TD 5 : Git et GitHub

BUT Science des Données – 2025-2026

Objectifs

- Créer un dépôt Git et effectuer des commits
- Publier un projet sur GitHub
- Travailler avec les branches et résoudre des conflits
- Appliquer les bonnes pratiques (messages de commit, `.gitignore`)

Exercice 1 : Premier dépôt Git (25 min)

Q1. Créez un dossier `mon_projet` et initialisez un dépôt Git :

```
mkdir mon_projet
cd mon_projet
git init
```

Q2. Vérifiez l'état du dépôt avec `git status`. Que voyez-vous ?

Q3. Créez un fichier `main.py` avec le contenu suivant, puis ajoutez-le à la staging area :

```
echo 'print("Hello Git!")' > main.py
git add main.py
```

Q4. Créez votre premier commit :

```
git commit -m "Ajouter le script principal"
```

Q5. Modifiez `main.py` pour ajouter une deuxième ligne :

```
print("Hello Git!")
print("Bienvenue dans le cours")
```

Utilisez `git diff` pour voir les modifications, puis commitez :

```
git diff
git add main.py
git commit -m "Ajouter un message de bienvenue"
```

Q6. Consultez l'historique avec `git log` puis avec `git log -oneline`. Quelle est la différence ?

Exercice 2 : Travailler avec GitHub (25 min)

Q1. Créez un nouveau repository sur GitHub :

1. Allez sur `github.com` et connectez-vous
2. Cliquez sur **New repository**
3. Nom : `mon_projet`
4. Laissez le repository **vide** (pas de README ni .gitignore)
5. Cliquez sur **Create repository**

Q2. Liez votre dépôt local au dépôt GitHub et poussez vos commits :

```
git remote add origin https://github.com/VOTRE_USER/mon_projet.git
git push -u origin main
```

Vérifiez sur GitHub que vos fichiers et commits sont visibles.

Q3. Simulez un collaborateur en clonant le dépôt dans un autre dossier :

```
cd ..
git clone https://github.com/VOTRE_USER/mon_projet.git
    mon_projet_clone
cd mon_projet_clone
git log --oneline
```

Q4. De retour dans `mon_projet`, créez un fichier `.gitignore` :

```
# Fichiers Python
__pycache__/
*.pyc

# Environnement
.env

# Sorties
output/
```

Commitez et poussez le `.gitignore` :

```
git add .gitignore
git commit -m "Ajouter␣le␣fichier␣.gitignore"
git push
```

Exercice 3 : Branches et merge (30 min)

Q1. Créez une nouvelle branche et basculez dessus :

```
git branch feature-calcul
git switch feature-calcul
```

Q2. Vérifiez que vous êtes sur la bonne branche avec `git branch`.

Q3. Créez un fichier `calcul.py` sur cette branche :

```
def addition(a, b):
    return a + b

def soustraction(a, b):
    return a - b
```

```
print(addition(3, 5))
```

Commitez ce fichier :

```
git add calcul.py
git commit -m "Ajouter les fonctions de calcul"
```

Q4. Revenez sur `main` et fusionnez la branche :

```
git switch main
git merge feature-calcul
```

Vérifiez que `calcul.py` est maintenant présent sur `main`.

Q5. Provoquez et résolvez un conflit :

a) Créez une branche `modification-a` et modifiez la ligne 7 de `calcul.py` :

```
git switch -c modification-a
```

Remplacez `print(addition(3, 5))` par `print(addition(10, 20))`, commitez.

b) Revenez sur `main`, créez `modification-b` et modifiez la même ligne :

```
git switch main
git switch -c modification-b
```

Remplacez `print(addition(3, 5))` par `print(addition(100, 200))`, commitez.

c) Fusionnez d'abord `modification-a` dans `main` (pas de conflit), puis fusionnez `modification-b` (conflit) :

```
git switch main
git merge modification-a
git merge modification-b
```

d) Ouvrez `calcul.py`, résolvez le conflit en choisissant une version, supprimez les marqueurs, puis :

```
git add calcul.py
git commit -m "Résoudre le conflit sur calcul.py"
```

Exercice 4 : Bonnes pratiques (10 min)

Q1. Parmi les messages de commit suivants, lesquels sont de bons messages ? Justifiez.

1. `fix`
2. `Corriger le bug de connexion à la base de données`
3. `modifications`
4. `Ajouter la validation des emails dans le formulaire`
5. `ça marche`

Q2. Affichez l'historique complet de votre projet avec le graphe des branches :

```
git log --oneline --graph --all
```

Identifiez les points de merge dans le graphe.