amU Faculté des sciences Aix Marseille Université

Année 2024-25

Domaine Sciences et Technologies
Master informatique
Génie logiciel : TP 1
Codes UE : SINA09EL
Mars Rover (1/3)

1 Introduction

Le but de ce TP est de programmer un simulateur pour des *rovers* de la NASA explorant Mars.

L'objectif pédagogique du projet est de vous faire travailler sur les outils et les bonnes pratiques pour le développement logiciel :

- la gestion de version ;
- l'utilisation de moteur de production ;
- l'utilisation de tests et séparation code de production et test ;
- l'introduction à l'intégration continue et déploiement/livraison continu (CI/CD).

2 Logiciels

Si vous effectuez ce TP depuis votre machine personnelle, il vous faut installer les outils suivants :

- **Terminal shell** si vous êtes sous *windows*, nous vous conseillons d'installer ubuntu sur windows pour avoir accès à un terminal shell. Sous *MacOS* et*linux*, un terminal shell est disponible de base. On vous conseille d'installer Oh-my-zsh pour votre terminal qui rend l'utilisation de *git* dans le terminal plus convivial.
- **JDK version 21 ou plus** la méthode d'installation dépend de votre système d'exploitation :
  - **Windows**: Télécharger et exécuter l'Installeur windows
  - **MacOS**: Télécharger et exécuter l'Installeur MacOS (attention à prendre le bon installeur suivant le processeur de votre ordinateur aarch64 pour les processeurs M1, M2, M3 ou M4 et x64 pour les processeurs *intel*)
  - **Linux**: exécuter la commande suivante dans le terminal `sudo apt install openjdk-21-jdk`. Un mot de passe administrateur vous sera demandé.
- **Gradle**: Suivez les instructions au lien suivant : install gradle
- **git**: Télécharger et installer git au lien suivant : download git
- **IntelliJ IDEA** : vous pouvez télécharger IntelliJ IDEA ou bien JetBrains Toolbox qui est un outil pour gérer les IDE proposé par l'entreprise JetBrains. En tant qu'étudiant, vous avez d'ailleurs accès à des licences gratuites pour leurs produits et notamment la version *ultimate* d'*IntelliJ IDEA* en créant un compte avec votre adresse AMU. Vous pouvez aussi utiliser Eclipse ou Visual studio code qui sont des outils similaires.

3 Création de projet

Le but de la première partie de ce TP est d'apprendre à configurer un projet en utilisant *IntelliJ IDEA* ou bien entièrement en ligne de commande. Le projet permettra d'avoir de :

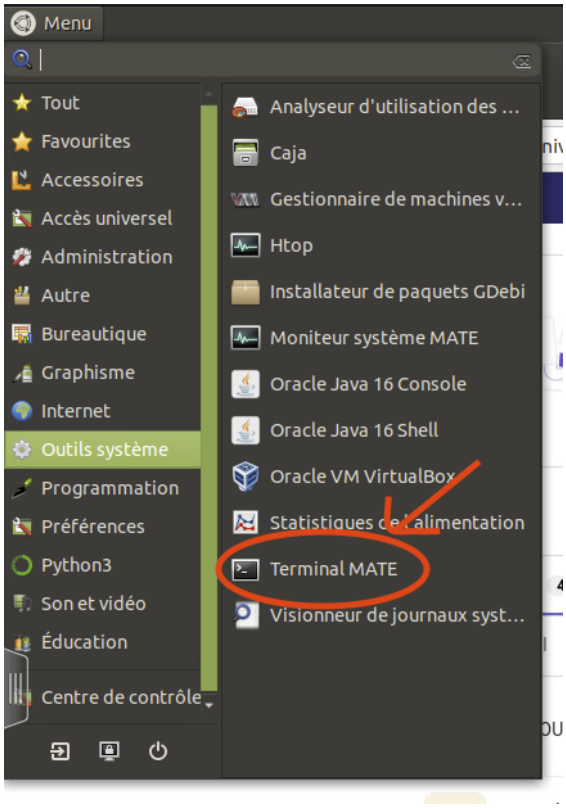
— la gestion de version via git avec la création de branches ;
— les tests unitaires avec Junit et AssertJ ;
— un moteur de production gradle afin de gérer des dépendances et d'exécuter les tests unitaires ou des outils de mesure de la qualité du code.

3.1 Configuration clé ssh pour le gitlab AMU

La première étape pour utiliser la gestion de version est de vous connecter au gitlab de l'université qui est accessible à l'adresse suivante : https://etulab.univ-amu.fr/. L'identifiant et le mot de passe sont ceux de votre compte étudiant AMU.

Afin de pouvoir accéder à distance à vos dépôts git, vous allez configurer une clé SSH dans votre profil. Pour cela, il vous faut :

1. Ouvrir un terminal (par exemple terminal MATE de la vdi Lunix qui est accessible dans le menu au sous-menu Outils systèmes). Si vous êtes sous *windows* et que vous avez installé Git pour windows, il est conseillé d'utiliser le terminal bash de git.



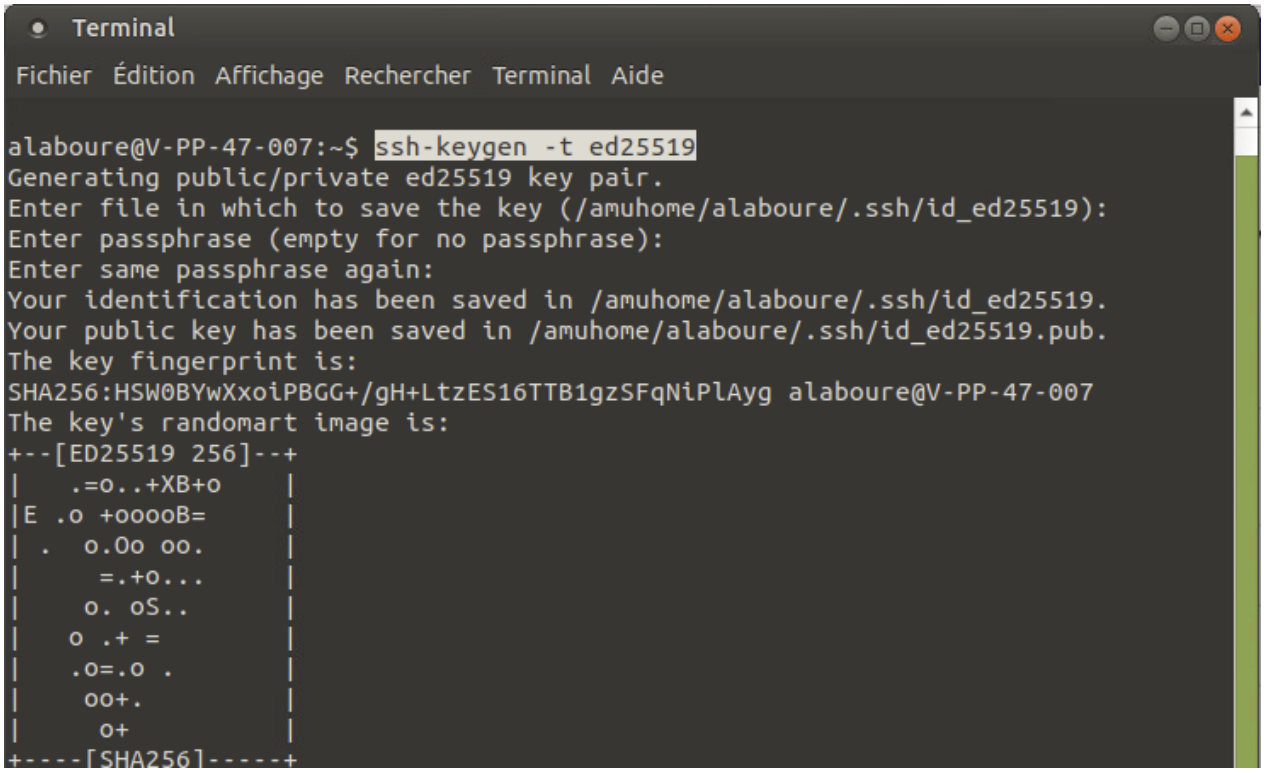
2. Générer une paire de clés privé/publique pour votre compte. Pour cela, il vous faut entrer la commande suivante dans le terminal :

```
~$ ssh-keygen -t ed25519
```

Appuyer une fois sur Entrée pour confirmer que vous voulez sauvegardez vos clés dans le répertoire par défaut.

Ensuite, il vous est demandé de rentrer deux fois une “passphrase”. Vous pouvez entrer une phrase (plusieurs mots donc) qui servira de mot de passe pour l’accès. Puisque la sécurité de vos dépôt n’est pas critique, vous pouvez vous contenter de ne rien rentrer (pas de passphrase) et donc d’appuyer de nouveau sur Entrée deux fois.

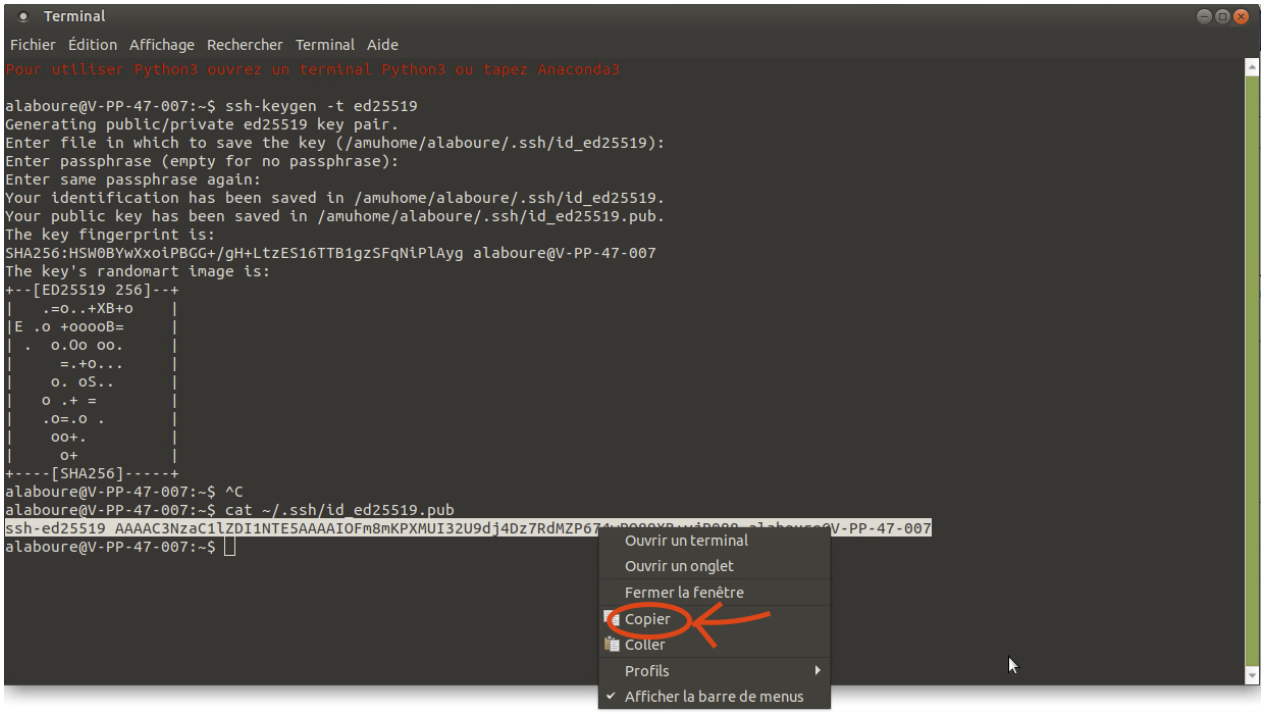
Si tout s’est bien passé, votre couple de clés a été généré et vous devriez voir un affichage similaire à celui ci-dessous :



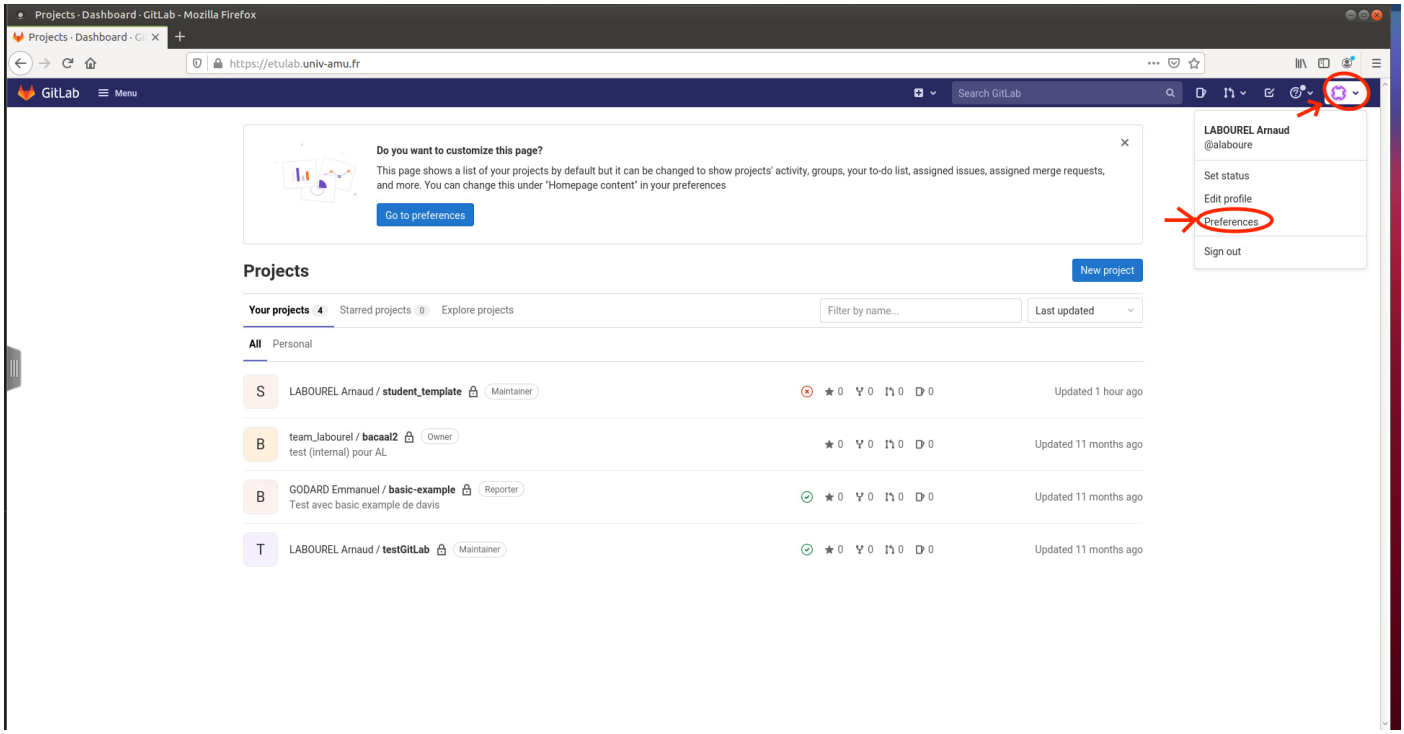
Afficher votre clé dans le terminal en entrant la commande suivante (sous powershell, il faut remplacer la commande `cat` par la commande `type`) :

```
~$ cat ~/.ssh/id_ed25519.pub
```

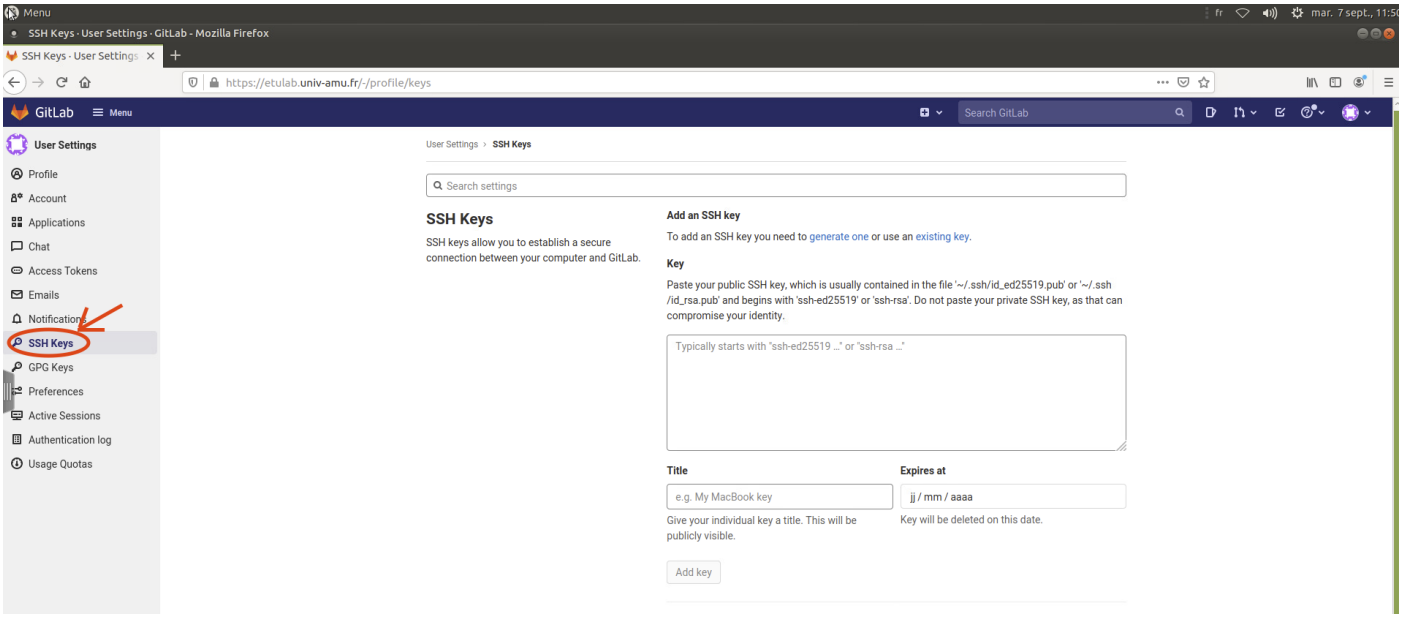
Sélectionner la ligne affichée par le terminal et copier là dans le presse-papier (sélection puis clic droit et copier)



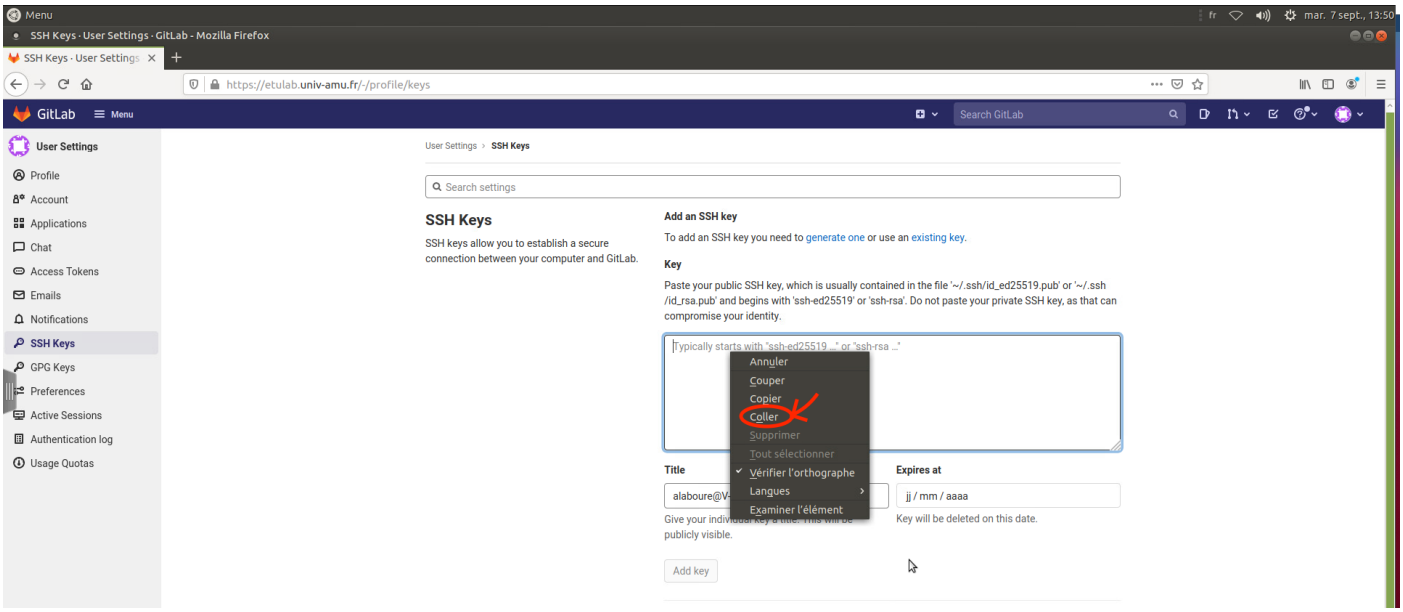
3. Configurer la clé dans votre compte gitlab. Pour cela, vous allez accéder aux préférences de votre compte gitlab. Il vous faut vous connecter à https://etulab.univ-amu.fr/ puis cliquez sur votre avatar en haut à gauche de l'écran puis sur *preferences* dans le menu qui apparaît.



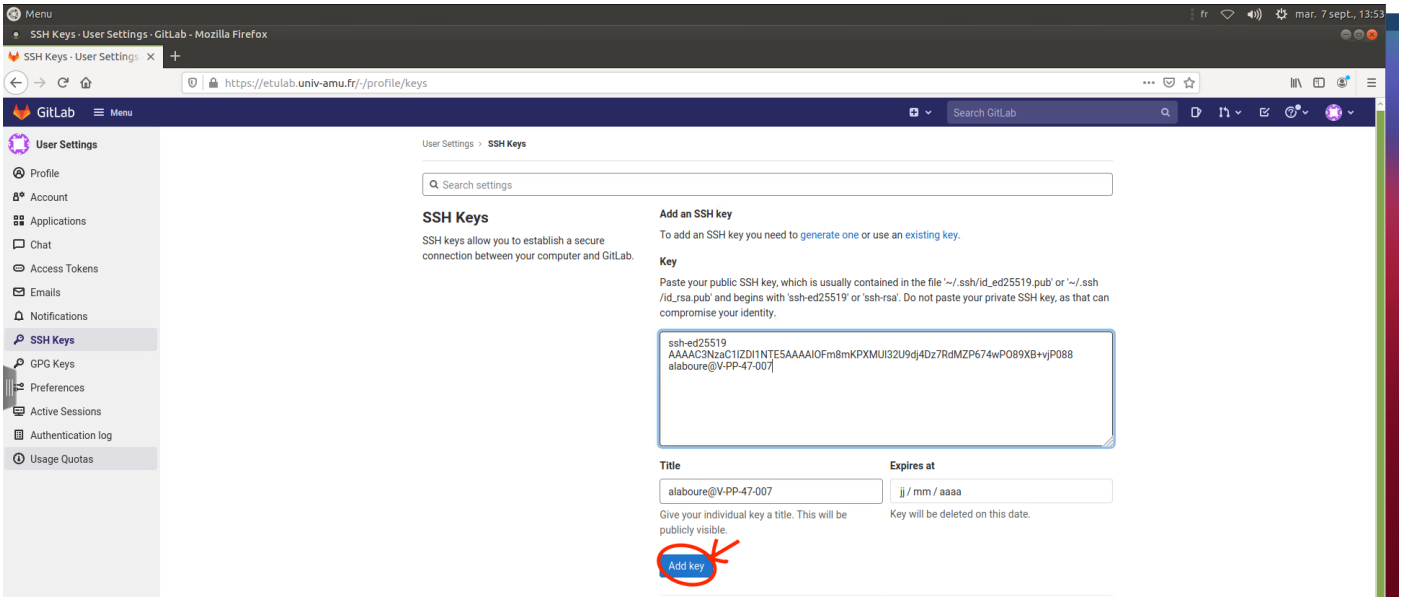
Allez au menu des clés SSH en cliquant sur le lien correspondant dans le menu de gauche.



4. Revenez sur la page de configuration des clés ssh sur votre navigateur. Coller votre clé (clic droit puis coller) dans l'espace prévu pour cela.



Ajouter la clé en cliquant sur le bouton *add*.



Vous devriez recevoir un mail sur votre mail étudiant confirmant que vous avez rajouté une clé. Vous pouvez rajouter autant de clés que vous voulez. Vous pouvez donc faire de même pour rajouter par exemple des clés supplémentaires si vous souhaitez accéder à votre dépôt depuis chez vous.

3.2 Création de projet à l'aide d'IntelliJ

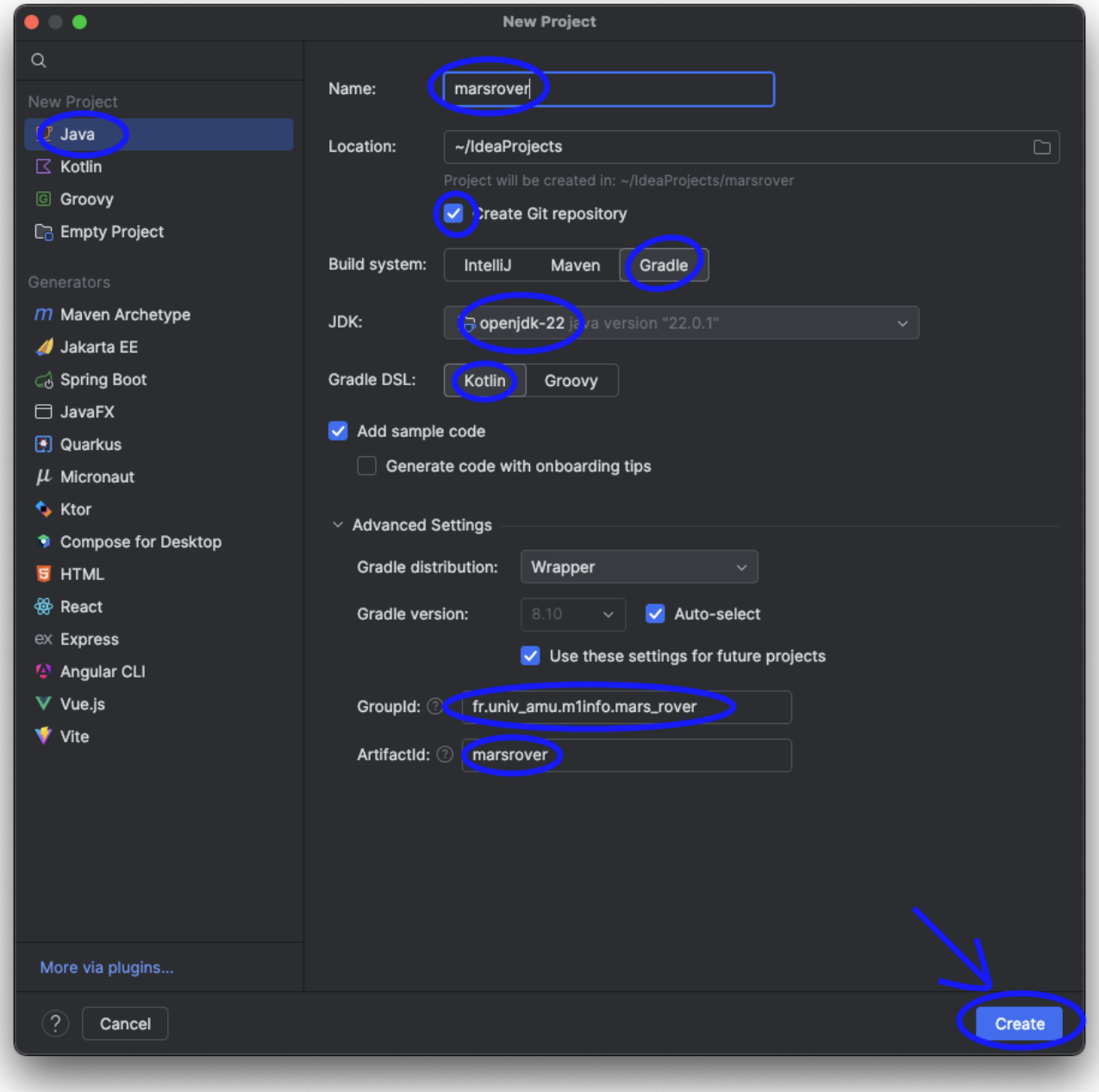
3.2.1 Création du dépôt local

- Lancez IntelliJ IDEA
- Commencez la création d'un nouveau projet via le menu `file -> New -> Project`.
- Créer un projet pour le TP. Pour ce faire, il vous faut (dans l'ordre du haut vers le bas) :
  - Sélectionner dans la colonne de gauche *Java* comme type de projet
  - choisir un nom (*name*) pour votre projet, pour ce projet, le nom sera `mars-rover` ;
  - choisir le répertoire de stockage (*Location*) du projet (vous pouvez laisser le répertoire de base qui est `~/IdeaProjects/`)
  - cocher la case *Create Git repository* pour créer un dépôt git local ;
  - choisir un JDK de version 21 ou plus (normalement JDK 22) et
  - choisir gradle comme moteur de production (*Build system*) ;
  - choisir Kotlin comme Gradle DSL ;
  - ouvrir les paramètres avancés (*Advanced Settings*) ;
    - choisir un identifiant de groupe [1] (*GroupId*) pour votre projet, pour ce projet, vous devez mettre `fr.univ_amu.m1info.mars_rover` ;
    - choisir un identifiant de composant [2] (*ArtifactId*) pour votre projet, vous pouvez laisser le nom du

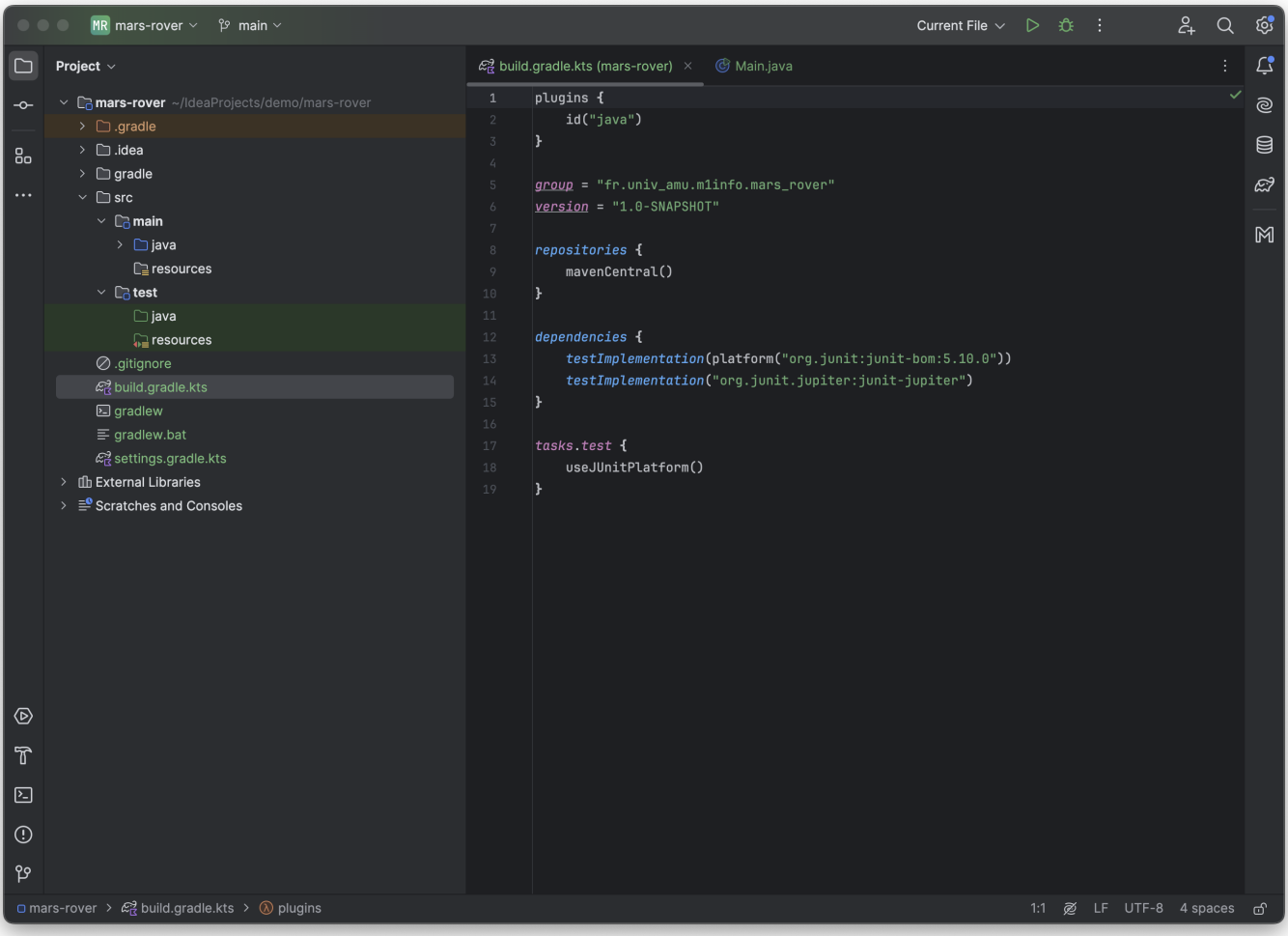
1. L'identifiant de groupe (*GroupId*) permet de connaitre l'organisation, l'entreprise, l'entité ou la communauté qui gère le projet. Par convention, on utilise le nom de domaine Internet inversé, selon la même logique que celle généralement recommandée pour les noms de packages Java.
2. L'identifiant de composant (*ArtifactId*) est le nom unique du projet au sein du groupe qui le développe. Généralement, l'identifiant du composant est le nom du projet.

projet (`mars-rover`).

— cliquez sur le bouton *Create*.

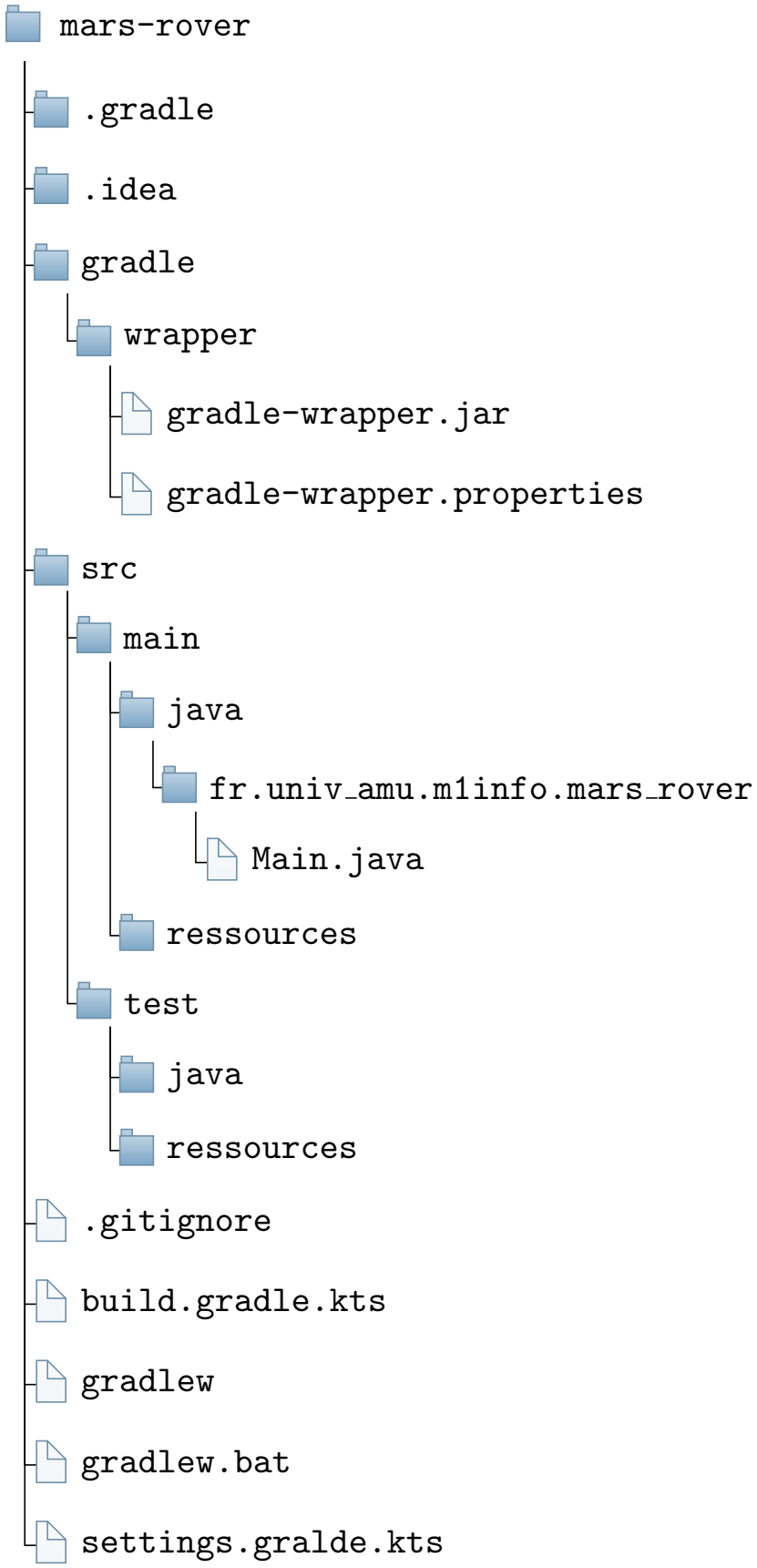


Votre projet devrait se configurer automatiquement et vous devriez obtenir l'affichage suivant.



3.2.2 Explication fichiers et répertoires du projet

Votre projet devrait avoir l'arborescence de fichiers suivante :


mars-rover
.gradle
.idea
gradle
wrapper
gradle-wrapper.jar
gradle-wrapper.properties
src
main
java
fr.univ_amu.m1info.mars_rover
Main.java
ressources
test
java
ressources
.gitignore
build.gradle.kts
gradlew
gradlew.bat
settings.gralde.kts

— Le répertoire `.gradle` contient les fichiers de configuration de *gradle*.
— Le répertoire `.idea` contient les fichiers de configuration d'*IntelliJ Idea*.
— Le répertoire `gradle` contient le gradle wrapper qui permet de fixer la version de *gradle* (la téléchargeant au besoin).
— Le répertoire `src` contient le code du projet.
  — Le sous-répertoire `main` contient les fichiers pour le code principal de l'application, c'est-à-dire les fichiers nécessaires pour le lancement de l'application. Le sous-répertoire `java` contient les fichiers *Java* alors que le sous-répertoire `ressources` contient les autres types de fichiers comme des fichiers de configuration ou des images pour l'interface graphique.
  — Le sous-répertoire `test` contient les fichiers pour le code de test. Le sous-répertoire `java` contient les fichiers *Java* alors que le sous-répertoire `ressources` contient les autres types de fichiers comme des fichiers d'entrée pour des tests de lecture.
— Le fichier `.gitignore` liste les fichiers à ignorer lors des *commit* de *git*.
— Le fichier `build.gradle.kts` est un fichier de configuration *gradle* définissant en autre les dépendances du projet.
— Les fichiers `gradlew` et `gradle.bat` sont les fichiers de script pour le *wrapper gradle* respectivement pour les systèmes *Unix* et *Windows*.
— Le fichier `settings.gradle.kts` est un fichier de configuration de *gradle* permettant en autre de définir le nom du projet.

3.2.3 Création du dépôt distant et du lien entre les deux dépôts

— Connectez-vous via à un navigateur à votre compte etulab et créez un nouveau projet vide (sans `readme`) nommé `mars-rover` en cliquant sur `+` en haut à gauche puis sur `Create new project/repository` ou bien avec le bouton `New project` en haut à droite.
— Copiez l'adresse du projet distant en cliquant sur le bouton `clone` puis sur l'icône de copie `ssh`.
— Retournez sur votre projet sur `IntelliJ` et allez dans le menu : `git` puis `Manage Remotes`.
— Cliquez sur le bouton `+` pour ajouter un dépôt distant à votre dépôt local.
— Copiez l'adresse de votre dépôt distant dans le champ `URL` et validez l'ajout en cliquant sur le bouton `OK` deux fois (une fois pour la fenêtre `Define remote` et une autre fois pour la fenêtre `Git remotes`).

3.3 Création de projet versionné en ligne de commande

Si vous avez créé votre projet avec *IntelliJ IDEA*, vous pouvez ignorer cette partie qui décrit la création de projet utilisant uniquement le terminal shell et donc sans *IntelliJ IDEA*.

3.3.1 Création de projet gradle en ligne de commande

La première étape de la création du projet est de créer un projet *gradle*.

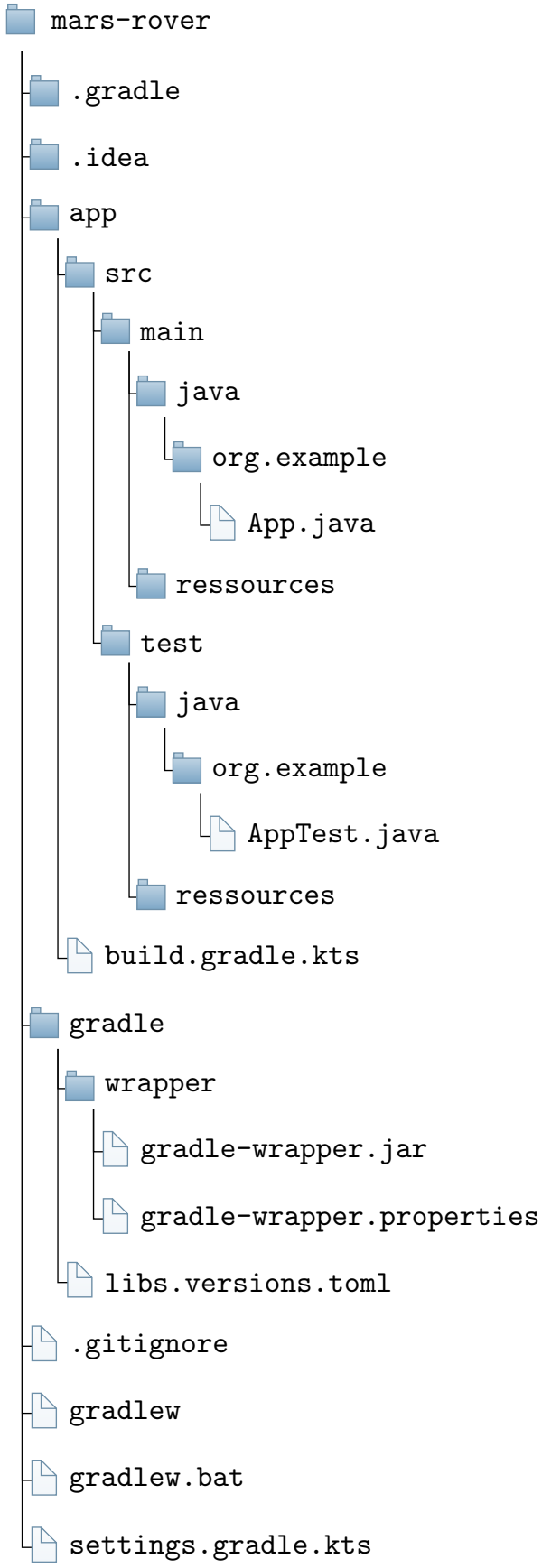
— Créez le répertoire qui va contenir votre projet qui devra être nommé `mars-rover`.
— Pour commencer la création de votre projet lancez la commande `gradle init` alors que vous êtes dans le répertoire `mars-rover`.

— On vous demande tout d'abord quel type de projet vous souhaitez créer. Tapez `1` puis `entrée` pour choisir application.
— On vous demande le langage de projet. Tapez `1` puis `entrée` pour choisir Java.
— On vous demande ensuite la version java du projet, vous pouvez laisser 21 qui est la version par défaut.
— On vous demande le nom de votre projet. Vous pouvez laisser le nom par défaut ou bien définir votre propre nom de projet.
— On vous demande si votre projet est une application simple ou application et *library*. Tapez `1` et validez avec la touche entrée pour choisir `Single application project`.
— On vous demande si vous préférez utiliser `Groovy` ou `Kotlin` pour le langage de script de *build*. Tapez `1` puis `entrée` pour choisir `Kotlin`.
— On vous demande quel framework de tests vous souhaitez utiliser. Tapez `4` puis `entrée` pour choisir `Junit Jupiter`.

—

3.3.2 Explication fichiers et répertoires du projet

Votre projet devrait avoir l'arborescence de fichiers suivante :


mars-rover
.gradle
.idea
app
src
main
java
org.example
App.java
ressources
test
java
org.example
AppTest.java
ressources
build.gradle.kts
gradle
wrapper
gradle-wrapper.jar
gradle-wrapper.properties
libs.versions.toml
.gitignore
gradlew
gradlew.bat
settings.gradle.kts

— Le répertoire `.gradle` contient les fichiers de configuration de *gradle*.
— Le répertoire `.idea` contient les fichiers de configuration d'*IntelliJ Idea*.
— Le répertoire `gradle` contient le gradle wrapper qui permet de fixer la version de *gradle* (la téléchargeant au besoin).
— Le sous-répertoire `src` du répertoire `app` contient les fichiers sources du projet.
  — Le sous-répertoire `main` contient les fichiers pour le code principal de l'application, c'est-à-dire les fichiers nécessaires pour le lancement de l'application. Le sous-répertoire `java` contient les fichiers *Java* alors que le sous-répertoire `ressources` contient les autres types de fichiers comme des fichiers de configuration ou des images pour l'interface graphique. Le fichier `App.java` contient la méthode `main` qui est exécuté lorsque vous taper la commande `gradle run`.
  — Le sous-répertoire `test` contient les fichiers pour le code de test. Le sous-répertoire `java` contient les fichiers *Java* alors que le sous-répertoire `ressources` contient les autres types de fichiers comme des fichiers d'entrée pour des tests de lecture. Le fichier `AppTest.java` contient des tests qui sont exécutés lorsque vous taper la commande `gradle test`. Si les tests échouent la commande vous donne un lien à ouvrir avec un navigateur.
  — Le fichier `build.gradle.kts` est un fichier de configuration *gradle* définissant en autre les dépendances du projet.
— Les fichiers `gradlew` et `gradle.bat` sont les fichiers de script pour le *wrapper gradle* respectivement pour les systèmes *Unix* et *Windows*.
— Le fichier `settings.gradle.kts` est un fichier de configuration de *gradle* permettant en autre de définir le nom du projet.

3.3.3 Création de dépôt `git`

— Dans le répertoire contenant votre projet `gradle`, exécutez la commande `git init --initial-branch=main` pour créer un dépôt `git` local.
— Dans le répertoire contenant votre projet `gradle`, exécutez la commande `git add non_de_fichier` pour ajouter les fichiers de votre projet à votre prochain `commit`.
— Il vous faudra ajouter tous les fichiers `Java` ainsi que les fichiers de configuration de gradle : `build.gradle.kts`, `gradlew`, `gradlew.bat`, `settings.gradle.kts`, `gradle-wrapper.properties` et `libs.versions.toml`.

— Exécutez la commande `git commit -m"premier message"` pour faire un premier `commit` de votre projet avec votre propre message.

3.3.4 Création du dépôt distant et du lien entre les deux dépôts

— Connectez-vous via à un navigateur à votre compte etulab et créez un nouveau projet en cliquant sur `menu` puis `Create new project`.
— Choisissez `Create blank project`.
— Choisissez un nom pour votre projet, décochez la création du `README.md` et validez la création en cliquant sur le bouton `Create project`.

— Copiez l'adresse pour cloner votre projet en cliquant sur le bouton `clone` puis sur l'icône de copie `ssh`.
— Utiliser la commande suivante pour rajouter l'adresse du dépôt distant à votre dépôt local. Il vous faudra bien évidemment remplacer le dernier argument de la commande par l'adresse de votre projet que vous avez préalablement copié : `git remote add origin adresse_projet`.
— Faites le premier `push` à l'aide de la commande `git push -u origin`. Pour les `push` suivants, vous pourrez simplement utiliser la commande `git push`, car vous avez configuré les branches.

4 Mars rover

4.1 Description

La NASA doit faire atterrir une escouade de *rovers* robotisés sur un plateau martien. Ce plateau rectangulaire doit être parcouru par les *rovers* afin que leurs caméras embarquées puissent obtenir une vue complète du terrain environnant et la transmettre à la Terre. La position d'un *rover* est représentée par un triplet composé des deux coordonnées x et y de la position initiale du *rover* et l'orientation initiale du *rover* correspondant à un des quatre points cardinaux (`NORTH`, `EAST`, `SOUTH` ou `WEST`). Le point d'origine $(0, 0)$ du système de coordonnées correspond au coin sud ouest du plateau. Le plateau est rectangulaire et est modélisé par une grille définie par une largeur et hauteur pour simplifier la navigation. Par exemple, le triplet $(0, 0, NORTH)$ correspond à un *rover* en position $(0, 0)$ et orienté vers le nord. Les coordonnées possibles pour une grille de largeur l et de hauteur h sont les couples (x, y) tels que $0 \leq x < l$ et $0 \leq y < h$. Les coordonnées $(l - 1, h - 1)$ correspondent au coin nord est de la grille.

Pour contrôler un *rover*, la NASA donne une suite finie de lettres. Les lettres possibles sont `L`, `R` et `M`. Les lettres `L` et `R` font tourner le *rover* de 90 degrés vers la gauche ou vers la droite respectivement, sans qu'il ne bouge de son emplacement actuel. La lettre `M` indique au *rover* d'avancer d'une case dans le sens de son orientation. On suppose que tout mouvement faisant sortir le *rover* de la grille entraîne sa destruction et qu'il n'exécute pas les commandes suivantes. Une configuration pourra comprendre plusieurs *rovers*, chacun agissant indépendamment l'un après l'autre. On supposera qu'il n'y pas d'interactions entre les *rover* (deux *rovers* peuvent être aux mêmes coordonnées au même moment). Le but étant d'explorer le plateau, la sortie devra comprendre le pourcentage de coordonnées explorées par les *rovers*, une case étant considérée comme étant explorée si elle correspond à une case dans la trajectoire d'un des *rovers*, y compris leurs coordonnées de départ.

4.2 Spécification des entrées/sorties

Le fichier d'entrée du problème est constitué d'une ligne décrivant la grille suivie d'un nombre non-nul de lignes chacune décrivant un *rover*. La ligne de description de la grille contient deux entiers séparés par un espace, le premier correspondant à la largeur de la grille alors que le deuxième correspond à sa hauteur. Une ligne de description d'un *rover* comporte les coordonnées en x et y de sa position initiale, son orientation initiale (écrit en lettre majuscule) et sa liste de commandes, chaque élément étant séparé par un espace.

Par exemple, une configuration correspondant à une grille de taille 5×5 avec deux *rovers* :

— le premier de coordonnées $(1, 2)$, orienté vers le nord avec comme commandes `LMLMLMLMM` ;
— le second de coordonnées $(3, 3)$, orienté vers l'est avec comme commandes `MMRMMRMRRM` ;

aura le fichier de configuration suivant :

```
5 5
1 2 NORTH LMLMLMLMM
3 3 EAST MMRMMRMRRM
```

Le fichier de sortie sera composé d'une ligne indiquant le pourcentage d'exploration du plateau (pourcentage des coordonnées explorées), suivi d'une ligne par *rover* correspondant à leur position finale (position à la fin de l'exécution de commande ou position juste avant la destruction du *rover* si celui-ci sort du plateau) avec leur orientation. Par exemple, la configuration précédemment décrite devra produire la sortie suivante :

```
28 %
1 3 NORTH
4 3 EAST
```

5 Méthodologie de travail

5.1 Tests unitaires avec Junit 5

Pour ce TP, vous devez tester à l'aide de tests unitaires toutes les classes que vous écrirez. Normalement votre projet est déjà configuré pour le framework de test JUnit 5. On vous conseille d'utiliser AssertJ qui ajoute entre autre des assertions facilitant l'écriture des tests. Pour cela, il vous faut rajouter *AssertJ* dans les dépendances de *gradle* en rajoutant les lignes suivantes dans le fichier `build.gradle.kts` :

```
dependencies {
    testImplementation("org.assertj:assertj-core:3.26.3")
}
```

Pour pouvoir utiliser facilement les assertions (méthodes `assertThat`) dans vos classes de test, il vous faut ajouter l'*import* suivant dans chacune vos classes de tests :

```
import static org.assertj.core.api.Assertions.*;
```

Vous trouverez davantage de détails sur les tests unitaires dans le document dédié aux tests

5.1.1 Tâches `run` et `build`

Pour configurer la tâche `run`, on va commencer par rajouter (si besoin) le plugin application dans le fichier `gradle.build.kts`. Pour cela, on rajoute la ligne suivante :

```
plugins {
  id("application")
}
```

Comme indiqué précédemment, la tâche `run` vous permet d'exécuter la méthode `main` de la classe spécifiée dans le fichier `gradle.build.kts`. Cette classe correspond à la propriété `mainClass` du plugin application à l'aide

des lignes suivantes :

```
application {
  mainClass = "fr.univ_amu.m1info.mars_rover.Main"
}
```

La tâche `build` définie par le plugin `java` permet de construire un `jar` dans le répertoire `build/libs` du projet. Afin que le `jar` fonctionne, il faut préciser la classe contenant le `main` à exécuter. Cela se fait par le biais du fichier manifest que l'on peut configurer en ajoutant les lignes suivantes dans le fichier `gradle.build.kts`.

```
tasks.withType<Jar> {
    manifest {
        attributes["Main-Class"] = "fr.univ_amu.m1info.mars_rover.Main"
    }
}
```

5.2 Intégration continue et déploiement/livraison continu (CI/CD)

CI/CD (*Continuous Integration/Continuous Delivery*) est la combinaison des pratiques d'intégration continue et de livraison/déploiement continu. L'intégration continue consiste à tester et contrôler (via des outils de mesure de qualité de code) en permanence les modifications itératives du code. La livraison continue consiste à rajouter à l'intégration continue la diffusion automatisée du logiciel (le déploiement restant non automatisé dans ce cas). Le déploiement continu ajoute le déploiement automatique dans les environnements de production comme la mise à jour des serveurs avec la nouvelle version du logiciel dans le cas d'une application web.

L'instance Gitlab de l'université d'Aix Marseille connu sous le nom d'etulab est configuré pour le CI/CD avec des *runners* intégré qui permette d'exécuter les tests et les scripts de livraison/déploiement. Gitlab permet de facilement mettre en place un processus de CI/CD simple. La principale étape est de configurer la pipeline que vous souhaitez exécuter lors des *push* sur le git. Pour cela, il faut passer par un fichier de configuration `.gitlab-ci.yml` à la racine du projet. Le fichier est au format YAML et permet de définir :

- l'image à utiliser pour exécuter les scripts ;
- la structure en *stages* (étapes) des différents *jobs* à exécuter ;
- les différents *jobs* avec le script à exécuter pour chacun et les artéfacts associés, c'est-à-dire les fichiers produits à conserver sur le serveur.

Vous trouverez le fichier de base à rajouter à votre projet au lien suivant : .gitlab-ci.yml.

La première ligne du fichier configure l'image à utiliser comme étant l'image alpine de gradle qui est l'image conseillée pour un CI/CD utilisant *gradle* comme moteur de production.

```
image: gradle:jdk22-alpine
```

Les lignes suivantes indiquent que le pipeline est composé de deux *stages build* et *tests* qui sont des *stages* classiques.

```
stages:
  - build
  - test
```

Les lignes suivantes configurent les options de *gradle* afin que le *Gradle Daemon* soit désactivé, car il est inutile dans un contexte de pipeline CI/CD.

```
variables:
  GRADLE_OPTS: "-Dorg.gradle.daemon=false"
```

Les lignes suivantes définissent un script à lancer avant le pipeline. Il configure la variable système `GRADLE_USER_HOME` afin que *gradle* stocke sa configuration globale dans le répertoire du projet.

```
before_script:
  - export GRADLE_USER_HOME=`pwd`/.gradle
```

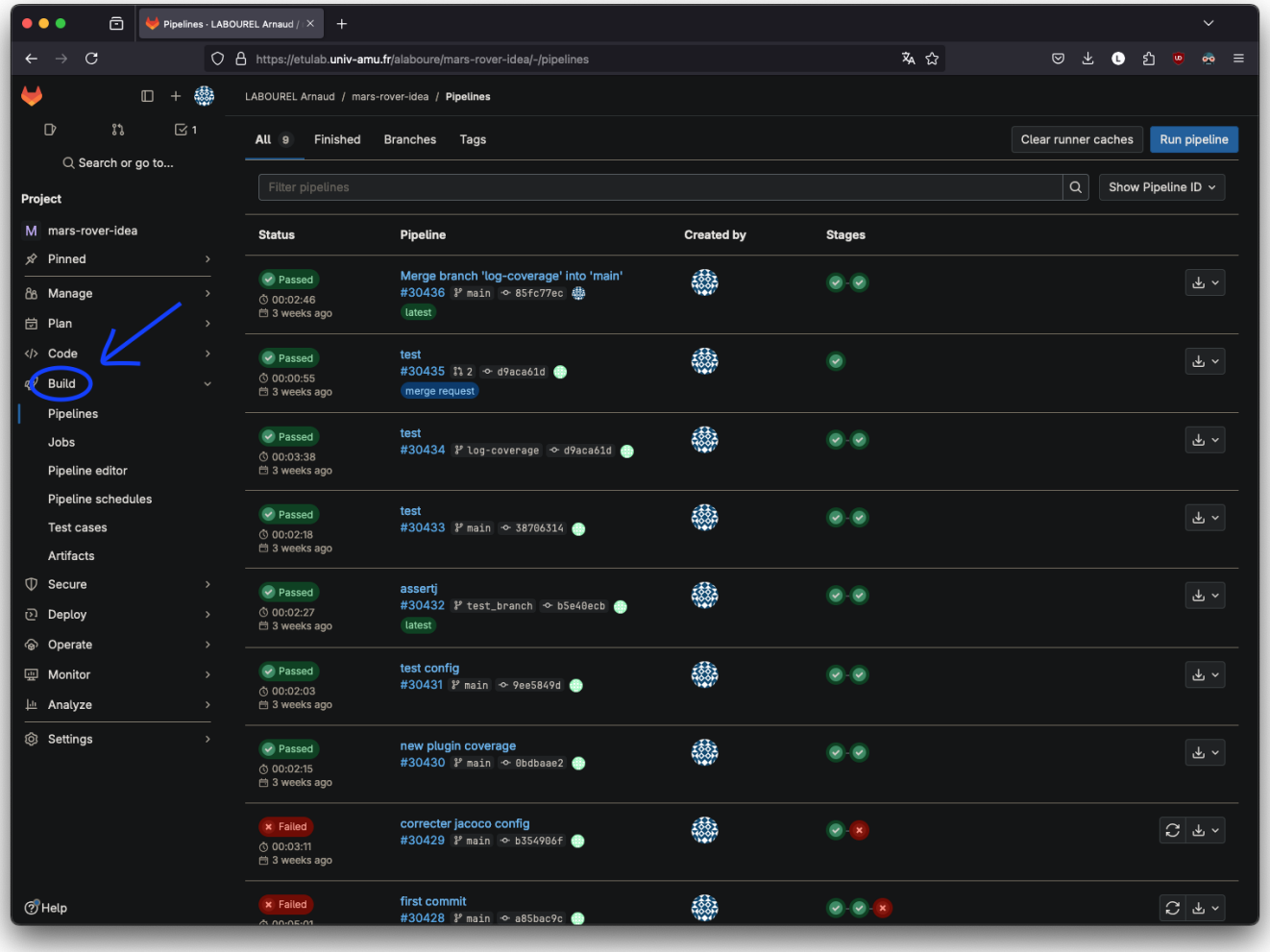
Les lignes suivantes définissent le job *build* dans le stage *build*. Ce *job* se contente d'exécuter la tâche *build* de *gradle* et de définir comme artéfact les fichiers *jar* produits dans le répertoire dédié.

```
build:
  stage: build
  script:
    gradle --build-cache build
  artifacts:
    paths:
      - build/libs/*.jar
    expire_in: 1 week
```

Finalement, les dernières lignes du fichier définissent le job *tests* dans le *stage test*. Ce *job* se contente d'exécuter la tâche *test* de *gradle* et de définir comme rapports les fichiers *xml* produits dans le répertoire dédié.

```
tests:
  stage: test
  script:
    - gradle test
  artifacts:
    when: always
    reports:
      junit: build/test-results/test/**/TEST-*.xml
```

Une fois que vous avez ajouté ce fichier à votre projet et que vous avez poussé la mise à jour sur le serveur, le pipeline devrait s'exécuter. Vous pouvez accéder à toutes les informations de vos pipelines via l'interface web de *gitlab* en cliquant sur *build* dans le menu de votre projet.



Le sous-menu *build* contient les articles suivants :

— *Pipelines* qui contient l'historique de tous les pipelines lancés par le projet (normalement un par *push*) ;
— *Jobs* qui contient l'historique de tous les *jobs* lancés par le projet (même contenu que l'article précédent, mais présenté par *job*) ;
— *Pipeline editor* permet de directement reconfiguré le pipeline via l'interface web ;
— *Test cases* qui permet de décrire des scénarios de test (pas utilisé dans ce TP) ;
— *Artefacts* permet de télécharger les artéfacts produits par les *jobs*.