans la partie des fichiers **untracked**.

Dans le fichier `README.md` , nous allons ajouter la ligne `# Référence SQL` .

Vérifions encore une fois l'état de notre repository:

```
git status
```

Toujours la même chose !

Nous allons maintenant enregistrer le fichier dans notre système de versionning:

Tout d'abord, on va l'ajouter en attente:

```
git add README.md
```

Vérifions l'état de notre repository:

```
git status
```

Là nous remarquons que le fichier `README.md` est passé en attente, on dit aussi en **staging**.

Nous allons enregistrer le changement maintenant:

```
git commit -m "feat: ajout fichier readme"
```

L'option `-m` nous permet de donner le message de commit.

Il est recommandé de choisir des messages clairs et concis. On les préfixe très souvent par `feat` , `fix` , etc.

Maintenant que nous avons enregistré le fichier, vérifions l'état de notre repertoire:

```
git status
```

Il semblerait que ça ait marché et qu'il n'y a plus rien à *commit*.

Listons les commit:

```
git log
```

Cette commande permet de voir l'historique des modification (des commits). Ou en clair, on peut apercevoir les messages laissés par les contributeurs du projet. Pour l'instant vous travaillez seul.

# 4. Notre première *feature branche*

Nous voulons continuer à créer notre référence SQL qu'on partagera à la communauté.

Pour cela nous commençons par créer une branche.

```
git branch feat/ajout-fichier-reference
```

Listons les branches:

```
git branch
```

Nous pouvons maintenant voir qu'il existe deux branches: `main` et `feat/ajout-fichier-reference` .
Nous remarquons `*` devant la branche `main` cela signifie que c'est la branche courante.

Nous allons changer de branche:

```
git checkout feat/ajout-fichier-reference
```

On peut vérifier que la branche a changé avec la même commande que tout à l'heure:

```
git branch
```

Cette fois-ci la branche `feat/ajout-fichier-reference` est précédée par une **étoile**.

Nous allons afficher les logs:

```
git log
```

Nous voyons le même historique que celui de la branche main.

> **Note:** La commande `git branch <nom branche>` permet de cloner la branche courante.

Créons un fichier `sql_reference.json` .

```
touch sql_reference.json
```

Dans ce fichier nous allons ajouter les lignes suivantes:

```
{
    "database": {
        "create": "CREATE DATABASE database_name;"
    }
}
```

Vérifier l'état du repertoire, nous avons un nouveau fichier à enregistrer.

```
git status
```

Ajoutez-le en *staging*.

```
git add sql_reference.json
```

Vérifier l'état de nouveau.

```
git status
```

Enregistrer les changement à travers un message de commit: `feat: ajout fichier de reference`

```
git commit -m "feat: ajout fichier de reference"
```

Vérifions l'état du dossier et consultons les logs. Nous remarquons deux messages de logs dans la branche courante ( `feat/ajout-fichier-reference` )

Revenons à la branche `main` :

```
git checkout main
```

Vérifions l'historique de cette branche. Vous avez noté la différence ?

> **Note:** Les branches sont puissantes car elles nous permettent de gérer deux historiques différents. Autrement, on peut travailler sur des features séparements.

Fusionnons la branche `feat/ajout-fichier-reference` à la branche `main` :

```
git merge feat/ajout-fichier-reference
```

Vérifions l'historique et aussi le contenu du fichier !

Nous allons maintenant supprimer la branche qu'on vient de fusionner.

```
git branch -d feat/ajout-fichier-references
```

Voici ainsi présentés les fonctionnalités principales de l'outil git:

- Les branches
- Les commits

## 5. La suite de notre référence

Nous allons continuer à ajouter des commandes dans notre objet de référence.

Tout d'abord nous allons créer une nouvelle branch `feat/ajout-reference-suppression-bdd` .

```
git branch feat/ajout-reference-suppression-bdd
git checkout feat/ajout-reference-suppression-bdd
```

ou alors

```
git checkout -b feat/ajout-reference-suppression-bdd
```

> **Note:** Les deux produisent le même résultat.

Ajoutons ensuite une entrée `drop` dans la partie database qui donne la commande pour supprimer une base de données.

Au final, nous devrions avoir un contenu identique que ce qui suit:

```
{
    "database": {
        "create": "CREATE DATABASE database_name;",
        "drop": "DROP DATABASE database_name;"
    }
}
```

Vérifions l'état de notre répertoire.

```
git status
```

Nous remarquons qu'un fichier a été modifié.

Nous pouvons aussi utiliser la commande diff pour lister les changements apportés:

```
git diff
```

Ajoutons le fichier avec le message: `feat: ajout reference suppression base de données`

```
git add sql_reference.json
git commit -m "feat: ajout reference suppression base de données"
```

Vous pouvez lister les logs

> **Note:** S'il y a beaucoup de logs, tout le contenu ne s'affiche pas en une fois, vous pouvez naviguer avec les touches de direction. Appuyez sur la touche **Q** pour sortir.

Aussi revenons à la branche principale `main` .

```
git checkout main
```

Nous allons fusionner les changements qu'on a ajoutés sur la branche précédente puis la supprimer.

```
git merge feat/ajout-reference-suppression-bdd
git log
git branch -d feat/ajout-reference-suppression-bdd
git branch
```

Nous allons maintenant créer une branche pour ajouter quelques références sur les tables:

Ajoutons une branche `feat/ajout-references-table`

```
git checkout -b feat/ajout-references-table
```

Rajoutons dans notre référence une entrée sur `table` contenant deux sous-entrées ( `create` et `drop` )

```
{
    "database": {
        // ... ancien contenu
    },
    "table": {
        "create": "CREATE TABLE table_name;",
        "drop": "DROP TABLE table_name;"
    }
}
```

Enregistrons ces changements avec le message: `feat: ajout references table` .

```
git status
git diff # pour voir les changements apportés aux fichiers
git add sql_reference.sql
git status
git commit -m "feat: ajout references table"
git status
git log --oneline
```

> **Note:** Le flag `--oneline` permet de voir les logs de façon plus condensée.

Fusionnons ces changements à la branche principale `main` .

```
git checkout main
git log --oneline # pour vérifier l'état actuel
git merge feat/ajout-references-table
git log --oneline
git branch -d feat/ajout-references-table
git branch # pour s'assurer de la suppression de la branche
```

Voilà ça !

Maintenant nous allons voir comment gérer plusieurs branches à la fois !

Créez une branche `feat/ajout-reference-insertion-ligne`

Ajoutez une entrée `row` dans le fichier de référence contenant une sous-entrée `insert` pour insérer une entrée dans une table:

```
{
    "database" : {
        // ... ancien contenu
    },
    "column": {
        // ... ancien contenu
    },
    "row": {
        "insert": "INSERT INTO table_name(columns_name) VALUES(column_values);"
    }
}
```

Committez les changements de la branche `feat/ajout-reference-insertion-ligne` avec le message: `feat: ajout reference insertion ligne` .

Revenez sur la branche `main` .

Créez une autre branche `feat/ajout-references-colonne` .

Dans cette branche, nous allons gérer les commandes en lien aux colonnes.

Commençons par ajouter une entrée `column` dans le fichier de référence contenant une sous-entrée `create` contenant la requête SQL pour créer une colonne dans une table `ALTER TABLE table_name ADD COLUMN column_name;` .

```
{
    "database" : {
        // ... ancien contenu
    },
    "column": {
        // ... ancien contenu
    },
    "column": {
        "create": "ALTER TABLE table_name ADD COLUMN column_name;"
    }
}
```

Committons ces modifications avec le message: `feat: ajout reference creation colonne` .

Revenons à la branche main pour effectuer une fusion de la branche `feat/ajout-reference-insertion-ligne` .

Supprimons cette branch après fusion.

Positionnons-nous maintenant au niveau de la branche `feat/ajout-references-colonne` .

> **Note:** Ceci arrive très souvent où nous devrions récupérer les modifications d'autres personnes. On effectue ce qu'on appelle **rebase** de notre branche.

Essayons:

```
git rebase main
```

Et hop !

En cas de conflits, nous devons les corriger puis mettre en attente les fichiers concernés.

```
git status
git add sql_reference.sql
```

Avant ensuite de continuer notre rebase.

```
git rebase --continue
```

Nous allons ensuite ajouter une sous-entrée `rename` sur la branche courante `feat/ajout-references-colonne` avec le contenu:

```
ALTER TABLE table_name RENAME COLUMN column_name TO new_name;
```

Voici à quoi devrait ressembler l'entrée `column` :

```
{
    // ... ancien contenu
    "column": {
        "create": "ALTER TABLE table_name ADD COLUMN column_name;",
        "rename": "ALTER TABLE table_name RENAME COLUMN column_name TO new_name;"
    }
}
```

Puis committez les changements avec le message: `feat: ajout référence mise à jour colonne`

Vous allez corriger la requête de création de table.

Pour cela, depuis la branche `main` créer une branche `fix/create-table-reference` et positionnez-vous dedans puis ajoutez des parenthèses après le nom de la table et commitez le changement avec le message `fix: correction référence create table` .

Revenez à la branche main puis faites un merge de la branche `fix/create-table-reference` avant de supprimer cette branche.

Dans la branche `feat/ajout-references-colonne` , on fait un **rebase**.

Positionnez-vous sur la branche main puis créez une branche `fix/ajout-references-rename-manquantes` où nous allons ajouter les sous-entrées `rename` de `database` et `table` :

```
{
    "database": {
        // ... ancien contenu
        "rename": "ALTER DATABASE database_name RENAME TO new_name;"
    },
    "table": {
        // ... ancien contenu
        "rename": "ALTER TABLE table_name RENAME TO new_name;"
    }
}
```

Committez les changements ensuite avec le message: `fix: ajout references rename manquantes` .

Ensuite on revient dans la branche `main` et on fait un merge avant de supprimer la branche `fix` .

Finalement, dans la branche `feat/ajout-references-colonne` on fait de nouveau un **rebase** en corrigeant les conflits éventuels ! (Il ne devrait pas y en avoir.)

## 6. Alors dernier défi

Une nouvelle branche `fix/ajout-references-lignes-manquantes` où il sera ajouté à `row` les sous-entrées `update` , `select` , `delete` dont les valeurs sont les requêtes SQL pour effectuer les actions correspondantes. Vous devriez avoir un contenu pareil:

```
{
    "row": {
        // ... ancien contenu
        "update": "UPDATE table_name SET column_name = new_value WHERE conditions;",
        "select": "SELECT columns_name FROM table_name;",
        "delete": "DELETE FROM table_name WHERE conditions;"
    }
}
```

Commit les changements sur cette branche avec le message `fix: ajout references lignes manquantes` et se positionner sur `main` afin de fusionner cette branche.

Se positionner ensuite sur la branche `feat/ajout-references-colonne` .

Ajouter une sous-entrée `drop` à `column` contenant la requête pour supprimer une colonne: `ALTER TABLE table_name DROP COLUMN column_name;` .

Commit les changements apportés par le message `feat: ajout reference suppression colonne` et vérifier avec `git log --oneline` .

Faire un rebase en utilisant la branche main et corriger les conflits si il y en a.

Se positionner sur la branche `main` et fusionner la branche `feat/ajout-references-colonne` à la branche courante.

Supprimer toutes les branches excepté main.

Afficher les logs.

Afficher les logs.

Vérifier l'état du repertoire courant.

# Notes de fin

Cet exercice est un bon début pour commencer à utiliser Git certes mais il vous faut maintenir le rythme en enchainant l'utilisation de ces notions dans vos projets personnels.

Et rétenez une chose, c'est en forgeant que l'on devient forgeron. Prenez les bonnes habitudes très vite dans vos projets et cela vous évitera de revenir en arrière très souvent.

Mamadou Moustapha Diallo, Lead GDSC IST Mamou - MARS 2024