Formation sur Git et GitHub

M2

Elise MARTIN
elise.martin@etu.univ-poitiers.fr

Blandine SERRES
blandine.serres@etu.univ-poitiers.fr

1. Introduction à Git

Qu'est-ce que Git ?

➔ Système de contrôle de version distribué, créé par Linus Torvalds en 2005.
➔ Permet de suivre les modifications apportées aux fichiers au cours du temps = versionning

Avantages du Versioning :

- **Collaboration :** Plusieurs développeurs peuvent travailler simultanément sans interférer.
- **Traçabilité :** Historique complet des modifications.
- **Sécurité :** Possibilité de restaurer des versions précédentes en cas d'erreur.
- **Documentation :** Historique des commits détaillant les modifications.

Installation de Git :

Selon votre système d'exploitation, l'installation de Git diffère **:**

- Windows : `https://git-scm.com/downloads/win`
- Linux/Unix (Debian/Ubuntu) : `sudo apt-get install git`
- macOS : `brew install git` (Homebrew requis)

NB : Si vous n'avez pas Homebrew, exécutez la commande :

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

Documentation pour plus de choix :

- Windows : https://git-scm.com/downloads/win
- Linux/Unix : https://git-scm.com/downloads/linux
- macOS : https://git-scm.com/downloads/mac

Configuration Initiale :

- Nom d'utilisateur : `git config --global user.name "UserName"`
- Email : `git config --global user.email "user.name01@xx.com"`
- Branche principale : `git config --global init.defaultBranch master`

NB : Vérifier la configuration faite : `git config --list`

Commandes de Base :

—— Créer un répertoire qui stockera le projet git, puis se déplacer dedans ——
—— Ouvrir un terminal ——

- Créer un dépôt local Git dans le répertoire créé : `git init`

NB : Cela crée un sous-dossier, nommé `.git` contenant les informations nécessaires au suivi des modifications du projet et son historique

—— Modifier le projet : ajout de fichiers ——

- Ajouter des fichiers : `git add nomFichier`

NB : Pour ajouter tous les fichiers modifiés en une seule commande : `git add .`

- Enregistrer les modifications dans le dépôt local : `git commit -m "commentaire"`

NB :

- Le commentaire qui suit -m est une description des modifications apportées
- Restaurer des fichiers à partir de l'historique de dépôt : `git checkout numéro`
- Retourner à la dernière version : `git switch -`
- Annuler un commit avec la commande : `git reset`
- Consulter l'historique des modifications : `git log`

Le schéma suivant présente les principales commandes Git :

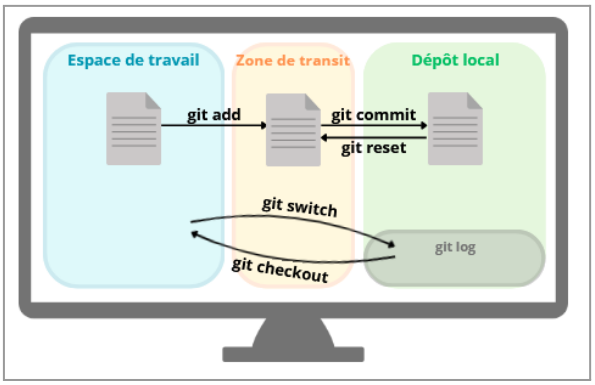

Schéma bilan de l'utilisation de Git en local

2. Introduction à GitHub & Gestion d'un dépôt

Qu'est-ce que GitHub ?

➔ Plateforme de développement collaboratif basée sur Git.
➔ Permet d'héberger des dépôts Git et offre des outils pour la gestion des projets.

Avantages de GitHub :

- **Hébergement de dépôts :** Stockage sécurisé sur le cloud.
- **Collaboration :** Travail simultané avec des révisions, commentaires et approbations de modifications.
- **Documentation :** Inclusion de fichiers README pour faciliter la prise en main.
- **Sécurité :** Contrôle d'accès granulaires pour protéger les projets.
- **Intégration :** Compatibilité avec divers outils et services de développement (ex : BlueJ).

Commandes de Base :

1) Création d'un dépôt (repository) et partage

- Sur GitHub, cliquer sur le bouton "New repository" pour créer un nouveau dépôt
- Remplir le nom du dépôt, sa description et choisir son accès (public ou privé)
- Cocher l'option pour initialiser le dépôt avec un fichier README si souhaité
- Cliquer sur "Create repository"
- Dans le menu "Settings", aller dans l'onglet "Collaborators" et ajouter un membre

2) Clonage du dépôt et évolution du projet local GitHub

—— Créer un répertoire qui stockera le projet git, puis se déplacer dedans ——
—— Aller sur la page du projet GitHub créé ——

- Importer un fichier dans le projet (si absence de README)
- Cliquer sur le bouton "Code" et copier l'URL HTTPS

—— Ouvrir un terminal ——

- Exécuter la commande : `git clone URL_DU_DEPOT_GIT`

> NB : La commande `git clone URL_DU_DEPOT_GIT` implique d'avoir au moins 1 fichier de créé dans le répertoire

—— Ajouter des fichiers, modifier ceux existants… ——

- Suivre l'état des modifications du dépôt (en cours de suivi ou non) : `git status`
- Ajouter des fichiers : `git remote add origin URL_DU_DEPOT_GIT`

> NB : La commande `git remote add origin URL_DU_DEPOT_GIT` n'est à faire qu'une seule fois pour chaque dépôt local. Cette commande va permettre de créer un lien du dépôt local vers le dépôt distant.
> Les fois suivantes, la commande `git add .` suffira

- Suivre l'état des modifications du dépôt (en cours de suivi ou non) : `git status`

> NB : Pour mieux suivre l'état de son projet (les modifications prêtes à être enregistrées), il est conseillé de faire un `git status` avant et après le `git add .`

- Enregistrer les modifications dans le dépôt local : `git commit -m "commentaire"`

> NB :
> - Le commentaire qui suit `-m` est une description des modifications apportées
> - Restaurer des fichiers à partir de l'historique de dépôt : `git checkout`
> - Retourner à la dernière version : `git switch -`
> - Annuler un commit avec la commande : `git reset`
> - Consulter l'historique des modifications : `git log`

3) Partage des modifications locales (personnel) vers le dépôt distant (partagé)

- Transférer les modifications du dépôt local vers le distant : `git push -u origin main`

> NB : La commande `git push -u origin main` n'est à faire qu'une seule fois pour chaque dépôt local. Cette commande va permettre de pousser les commits du dépôt local vers le dépôt distant.
> Les fois suivantes, la commande `git push` suffira

4) Récupérer des modifications du dépôt distant

- Récupérer les nouveautés du projet : `git pull`

> NB : Le `git pull` doit être effectué avant de commencer à travailler sur une nouvelle tâche.
> ➢ Pour récupérer spécifiquement la branche "main" : `git pull origin main`

Le schéma suivant présente les principales commandes Git :

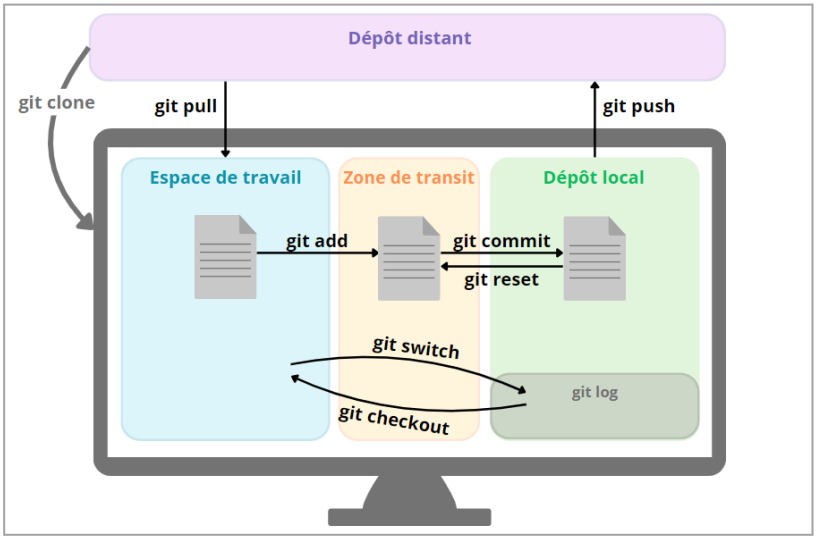

Schéma bilan de l'utilisation de Git en local

3. Gestion des conflits

Lorsque vous travaillez sur des projets collaboratifs avec Git, les conflits peuvent survenir si plusieurs développeurs modifient le même code. Voici un exemple :

Code d'origine :

```
def calc_array(arr):
    total = 0
    for i in range(len(arr)):
        total += arr[i]
    return total
```

Modifications développeur 1 :

```
def sum_array(arr):
    total = 0
    for i in range(len(arr)):
        total += arr[i]
    print("Total : ", total)
      # Affichage du total
    return total
```

Modifications développeur 2 :

```
def sum_array(arr):
    total = sum(arr)  # Utilisation
de la fonction sum()
    return total
```

Conflit :

```
def sum_array(arr):
<<<<<<< HEAD
    total = 0
    for i in range(len(arr)):
        if isinstance(arr[i], (int, float)):  # Vérification de type
            total += arr[i]
    return total
=======
    total = sum(arr)  # Utilisation de la fonction sum() intégrée
    return total
>>>>>>>
```

La gestion de ces conflits est une partie essentielle de la collaboration sur des projets utilisant Git. Elle vise à éviter au maximum les problèmes, et c'est pourquoi, il y a des points à respecter :

1. **Répartir les tâches clairement :**
    - Assigner des tâches spécifiques à chaque membre de l'équipe pour ne pas travailler simultanément sur les mêmes classes et fichiers.
    - Éviter de modifier du code qui ne vous est pas attribué même cela peut être tentant.
    - Planifier des réunions régulières pour discuter de l'avancement des tâches.

2. **Faire des mises à jour fréquentes :**
    - Récupérer les modifications du dépôt distant avant chaque nouveau travail avec `git pull`.
    - Intégrer et pousser les modifications vers le dépôt distant, de façon régulière, avec `git commit -m "commentaire"` et `git push`.

3. **Ecrire son prénom dans le message de commit :**
    - Cela permet d'indiquer qui a travaillé sur le fichier, et ainsi, de savoir à qui s'adresser en cas de questionnement ou de problème.

4. **Utiliser des branches pour isoler le travail de chaque développeur :**
    - Chaque membre peut travailler sur sa propre branche et fusionner les changements dans la branche principale une fois les modifications terminées et testées.

4. Bonnes pratiques sur Git

Pour éviter de rencontrer des problèmes lors d'un projet, des bonnes pratiques peuvent être suivies, assurant une collaboration fluide.

Utiliser un fichier .gitignore :

1) Fonctionnalités du fichier .gitignore :

- Ignorer des fichiers et répertoires spécifiés lors des commit et push.
- Éviter les conflits en ignorant les fichiers créés lors de la compilation (ex : .class).
- Assurer la sécurité et la propreté du dépôt en excluant les fichiers temporaires, les dépendances et les informations sensibles, qui sont propres à chaque dépôt local (ex : *.log, *.tmp)

2) Création du fichier .gitignore :

Ce fichier `.gitignore` peut être ajouté lors de la création du projet sur GitHub, avec l'utilisation d'un template :

Créer un .gitignore à l'aide de templates lors de la création du projet GitHub

Toutefois, si le `.gitignore` n'a pas été créé, il peut être ajouté manuellement.

- ❖ Créer un nouveau fichier texte et le renommer `.gitignore`.
- ❖ Ouvrir le fichier et ajouter les extensions à ignorer à l'intérieur.

NB : Voici 3 exemples d'extensions à ajouter :

```
# Ignorer les fichiers de classe Java compilés
*.class

# Ignorer les fichiers journaux
*.log

# Ignorer les fichiers temporaires
*.tmp
```

Utiliser un fichier README.md :

1) Importance du README.md :

- Première impression sur le projet : Il s'agit du premier fichier que les nouveaux contributeurs et utilisateurs voient dans un dépôt.
- Possède les informations essentielles : Son contenu explique pourquoi le projet est utile, ce qu'il est possible de faire avec et comment l'utiliser.

2) Points clés à inclure dans un README.md :

- Titre du projet : Le nom du projet.
- Description : Une brève description de ce que fait le projet.
- Installation : Instructions sur la façon d'installer et configurer le projet.
- Utilisation : Instructions pour prendre en main et utiliser le projet.

3) Exemple bref d'un fichier README.md :

```
# MonProjetJava

## Description
Ce projet est une application Java développée sous BlueJ pour gérer les
tâches quotidiennes.

## Installation
1. Clonez le dépôt (sur un terminal) :
  git clone https://github.com/votre-utilisateur/MonProjetJava.git

2. Accédez au répertoire du projet (sur un terminal) :
  cd MonProjetJava

3. Ouvrez le projet dans VSCode :
  - Lancez VSCode.
  - Ouvrez le projet en sélectionnant le répertoire cloné.

## Utilisation
Pour exécuter l'application dans VSCode :
- Ouvrez la classe principale et cliquez sur "Start Debugging".
- Cliquez sur "Run without Debugging" pour exécuter l'application.
```

5. GitHub et BlueJ

1) Création d'un token

- Sur GitHub, aller dans :
  `Settings` > `Developer settings` > `Personal access tokens`
- Cliquer sur :
  `Tokens (classic)` > `Generate new token` > `Generate new token (classic)`
- Donner un nom et choisir un délai d'expiration
- Cocher les cases `repo` et `admin:repo_hook`

NB :
- La case `repo` permet d'accorder un accès complet en lecture et en écriture aux dépôts
- La case `admin:repo_hook` permet de permet de gérer les actions automatiques dans le dépôt, comme les commits ou les pull

- Cliquer sur `Generate token`

2) Création d'un dépôt (repository) et partage

- Sur GitHub, cliquer sur le bouton "New repository" pour créer un nouveau dépôt
- Remplir le nom du dépôt, sa description et choisir son accès (public ou privé)
- Cocher l'option pour initialiser le dépôt avec un fichier README si souhaité
- Ajouter le `.gitignore` avec le template "Java"
- Cliquer sur "Create repository"
- Dans le menu "Settings", aller dans l'onglet "Collaborators" et ajouter un membre

3) Ouvrir un Projet GitHub sur BlueJ et évolution du projet local

- Sur BlueJ, aller dans :
  `Outils` > `Equipes` > `Extraction Projet`
- Accéder au dépôt nouvellement créé sur GitHub
- Cliquer sur le bouton "Code" et copier l'URL HTTPS
- Coller l'URL dans la barre `Repository URI`
- Entrer son nom, email, pseudo (utilisateur) et son token (mot de passe)
- Cliquer sur `Tester connexion…` puis `OK`
- Nommer le projet et modifier la localisation pour le cloner.

NB : Vous n'êtes pas obligé de saisir la branche, cela prendra `master` par défaut

—— Ajouter des fichiers, modifier ceux existants… ——

4) Partage des modifications locales (personnel) vers le dépôt distant (partagé)

- Allez dans :
  `Outils` > `Equipes` > `Commit/Push to Repository…`
- Ecrire un commentaire (équivalent de la commande `git commit -m "commentaire"`)
- Appuyer sur `Commit`
- Appuyer sur `Push` et fermer

5) Récupérer des modifications du dépôt distant

- Allez dans :
  `Outils` > `Equipes` > `Mise à jour depuis dépôt`
- Cliquer sur `Mise à jour` pour récupérer la classe créée par votre binôme