AVIGNON UNIVERSITÉ

Rapport TP1
Fork them all

Étudiant
Michel Marie LAMAH

19 février 2023

Master Informatique
Intelligence Artificielle

UE Génie logiciel avancé
ECUE Techniques de test

Responsable
Daniel SALAS
Emmanuel FERREIRA

UFR
SCIENCES
TECHNOLOGIES
SANTÉ

AVIGNON UNIVERSITÉ
CERI

CENTRE
D'**ENSEIGNEMENT**
ET DE **RECHERCHE**
EN **INFORMATIQUE**
ceri.univ-avignon.fr

Sommaire

1 Introduction

Dans cet TP[1], il sera question de prendre en main le gestionnaire de version Git [1] mais aussi débuter avec les tests en JAVA en utilisant un projet MAVEN [2].

Pour réaliser ce travail, j'utiliserai :

- **IDE** : Intellij IDEA [3]
- **Dépôt de repository Git** : Github [4]

Toutes les modifications réalisées ici se trouvent sur la branche **TP1** de mon repository [2].

2 Épisode 1 - Récupération de l'API

Pour forker tous les fichiers des tps que nous auront à faire, il suffit de nous rendre sur ce lien

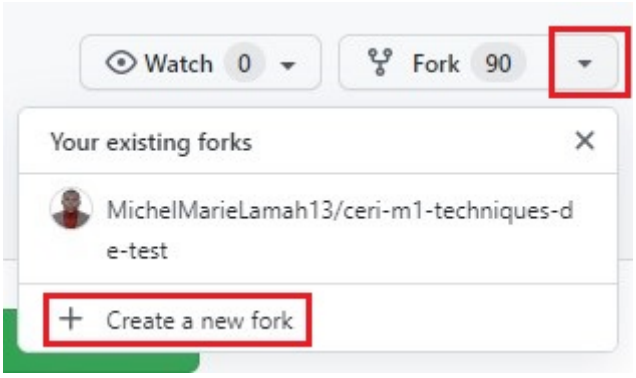

Figure 1. Création Fork

Une fois le projet forké dans notre compte github, pour le cloner, dans Intellij, on le fait facilement en procédant comme suit :

Onglet **File** → **New** → **Project From Version Control**

Dans la fenêtre qui s'affiche, il nous faut ajouter le lien github du repo à cloner. Pour trouver le lien, il nous suffit de nous rendre sur le repository et d'effectuer les étapes suivantes :

1. https://git-scm.com/
2. https://maven.apache.org/
3. https://www.jetbrains.com/fr-fr/idea/
4. https://github.com/

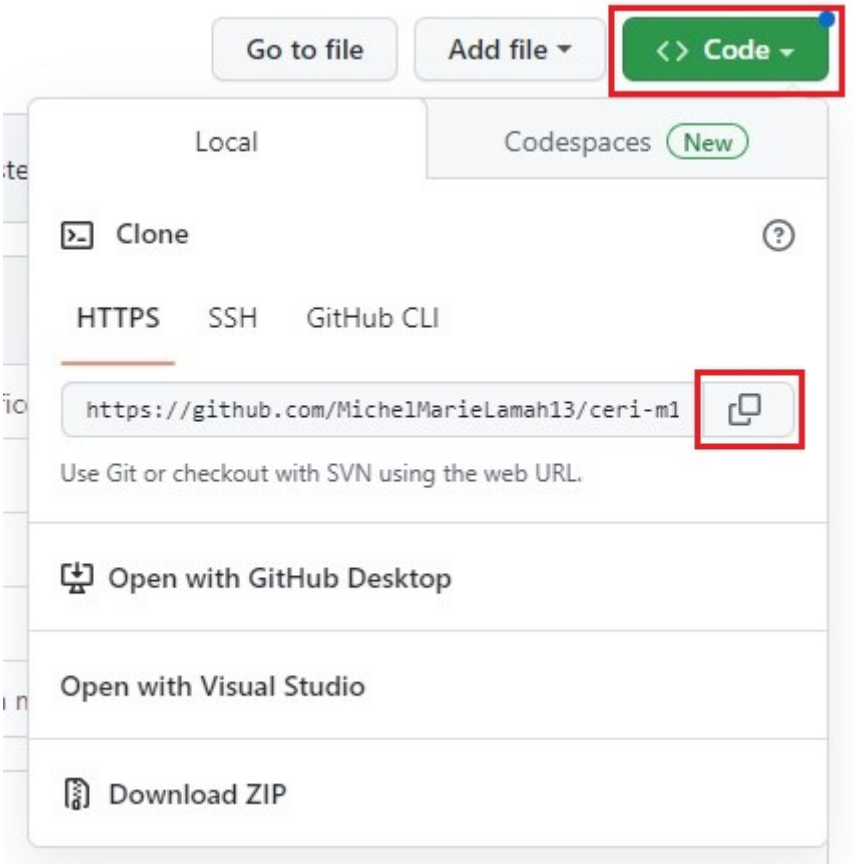

Figure 2. Copier lien du github

Une fois le lien copier, on l'ajoute comme indiqué ci-dessous :

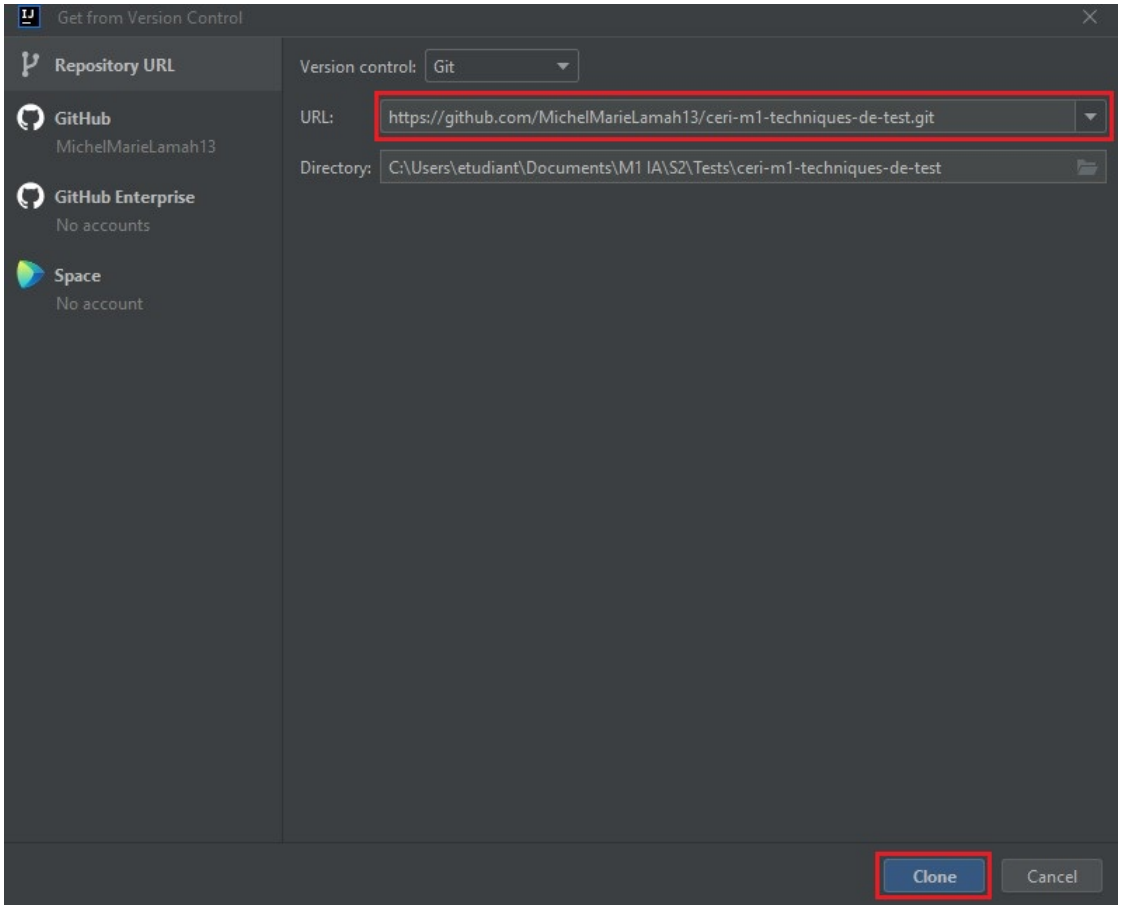

Figure 3. Coller lien du github dans intellij

3 Épisode 2 - Création du projet MAVEN

Une fois le projet cloné, on obtient la structure ci-dessous :

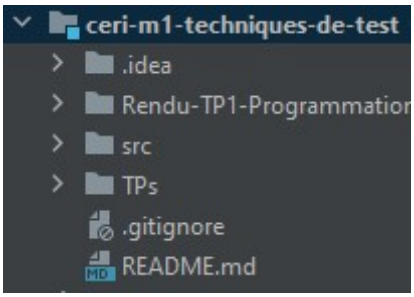

Figure 4. Structure du projet cloné

Dans le projet cloné, on remarque aisément qu'il n'y a pas fichier **pom.xml**, qui permet de reconnaître un **projet Maven**.

Du coup on doit le transformer en un projet Maven et pour cela on peut soit créer manuellement un fichier pom.xml ou soit utiliser l'IDE pour le faire.

Dans notre cas on va utiliser l'IDE et pour cela, on procède comme suit :

Dans l'arborescence du projet

Clic droit sur le nom du projet → **Add Framework Support** → **Maven**

En faisant cela, l'IDE crée le repertoire **\src\test** puis un fichier **pom.xml**

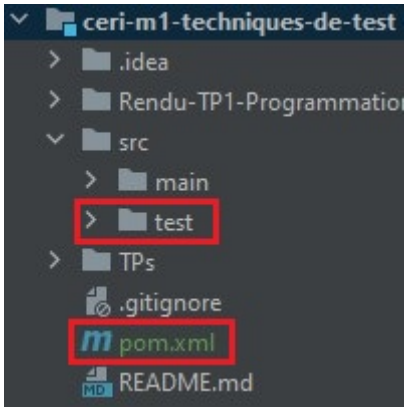

Figure 5. Structure du projet maven obtenu

Toutefois le fichier pom.xml est minimal

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
            http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>groupId</groupId>
    <artifactId>ceri-m1-techniques-de-test</artifactId>
    <version>1.0-SNAPSHOT</version>


</project>
```

Nous allons modifier ce fichier pour y ajouter d'autres informations.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
            http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <!-- Basics information -->
    <groupId>org.univavignon</groupId>
    <artifactId>ceri-m1-techniques-de-test</artifactId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13.2</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.mockito</groupId>
            <artifactId>mockito-core</artifactId>
            <version>5.1.1</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <!-- Build Settings-->
```

```
    <properties>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <!-- Project Meta Data-->
    <name>TP Techniques de Test</name>
    <description>Mise en pratique des tests en Java</description>
    <url>https://univ-avignon.fr/</url>
    <developers>
        <developer>
            <id>MichelMarieLamah13</id>
            <name>Michel Marie LAMAH</name>
            <email>michel-marie.lamah2@alumni.univ-avignon.fr</email>
            <properties>
                <owner>true</owner>
            </properties>
        </developer>
    </developers>
    <contributors>
        <contributor>
            <name>Daniel SALAS</name>
            <email>daniel.salas@univ-avignon.fr</email>
            <properties>
                <teacher>true</teacher>
            </properties>
        </contributor>
    </contributors>

</project>
```

Dans le fichier précédent, nous avons ajouté les dépendances **JUnit** et **Mockito**, quelques méta-données du projet mais aussi la version du JDK qui sera utilisé pour la compilation du projet.

4 Épisode 3 - Synchronisation avec Git

Bien avant d'intégrer nos modifications dans GitHub, nous allons essayé de compiler le projet pour voir si tout fonctionne correctement.

Pour cela, nous utilisons la commande ci-dessous :

```
mvn compile
```

Et comme sortie, nous obtenons :

```
[INFO] ------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------
[INFO] Total time:  0.790 s
[INFO] Finished at: 2023-02-19T04:41:38+01:00
[INFO] ------------------------------------
```

Figure 6. Le résultat obtenu

Maintenant que la compilation est effectuée avec succès, nous allons alors faire le

commit.

Dans Intellij, on procède comme suit :

Onglet **Git** → **Commit**

5 Conclusion

Dans cet tp, il était question de prendre en main Maven ainsi que le gestionnaire de version Github.

On remarque aisément que Maven avec ces commandes **mvn**, nous permet largement de réaliser plusieurs tâches, de la compilation en passant par le test jusqu'au déploiement.

Et Github à travers Git nous permet de faire plusieurs versions de notre projet, par exemple dans ce tp, j'ai effectué tout mon travail en utilisant une nouvelle branche pour ne pas modifier la branche origin.

Références

[1] Daniel SALAS. *TP1 : Fork them all*. Université Avignon. 2023. url : https://github.com/Youkoulanda/ceri-m1-techniques-de-test/blob/master/TPs/TP1.md.

[2] LAMAH Michel Marie. *TP1 : Fork them all*. Université Avignon. 2023. url : https://github.com/MichelMarieLamah13/ceri-m1-techniques-de-test/tree/TP01.