TP : Systèmes de Gestion de Versions avec GIT et GitHub

Travail en binôme

Année universitaire 2025-2026

Table des matières

1 Objectifs

L'objectif de cet atelier est d'apprendre à utiliser Git pour :

— Versionner vos projets (PFA, PFE, etc.)

— Collaborer efficacement en équipe

— Gérer les modifications et résoudre les conflits

Vous allez d'abord travailler sur un dépôt local (clone d'un dépôt partagé), puis sur un dépôt partagé distant (GitHub).

Important

Ce TP doit être réalisé en **binôme** avec deux ordinateurs distincts pour simuler un véritable travail collaboratif.

2 Préparation

2.1 Création d'un compte GitHub

1. Rendez-vous sur `https://github.com`
2. Créez un compte gratuit si vous n'en avez pas déjà un
3. Confirmez votre adresse e-mail via le courriel de confirmation
4. Notez votre nom d'utilisateur GitHub

2.2 Installation de Git

2.2.1 Sur Windows

1. Téléchargez Git : `http://git-scm.com/download/win`
2. Installez en gardant les options par défaut
3. Lancez **Git Bash** (clic droit sur un dossier → "Git Bash Here")

2.2.2 Sur Linux/Ubuntu

```
sudo apt-get update
sudo apt-get install git
```

2.2.3 Sur macOS

1. Téléchargez depuis `https://git-scm.com/download/mac`
2. Ou installez via Homebrew : `brew install git`

2.3 Vérification de l'installation

Dans un terminal (ou Git Bash), vérifiez l'installation :

```
git --version
```

3 Configuration de Git

3.1 Configuration globale de l'utilisateur

Configurez votre identité Git (à faire sur les deux PC) :

```
git config --global user.name "Votre Nom Prenom"
git config --global user.email "votre.email@example.com"
```

Vérifiez votre configuration :

```
git config --list
```

4 Authentification GitHub

Important

Choisissez UNE des deux méthodes d'authentification suivantes. La méthode par Token est recommandée pour les débutants.

4.1 Méthode 1 : Authentification par Token (Recommandée)

4.1.1 Création du Personal Access Token

1. Connectez-vous à GitHub
2. Cliquez sur votre photo de profil (en haut à droite) → **Settings**
3. Dans le menu de gauche, allez tout en bas : **Developer settings**
4. Cliquez sur **Personal access tokens** → **Tokens (classic)**
5. Cliquez sur **Generate new token** → **Generate new token (classic)**
6. Donnez un nom descriptif (ex : "TP_Git_2025")
7. Définissez une expiration (ex : 90 jours)
8. Cochez les permissions suivantes :
   - `repo` (toutes les sous-options)
   - `workflow`
   - `write:packages`
   - `read:packages`
9. Cliquez sur **Generate token**
10. **IMPORTANT :** Copiez immédiatement le token généré et sauvegardez-le dans un fichier texte sécurisé (vous ne pourrez plus le revoir)

Astuce

Conservez votre token dans un endroit sûr. Vous en aurez besoin à chaque opération `push` ou `pull`.

4.1.2 Utilisation du Token

Lors de votre premier `push` ou `clone`, Git vous demandera vos identifiants :

- **Username :** Votre nom d'utilisateur GitHub
- **Password :** Collez votre Personal Access Token (PAS votre mot de passe GitHub)

4.1.3 Sauvegarder le Token de manière permanente

Pour ne pas avoir à retaper le token à chaque fois, configurez le gestionnaire de credentials Git :

```
# Sauvegarder le token de maniere permanente
git config --global credential.helper store
```

Comment ça fonctionne :

1. Vous exécutez cette commande **une seule fois**
2. La prochaine fois que vous faites `git push` ou `git pull`, Git vous demandera :
   - Username : `votre-username-github`
   - Password : `votre-token` (ghp_xxxxxxxxxxxx)
3. Git sauvegarde automatiquement ces identifiants dans le fichier `~/.git-credentials`
4. **À partir de là, vous n'aurez plus jamais besoin de retaper le token !**

> **Astuce**
>
> Comme avec SSH, une fois configuré, l'authentification devient transparente. Vous travaillez normalement sans vous soucier du token.

4.2 Méthode 2 : Authentification par SSH (Avancée)

4.2.1 Génération de la clé SSH

1. Créez le répertoire SSH (si nécessaire) :

```
mkdir -p ~/.ssh
cd ~/.ssh
```

2. Générez une paire de clés SSH :

```
ssh-keygen -t ed25519 -C "votre.email@example.com"
```

> **Astuce**
>
> Si votre système ne supporte pas ed25519, utilisez : `ssh-keygen -t rsa -b 4096 -C "votre.email@example.com"`

3. Appuyez sur **Entrée** pour accepter l'emplacement par défaut
4. Appuyez sur **Entrée** deux fois pour ne pas mettre de passphrase (ou définissez-en une)

4.2.2 Récupération de la clé publique

```
# Sur Linux/Mac
cat ~/.ssh/id_ed25519.pub

# Sur Windows (Git Bash)
cat ~/.ssh/id_ed25519.pub
# Ou
notepad ~/.ssh/id_ed25519.pub
```

Copiez tout le contenu de la clé publique (commence par `ssh-ed25519` ou `ssh-rsa`).

4.2.3 Ajout de la clé sur GitHub

1. Sur GitHub, allez dans **Settings** (photo de profil → Settings)
2. Dans le menu de gauche : **SSH and GPG keys**
3. Cliquez sur **New SSH key**
4. **Title :** Donnez un nom descriptif (ex : "PC_TP_Git")
5. **Key type :** Sélectionnez "Authentication Key"
6. **Key :** Collez votre clé publique
7. Cliquez sur **Add SSH key**

4.2.4 Test de la connexion SSH

```
ssh -T git@github.com
```

Vous devriez voir : `Hi username! You've successfully authenticated...`

5 Travaux Pratiques - PC 1 (Étudiant A)

5.1 Création du projet local

1. Créez un dossier pour votre projet :

```
mkdir projet-git-tp
cd projet-git-tp
```

2. Initialisez le dépôt Git :

```
git init
```

3. Créez un fichier README :

```
echo "# Projet TP Git" > README.md
echo "Travail realise par [Nom Etudiant A] et [Nom Etudiant B]" >>
    README.md
```

4. Ajoutez le fichier au suivi Git :

```
git add README.md
```

5. Effectuez votre premier commit :

```
git commit -m "Premier commit : ajout du README"
```

5.2 Création du dépôt distant sur GitHub

1. Sur GitHub, cliquez sur le + en haut à droite → **New repository**
2. **Repository name :** `projet-git-tp`
3. **Description :** "TP Git - Gestion de versions"
4. Laissez en **Public** (ou Private selon préférence)
5. **NE cochez PAS** "Initialize this repository with a README"
6. Cliquez sur **Create repository**

5.3 Liaison du dépôt local avec le dépôt distant

Si vous utilisez HTTPS (Token) :

```
git remote add origin https://github.com/VotreUsername/projet-git-tp.git
git branch -M main
git push -u origin main
```

Si vous utilisez SSH :

```
git remote add origin git@github.com:VotreUsername/projet-git-tp.git
git branch -M main
git push -u origin main
```

Astuce

Lors du premier `push` avec HTTPS, entrez votre username GitHub et votre **token** (pas votre mot de passe).

5.4 Ajout d'un collaborateur

1. Sur la page GitHub du projet, cliquez sur **Settings**
2. Dans le menu de gauche : **Collaborators**
3. Cliquez sur **Add people**
4. Recherchez l'Étudiant B par son username ou email
5. Sélectionnez le rôle **Write**
6. Envoyez l'invitation

Important

L'Étudiant B doit accepter l'invitation reçue par email ou sur GitHub avant de continuer.

6 Travaux Pratiques - PC 2 (Étudiant B)

6.1 Acceptation de l'invitation

1. Vérifiez vos emails ou les notifications GitHub
2. Acceptez l'invitation à collaborer sur le projet

6.2 Clonage du dépôt

Si vous utilisez HTTPS (Token) :

```
git clone https://github.com/UsernameEtudiantA/projet-git-tp.git
cd projet-git-tp
```

Si vous utilisez SSH :

```
git clone git@github.com:UsernameEtudiantA/projet-git-tp.git
cd projet-git-tp
```

6.3 Vérification du dépôt

```
# Afficher le contenu
ls -la

# Afficher l'historique
git log

# Afficher le statut
git status
```

7 Cycle de travail collaboratif

7.1 PC 2 : Création et modification de fichiers

1. Créez un fichier avec votre prénom :

```
echo "Je suis l'Etudiant B" > etudiantB.txt
echo "Date : $(date)" >> etudiantB.txt
```

2. Vérifiez le statut :

```
git status
```

3. Ajoutez le fichier au suivi :

```
git add etudiantB.txt
```

4. Commitez la modification :

```
git commit -m "Ajout du fichier etudiantB.txt"
```

5. Envoyez vers le dépôt distant :

```
git push origin main
```

7.2 PC 1 : Récupération des modifications

1. Récupérez les dernières modifications :

```
git pull origin main
```

2. Vérifiez la présence du nouveau fichier :

```
ls -la
cat etudiantB.txt
```

3. Créez votre propre fichier :

```
echo "Je suis l'Etudiant A" > etudiantA.txt
echo "Mon role : Chef de projet" >> etudiantA.txt
```

4. Ajoutez, commitez et poussez :

```
git add etudiantA.txt
git commit -m "Ajout du fichier etudiantA.txt"
git push origin main
```

7.3 PC 2 : Synchronisation

```
git pull origin main
ls -la
```

8 Gestion des branches

8.1 PC 1 : Création d'une branche de développement

```
# Creer une nouvelle branche
git branch developpement

# Basculer sur cette branche
git checkout developpement
# Ou en une seule commande
# git checkout -b developpement

# Verifier la branche active
git branch
```

8.2 PC 1 : Travail sur la branche

```
# Creer un nouveau fichier
echo "Fonctionnalite en cours de developpement" > fonctionnalite.txt

# Ajouter et commiter
git add fonctionnalite.txt
git commit -m "Ajout d'une nouvelle fonctionnalite"

# Pousser la branche vers GitHub
git push origin developpement
```

8.3 PC 2 : Récupération de la branche

```
# Recuperer toutes les branches
git fetch origin

# Lister toutes les branches (locales et distantes)
git branch -a

# Basculer sur la branche developpement
git checkout developpement

# Verifier le contenu
ls -la
```

8.4 Fusion de branches (PC 1)

```
# Retourner sur la branche principale
git checkout main

# Fusionner la branche developpement
```

```
git merge developpement

# Pousser les modifications
git push origin main
```

9 Gestion des conflits

9.1 Simulation d'un conflit

Sur PC 1 et PC 2 simultanément (sans pull) :
PC 1 :

```
echo "Modification par l'Etudiant A" > conflit.txt
git add conflit.txt
git commit -m "PC1 : Ajout du fichier conflit"
git push origin main
```

PC 2 : (sans faire de pull avant)

```
echo "Modification par l'Etudiant B" > conflit.txt
git add conflit.txt
git commit -m "PC2 : Ajout du fichier conflit"
git push origin main
```

9.2 Résolution du conflit sur PC 2

Git refusera le push. Vous devez d'abord récupérer les modifications :

```
git pull origin main
```

Git indiquera un conflit. Ouvrez le fichier `conflit.txt` :

```
<<<<<<< HEAD
Modification par l'Etudiant B
=======
Modification par l'Etudiant A
>>>>>>> [hash du commit]
```

Éditez le fichier pour résoudre le conflit :

```
# Editez conflit.txt et gardez les deux modifications
echo "Modification par l'Etudiant A" > conflit.txt
echo "Modification par l'Etudiant B" >> conflit.txt

# Marquez le conflit comme resolu
git add conflit.txt
git commit -m "Resolution du conflit sur conflit.txt"
git push origin main
```

10 Commandes utiles

10.1 Visualisation de l'historique

```
# Historique complet
git log

```

```
# Historique simplifie
git log --oneline

# Historique graphique
git log --graph --oneline --all

# Historique avec details
git log --stat
```

10.2 Gestion des modifications

```
# Voir les differences non commitees
git diff

# Voir les differences entre deux commits
git diff commit1 commit2

# Annuler les modifications locales (non commitees)
git checkout -- nomfichier.txt

# Annuler le dernier commit (garde les modifications)
git reset --soft HEAD~1

# Annuler le dernier commit (supprime les modifications)
git reset --hard HEAD~1
```

10.3 Informations sur le dépôt

```
# Voir la configuration
git config --list

# Voir les remotes
git remote -v

# Voir toutes les branches
git branch -a

# Voir le statut detaille
git status -v
```

11 Exercices pratiques

11.1 Exercice 1 : Créer une structure de projet

1. **PC 1 :** Créez l'arborescence suivante :

```
mkdir -p src docs tests
touch src/main.py docs/documentation.md tests/test_main.py
```

2. Ajoutez du contenu à chaque fichier
3. Commitez et poussez les modifications
4. **PC 2 :** Récupérez les modifications et vérifiez la structure

11.2 Exercice 2 : Travailler sur des branches parallèles

1. **PC 1 :** Créez une branche `feature-login`
2. **PC 2 :** Créez une branche `feature-dashboard`
3. Chacun développe sa fonctionnalité sur sa branche
4. Fusionnez les deux branches dans `main`

11.3 Exercice 3 : Simuler et résoudre un conflit complexe

1. Créez un fichier `config.txt` avec plusieurs lignes
2. Les deux PC modifient les mêmes lignes différemment
3. Commitez sans pull
4. Résolvez le conflit en conservant les meilleures parties de chaque version

12 Bonnes pratiques

1. **Commitez souvent** avec des messages clairs et descriptifs
2. **Faites des pull régulièrement** avant de commencer à travailler
3. **Utilisez des branches** pour les nouvelles fonctionnalités
4. **Écrivez des messages de commit explicites** :
   - Bon : "Ajout de la fonction de connexion utilisateur"
   - Mauvais : "update" ou "fix"
5. **Vérifiez le statut** avant de commiter : `git status`
6. **Ne commitez jamais** :
   - Des mots de passe ou tokens
   - Des fichiers compilés (.o, .class, .pyc)
   - Des dossiers de dépendances (node_modules, venv)
7. **Utilisez un fichier .gitignore** pour exclure les fichiers inutiles

13 Fichier .gitignore

Créez un fichier `.gitignore` à la racine du projet :

```
# Fichiers Python
*.pyc
__pycache__/
venv/
.env

# Fichiers Java
*.class
target/

# IDE
.vscode/
.idea/
*.swp

# OS
.DS_Store
```

```
Thumbs.db

# Logs
*.log
```

14 Ressources supplémentaires

— Documentation officielle Git : `https://git-scm.com/doc`

— GitHub Guides : `https://guides.github.com`

— Aide GitHub : `https://docs.github.com`

— Tutoriel interactif : `https://learngitbranching.js.org`

15 Conclusion

Ce TP vous a permis de découvrir les bases de Git et GitHub pour le travail collaboratif. Vous savez maintenant :

— Configurer Git et s'authentifier sur GitHub

— Créer et cloner des dépôts

— Effectuer le cycle complet : add, commit, push, pull

— Travailler avec des branches

— Gérer et résoudre des conflits

Continuez à pratiquer ! Git deviendra rapidement un outil indispensable dans votre carrière de développeur.