amU Faculté des sciences Aix Marseille Université

Année 2025-26

Domaine Sciences et Technologies
Master informatique
**Génie logiciel : TP 1**
**Codes UE : SINA09EL**
**Mars Rover (2/3)**

# 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 ;

Ce TP peut se faire seul ou à deux (en binôme).

# 2 Installation de logiciels sur votre machine personnelle

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 ;
— 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 ;

On vous conseille aussi d'utiliser un IDE moderne pour le développement et non pas un simple éditeur de texte. Les instructions qui suivent décrivent l'utilisation d'IntelliJ IDEA mais vous pouvez utiliser d'autres IDE tels que Eclipse ou Visual studio code du moment que vous savez l'utiliser.

La création de projet se fera en plusieurs étapes :

1. La configuration d'une clé `ssh` pour se connecter au gitlab d'AMU appelé etulab.
2. La création de dépôts pour le projet : un sur le serveur d'*etulab* et un local sur lequel vous aller travailler (soit sur votre machine personnelle ou bien votre compte personnel AMU). Pour créer les deux dépôts, on vous propose soit d'utiliser l'IDE IntelliJ IDEA ou bien le terminal en ligne de commande.

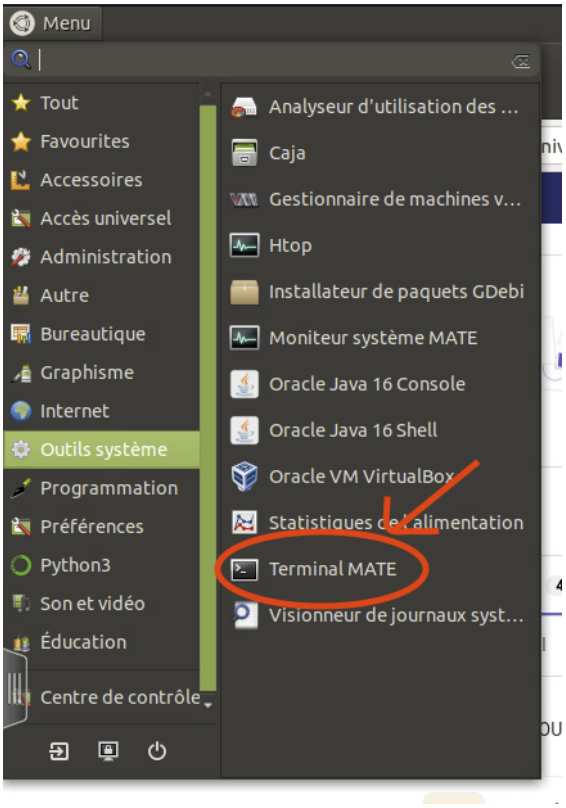
### 3.1 Configuration clé ssh pour le gitlab AMU

**Tâche 1 :** créez puis utilisez une clé ssh en suivant les instructions qui suivent.

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 devoir configurer une clé SSH dans votre profil si ce n'est pas déjà fait. 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.



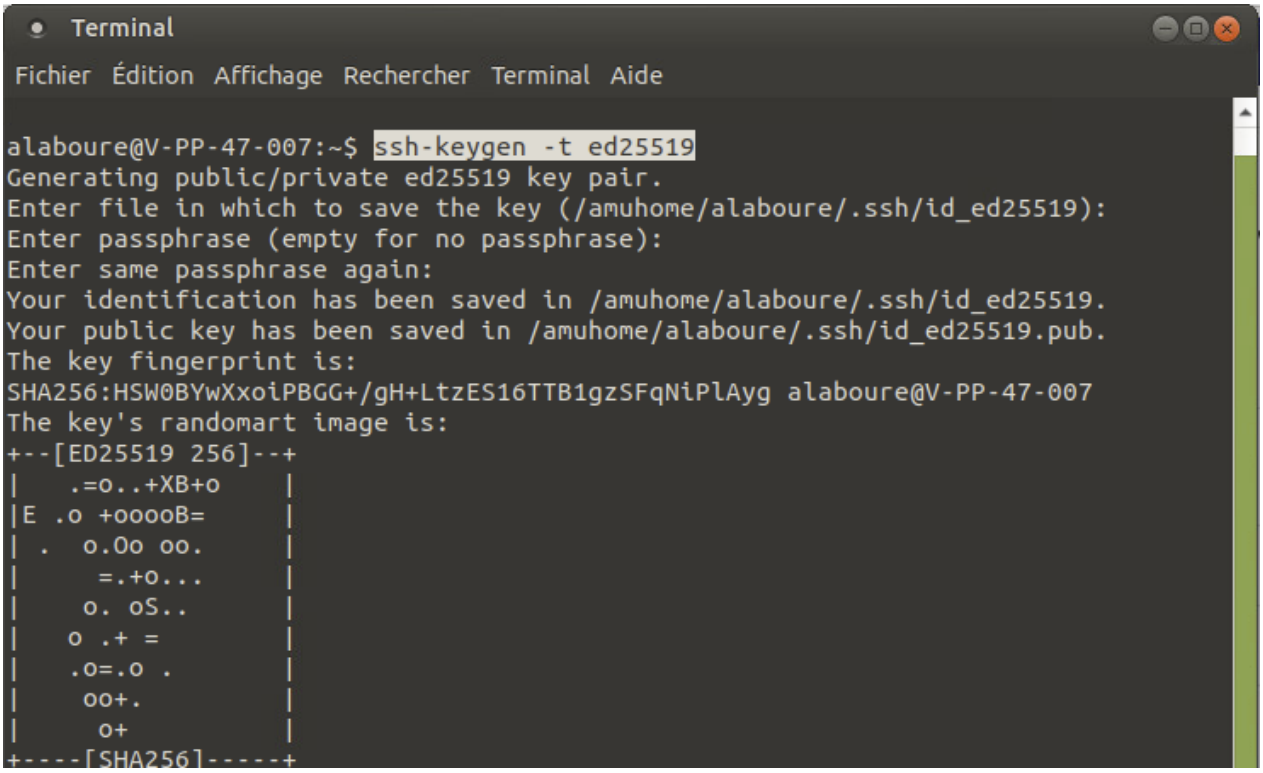

2. Générer une paire de clés privé/publique pour votre compte. Pour cela, il vous faut entrer la commande suivante dans le terminal :

```
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.

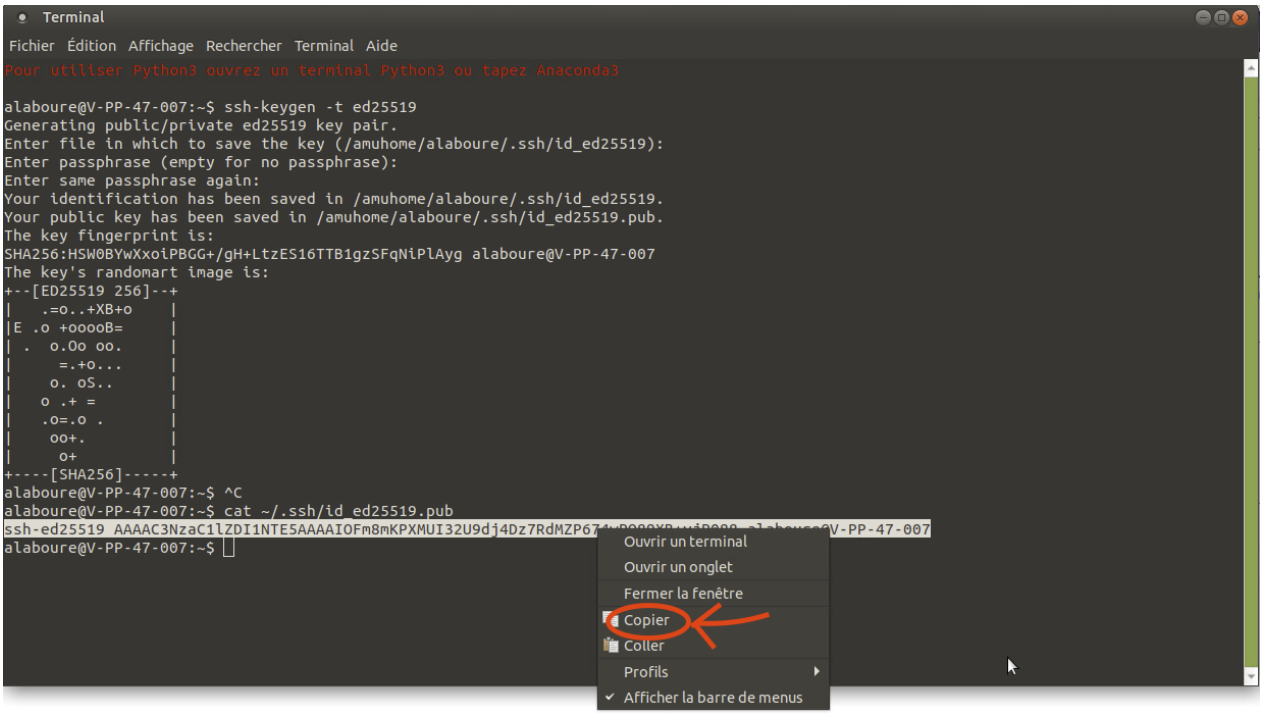
Si tout s’est bien passé, votre couple de clés a été généré et vous devriez voir un affichage similaire à celui ci-dessous :



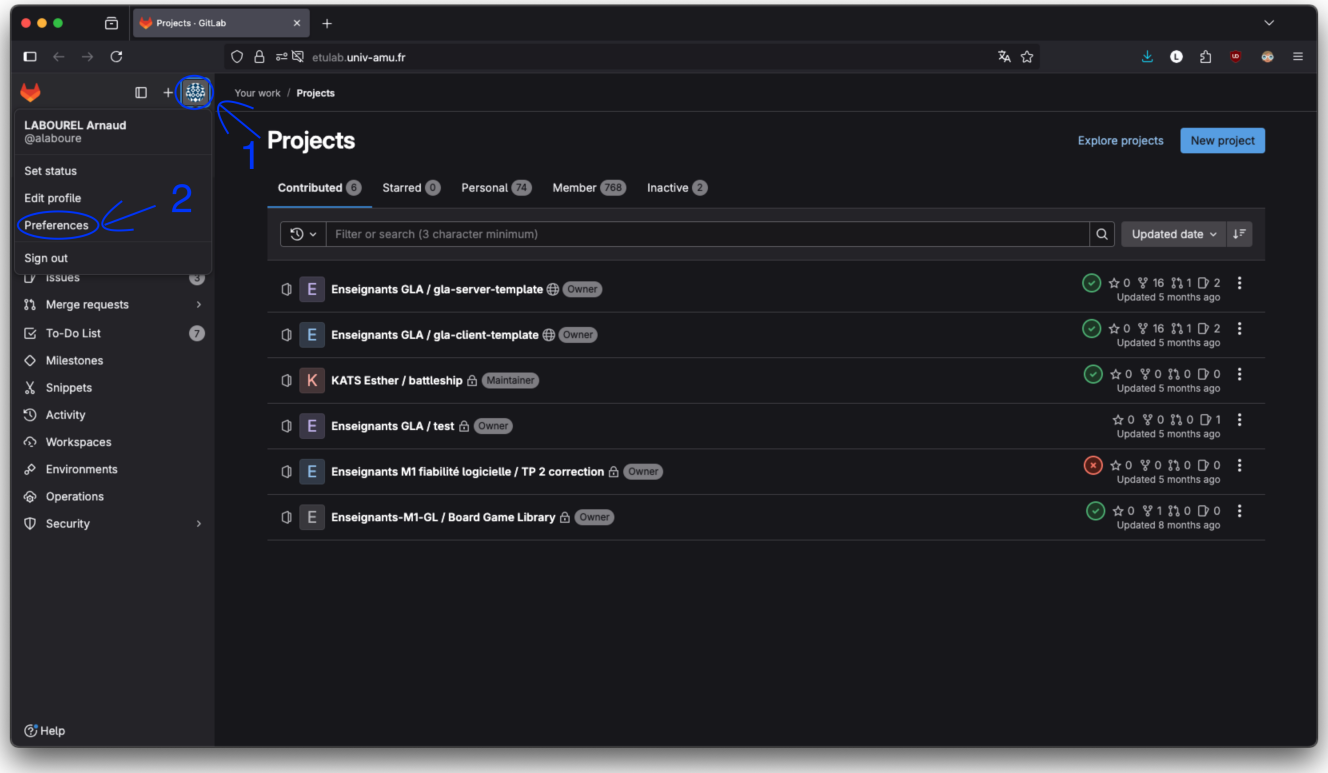

Afficher votre clé dans le terminal en entrant la commande suivante (sous powershell, il faut remplacer la commande `cat` par la commande `type`) :

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

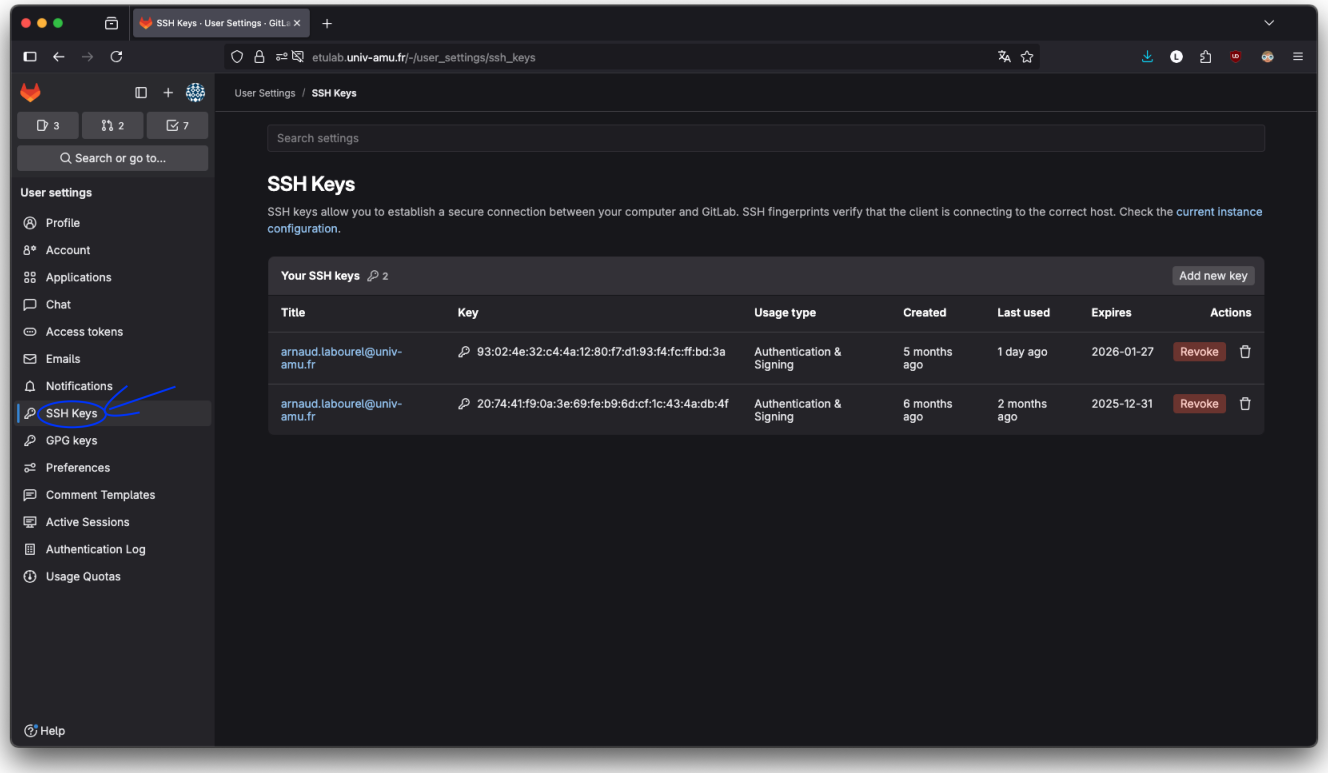
Sélectionner la ligne affichée par le terminal et copier là dans le presse-papier (sélection puis clic droit et copier)



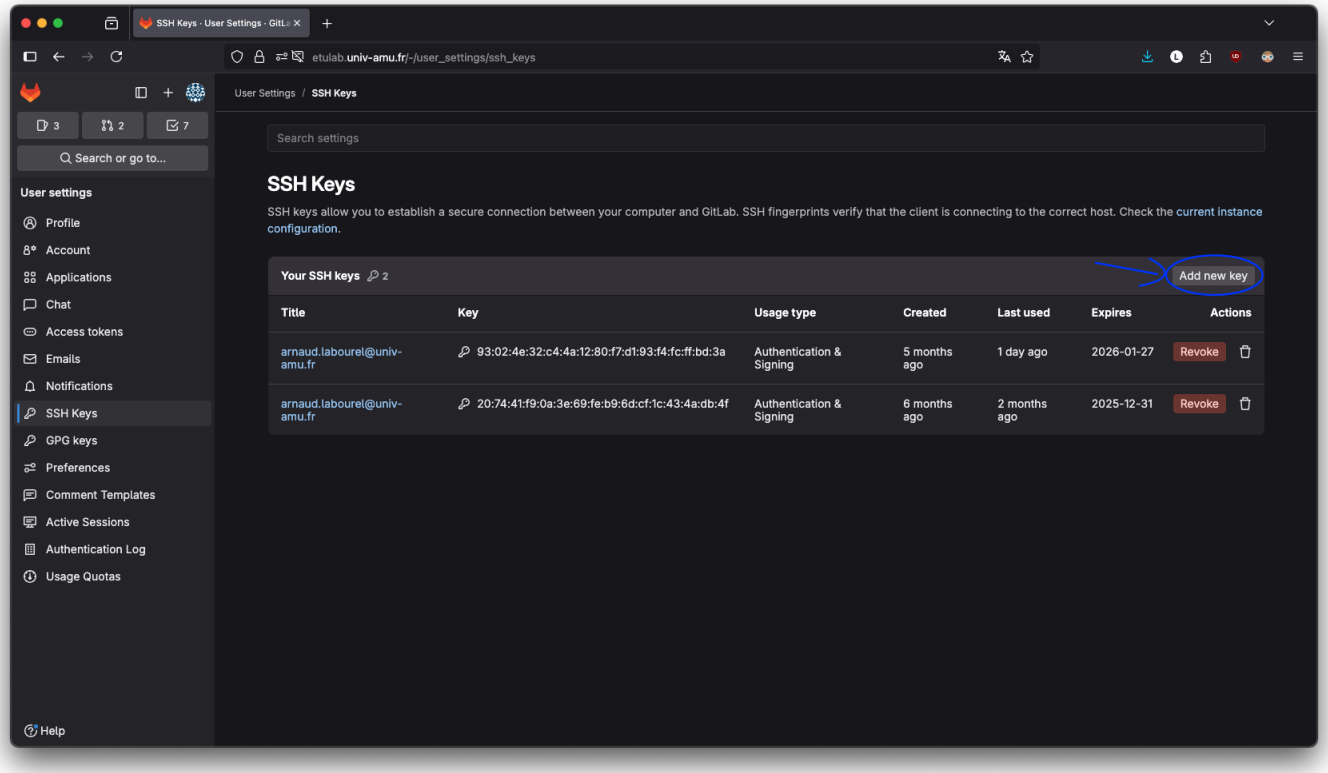
3. Configurer la clé dans votre compte gitlab. Pour cela, vous allez accéder aux préférences de votre compte gitlab. Il vous faut vous connecter à https://etulab.univ-amu.fr/ puis cliquez sur votre avatar en haut à gauche de l'écran puis sur *preferences* dans le menu qui apparaît.



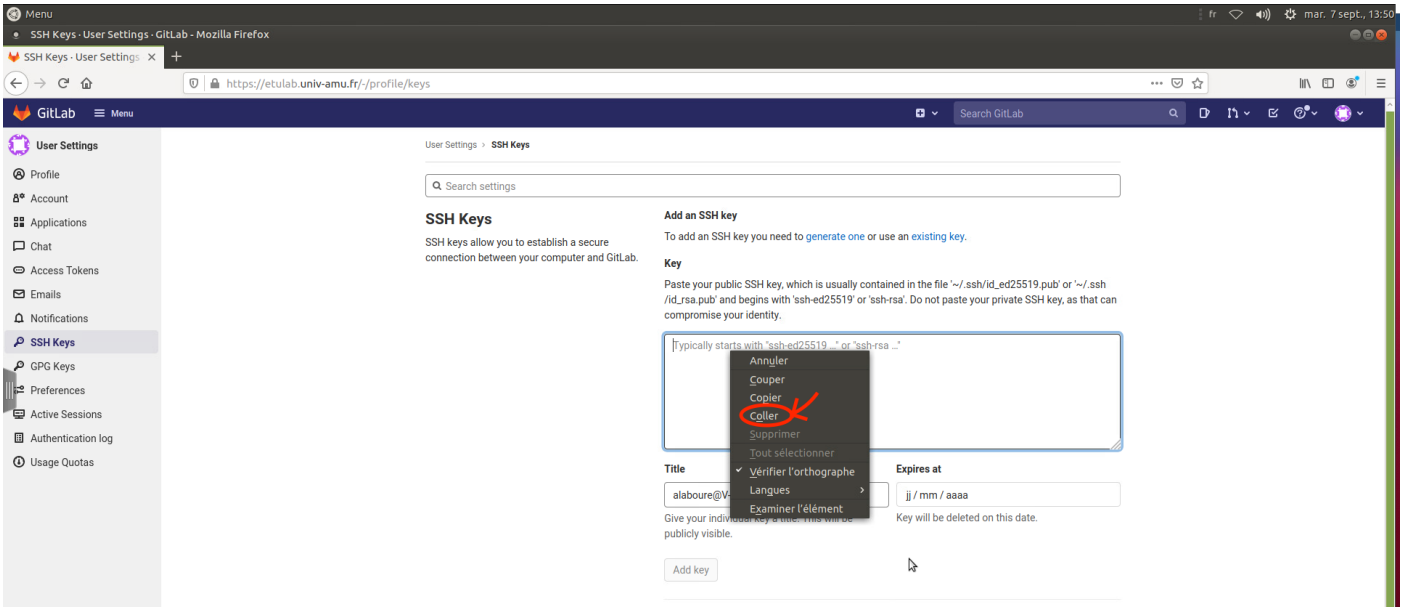

4. Allez au menu des clés SSH en cliquant sur le lien correspondant dans le menu de gauche.



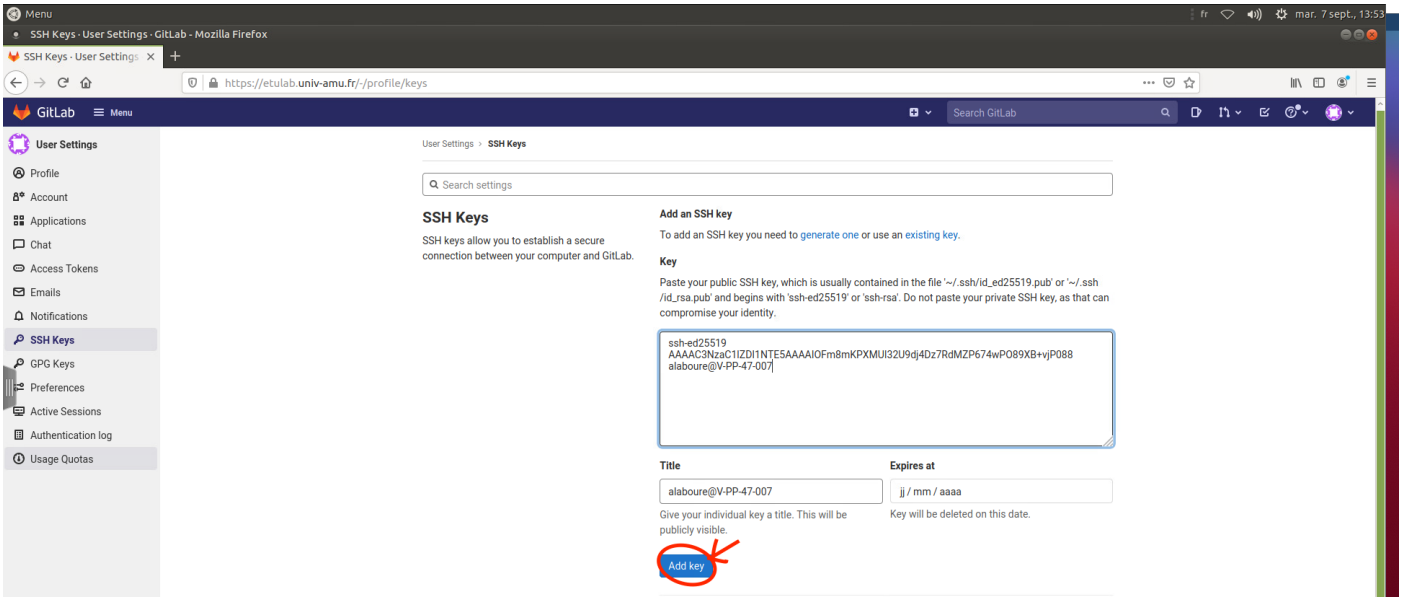

5. Cliquez sur le bouton `add new key`




6. 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*.



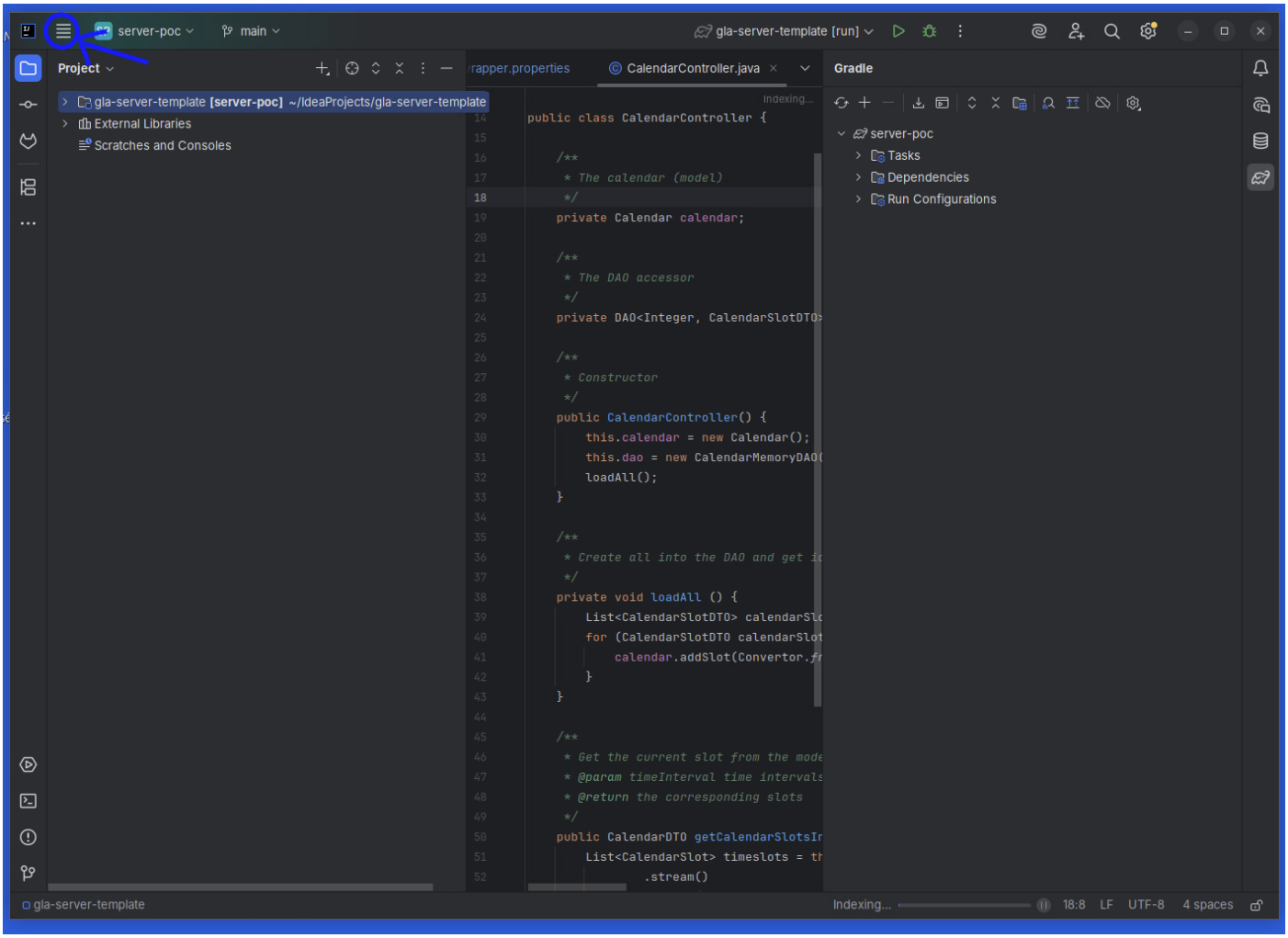

Vous devriez recevoir un mail sur votre mail étudiant confirmant que vous avez rajouté une clé. Vous pouvez rajouter autant de clés que vous voulez. Vous pouvez donc faire de même pour rajouter par exemple des clés supplémentaires si vous souhaitez accéder à votre dépôt depuis chez vous.

> **Tâche 2 :** créez votre projet en suivant soit les instructions pour le faire avec IntelliJ IDEA ou bien en ligne de commandes.

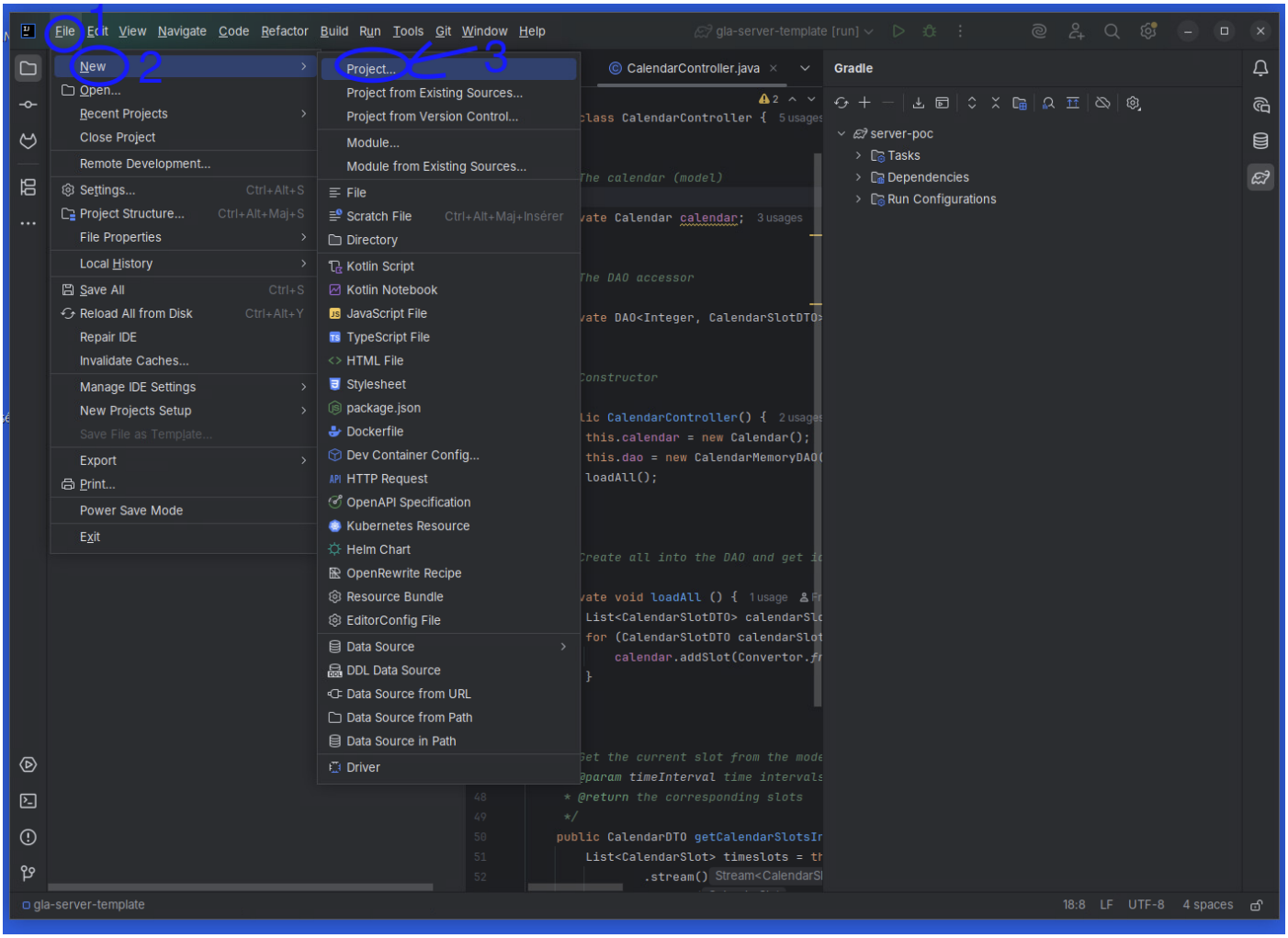
## 3.2 Création de projet à l'aide d'IntelliJ

### 3.2.1 Création du dépôt local

1. Lancez IntelliJ IDEA
2. Accédez au menu en cliquant sur l'icône en haut à gauche de la fenêtre :




3. Commencez la création d'un nouveau projet via le menu `file` -> `New` -> `Project` :




4. Créer un projet pour le TP. Pour ce faire, il vous faut (dans l'ordre du haut vers le bas) :

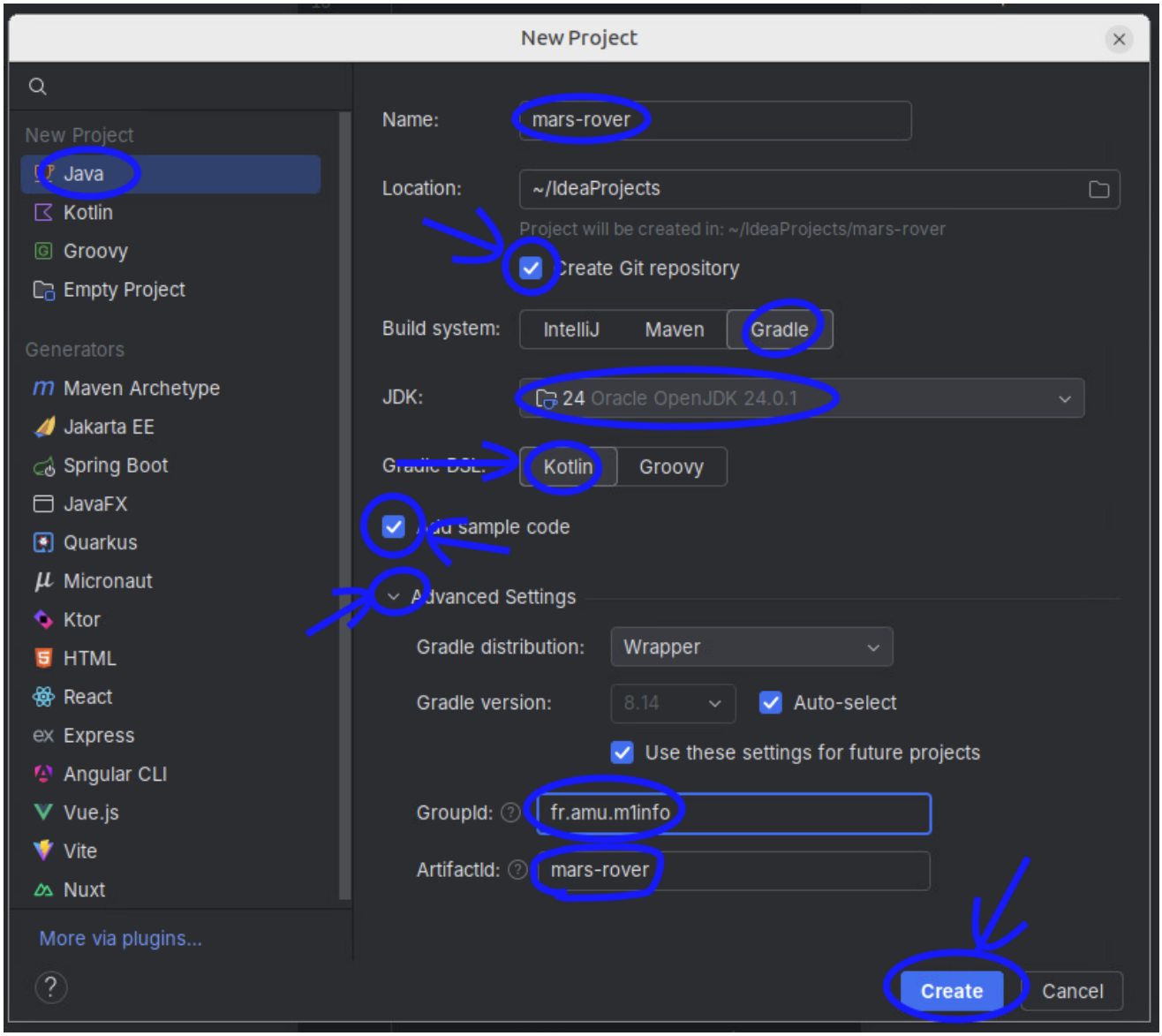
— Sélectionner dans la colonne de gauche *Java* comme type de projet
— choisir un nom (*name*) pour votre projet, pour ce projet, le nom sera `mars-rover` ;
— choisir le répertoire de stockage (*Location*) du projet (vous pouvez laisser le répertoire de base qui est `~/IdeaProjects/`)
— cocher la case *Create Git repository* pour créer un dépôt git local ;
— choisir un JDK de version 21 ou plus (normalement JDK 24 sur les ordinateurs de l'université) 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` ;

[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.

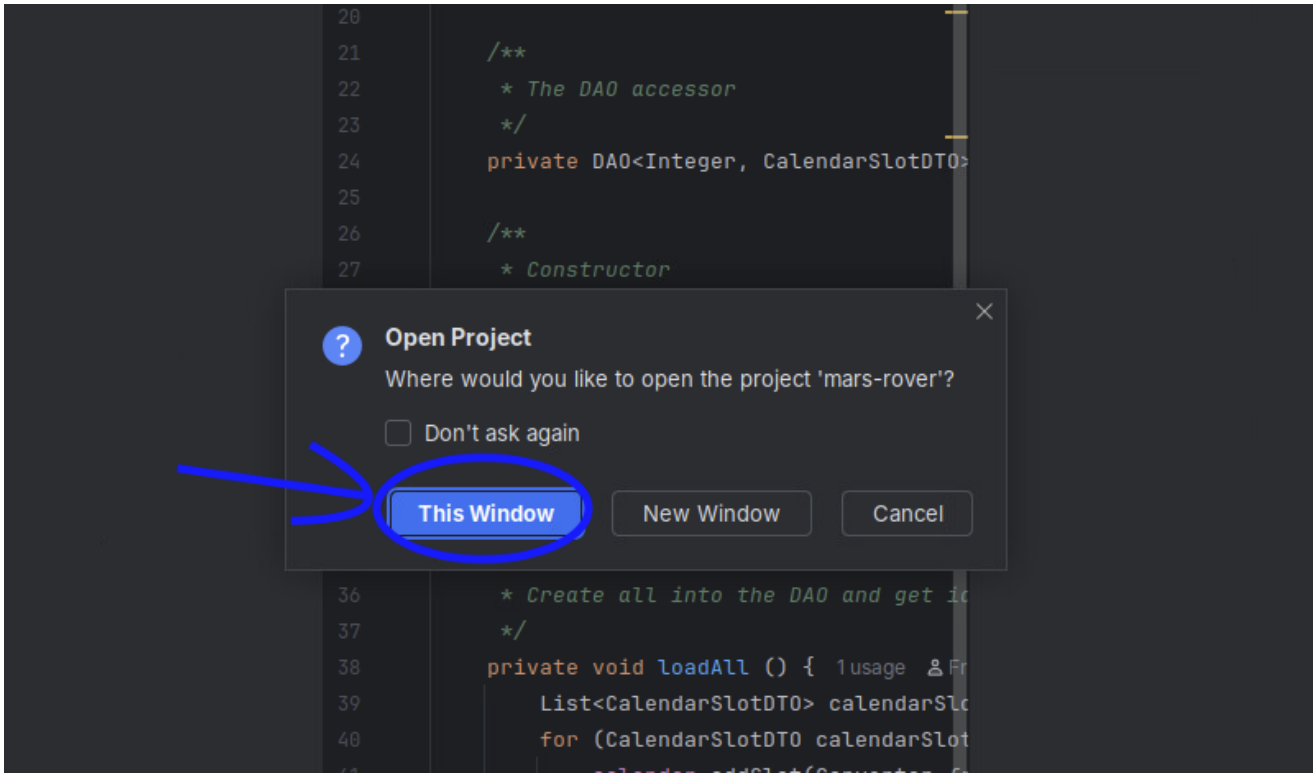
— choisir un identifiant de composant [2] (*ArtifactId*) pour votre projet, vous pouvez laisser le nom du projet (`mars-rover`).

— cliquez sur le bouton *Create*.

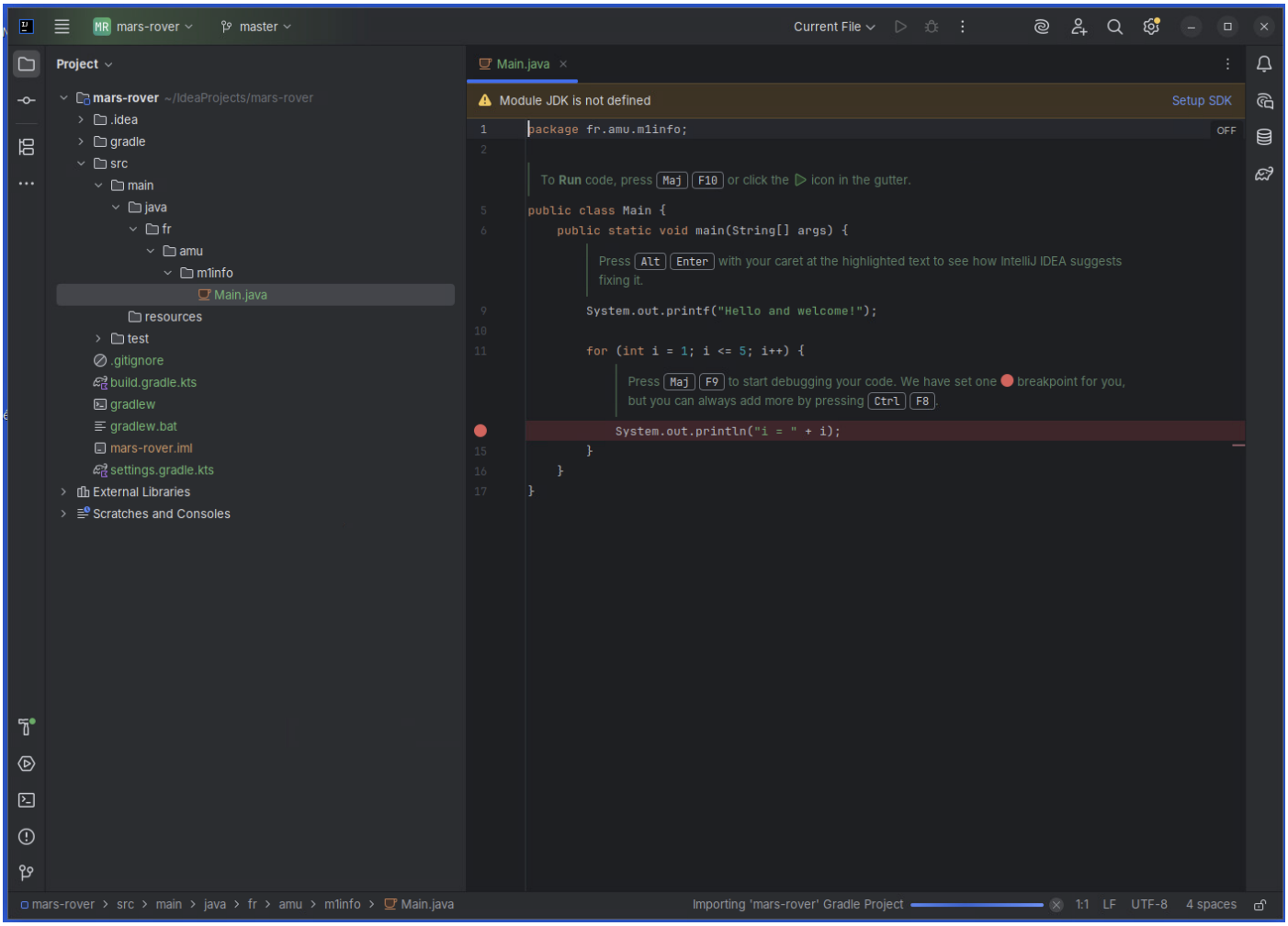



— Si vous avez déjà un projet ouvert, une fenêtre s'ouvre pour vous demander si vous souhaitez ouvrir le projet dans une nouvelle fenêtre ou non. On vous conseille d'ouvrir le projet dans la fenêtre courante en cliquant sur le bouton `This window`

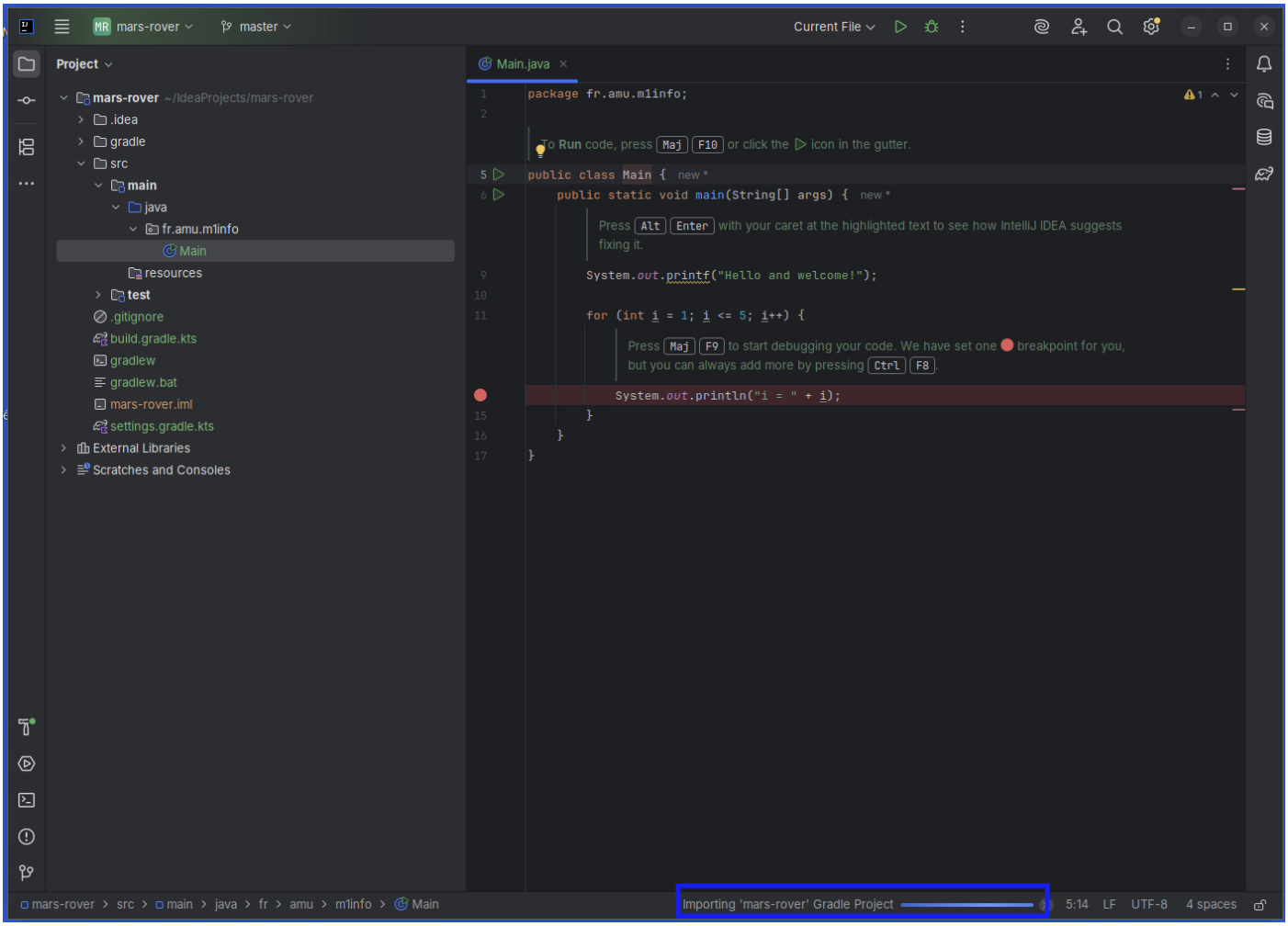
[2] L'identifiant de composant (*ArtifactId*) est le nom unique du projet au sein du groupe qui le développe. Généralement, l'identifiant du composant est le nom du projet.



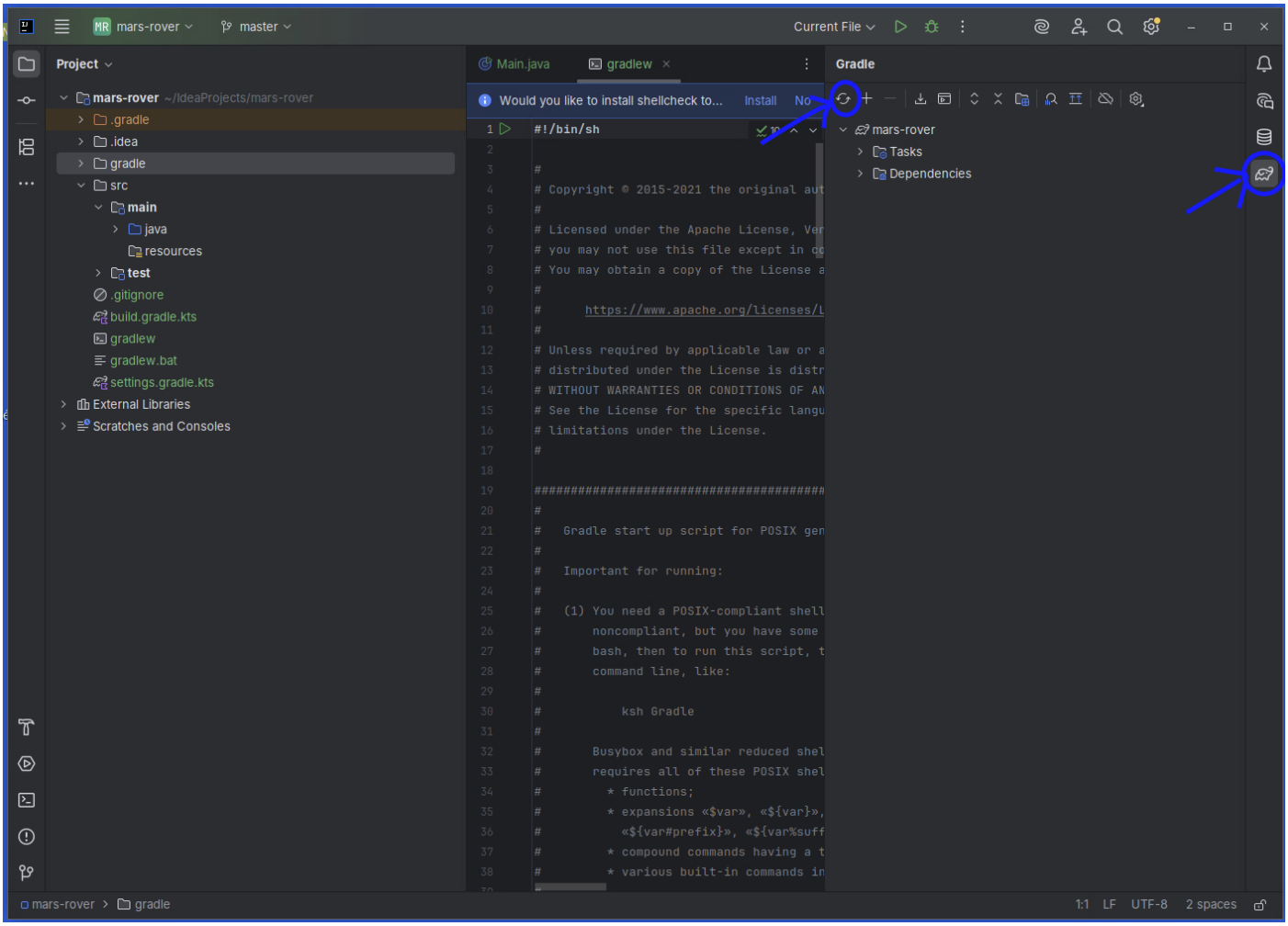

Votre projet devrait s'ouvrir et vous devriez obtenir l'affichage suivant :




Laisser le temps à votre projet de se configurer la barre en bas devrez indiquer que l'import est en cours :

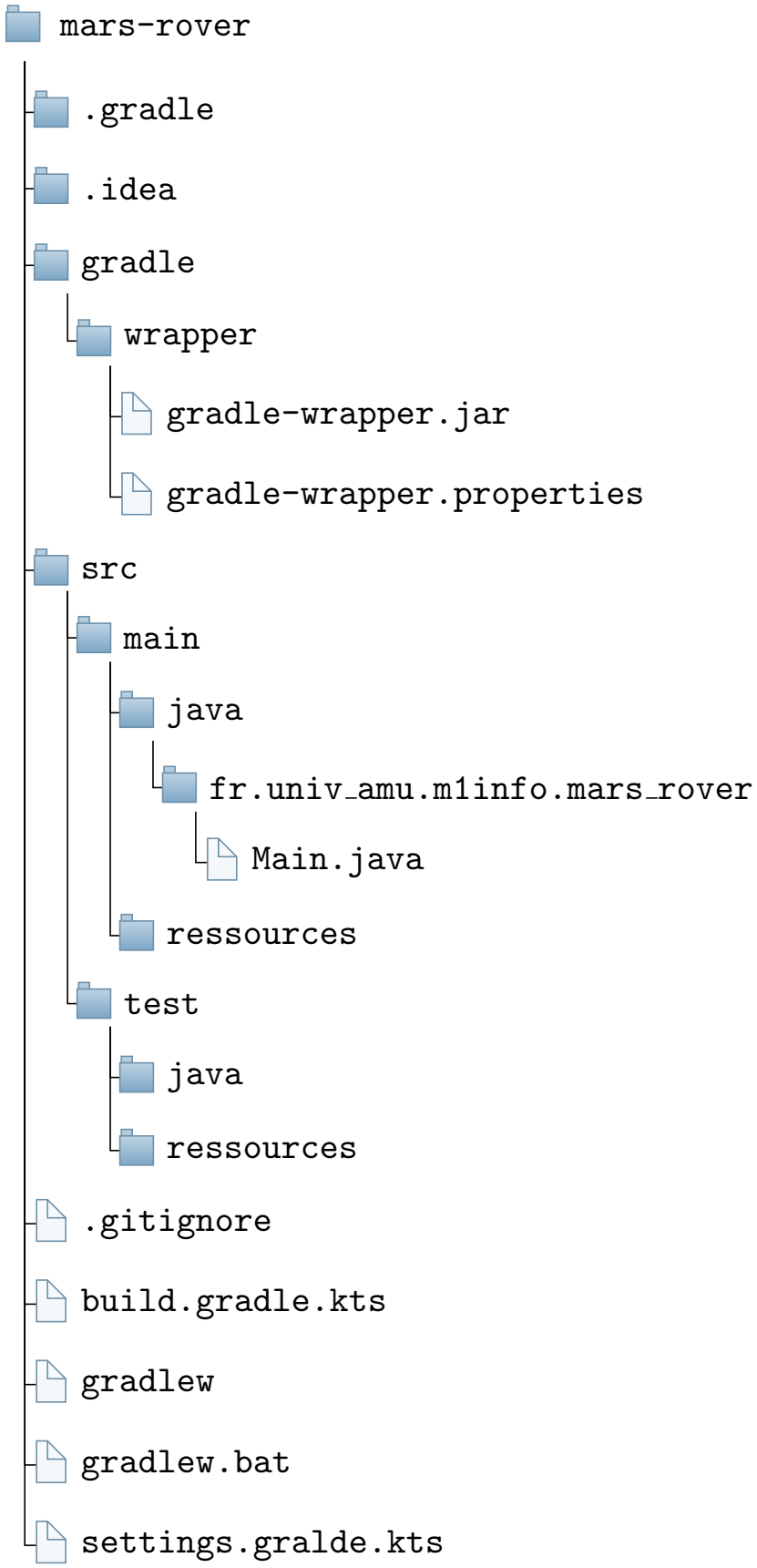



Une fois l'import *gradle* fini, vous pouvez ouvrir le menu gradle en cliquant sur l'icône éléphant à droite. Vous avez éventuellement besoin de cliquer sur l'icône avec deux flèches en haut à gauche de l'onglet *gradle* pour faire apparaitre la liste des tâches *gradle*.




### 3.2.2 Explication fichiers et répertoires du projet

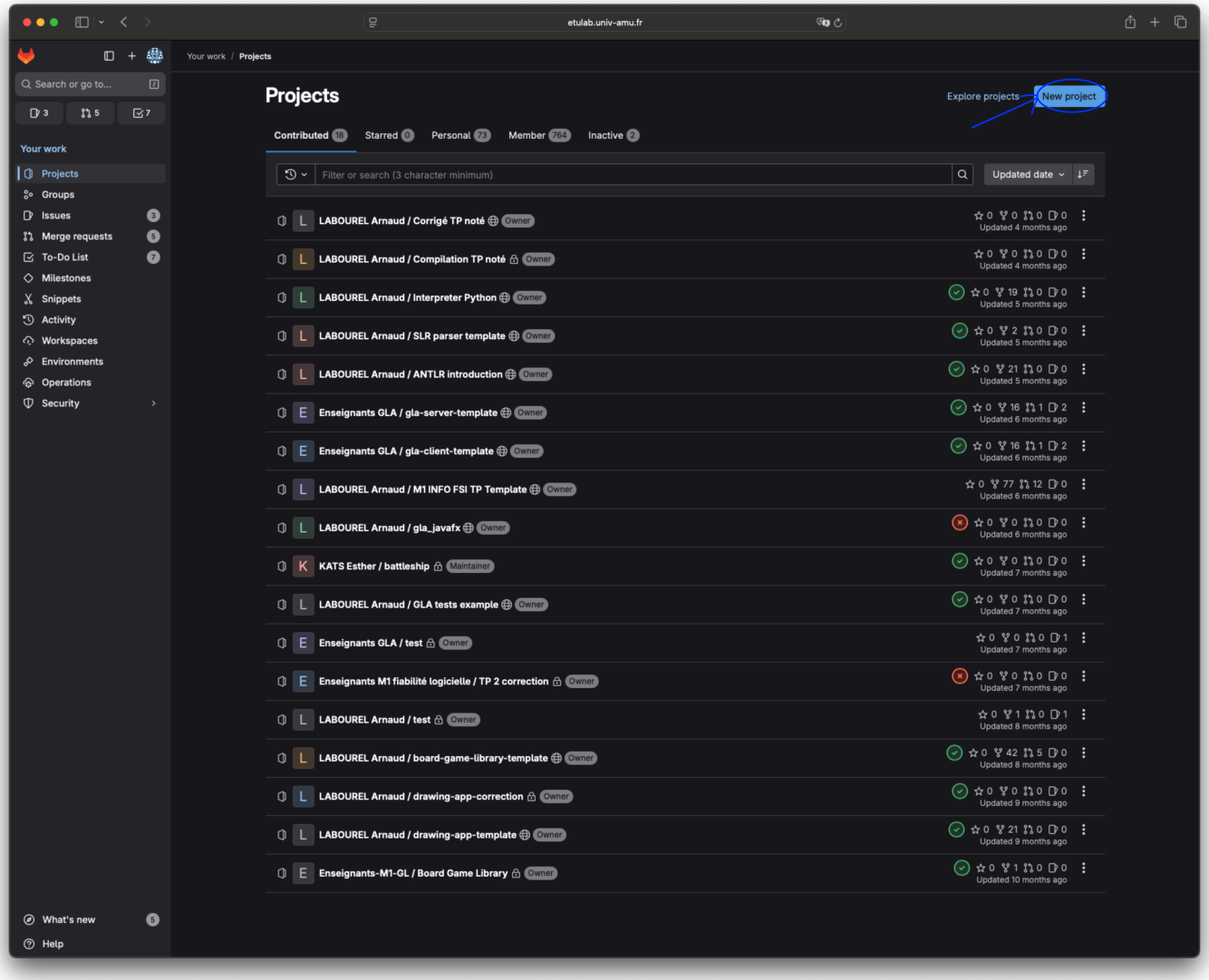
Votre projet devrait avoir l'arborescence de fichiers suivante :


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

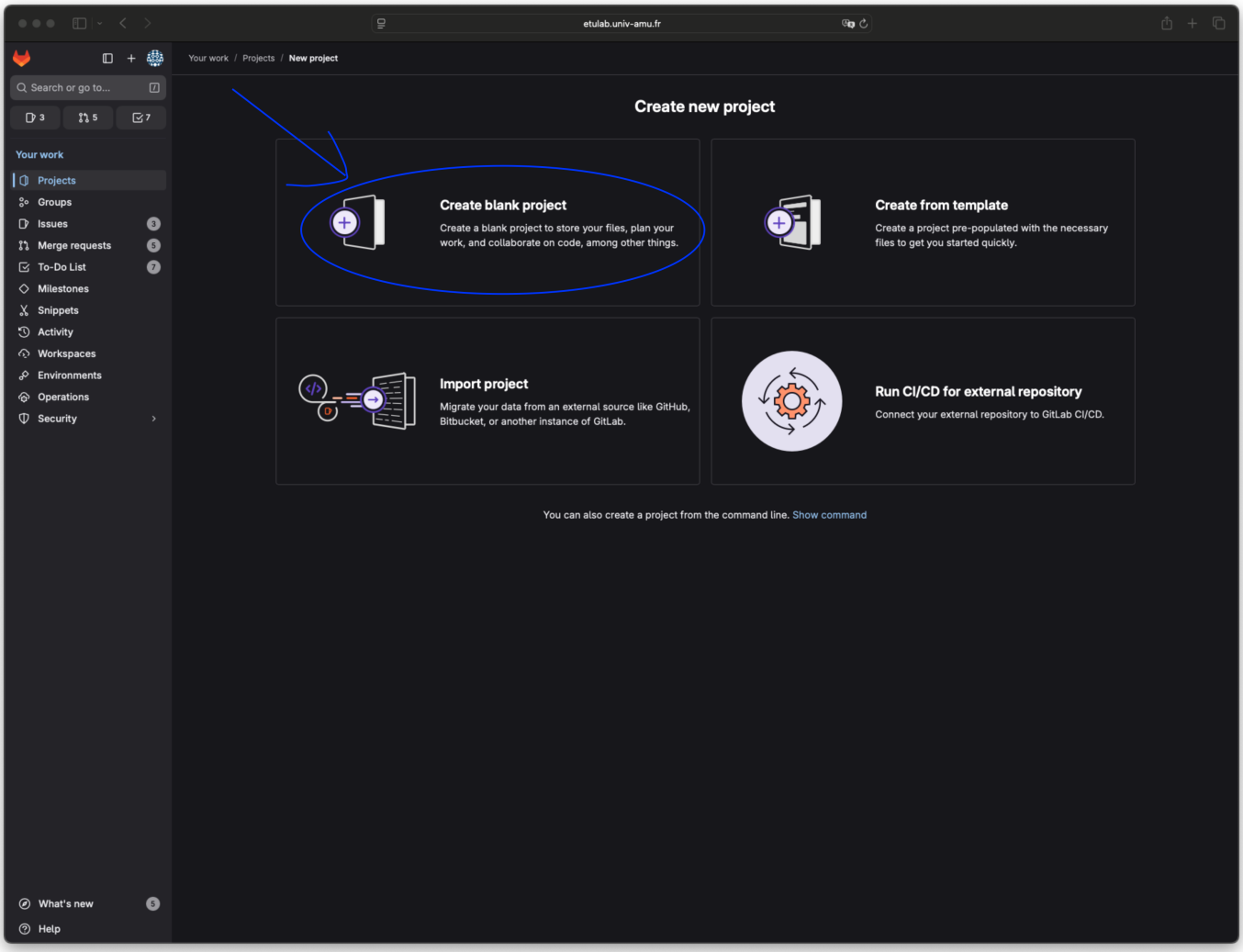
— Le répertoire `.gradle` contient les fichiers de configuration de *gradle*.
— Le répertoire `.idea` contient les fichiers de configuration d'*IntelliJ Idea*.
— Le répertoire `gradle` contient le gradle wrapper qui permet de fixer la version de *gradle* (la téléchargeant au besoin).
— Le répertoire `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

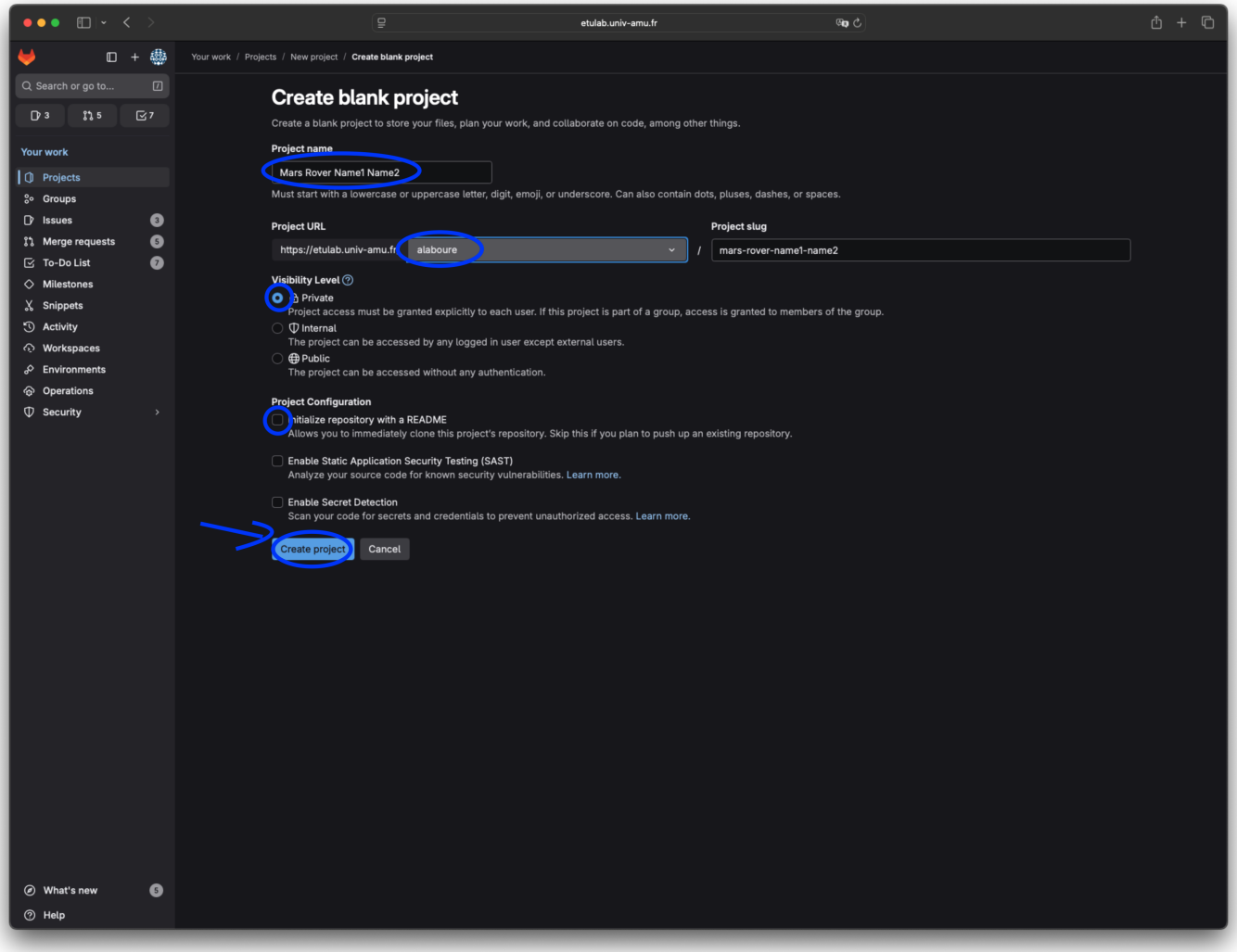
— Connectez-vous via à un navigateur à votre compte etulab.
— Commencez la création d'un nouveau projet en cliquant sur `menu` puis `Create new project`.



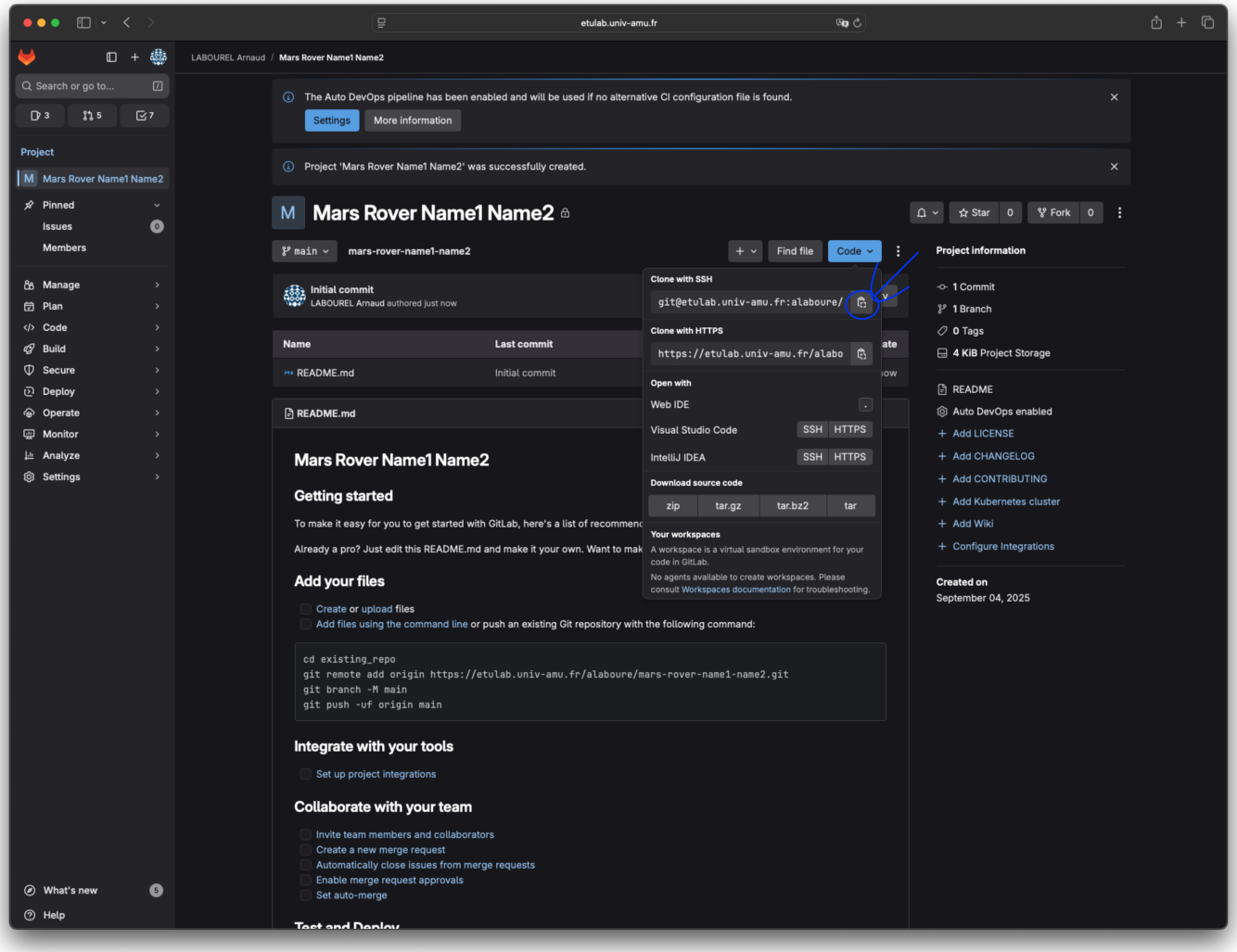

— Choisissez `Create blank project`.



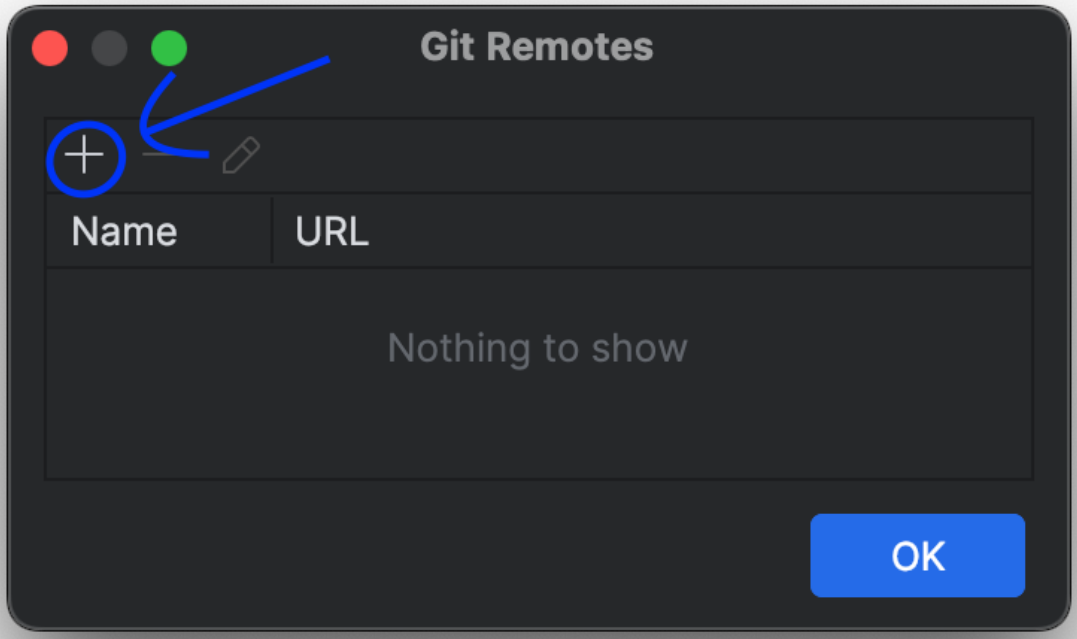

— Remplissez le formulaire de la façon suivante :
  — Choisissez comme nom de projet (*project name*) `Mars Rover` suivi par les noms complets (nom suivi du prénom) du ou des deux étudiants dans le projet ;
  — Indiquez comme *namespace* le *namespace* correspondant à votre compte (numéro d'étudiant précédé de la première de votre nom) ;
  — Vérifier que le *project slug* est `mars-rover` suivi du nom du ou des étudiants du projet séparés par des tirets ;
  — Décochez la création du `README.md` (case à côté de *Initialize repository with a README*) ;
  — Validez la création en cliquant sur le bouton `Create project`.



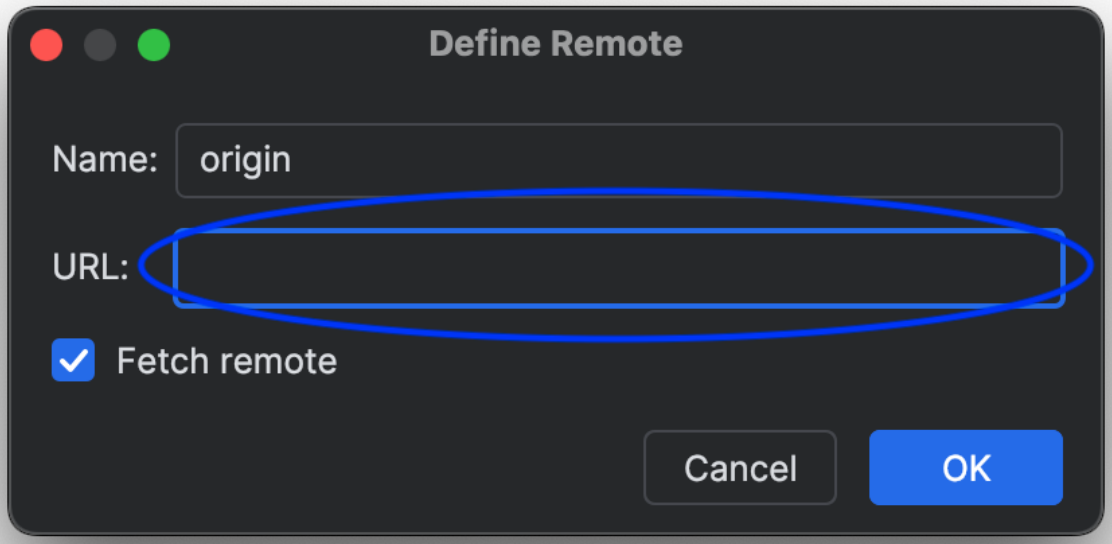

— Une fois votre projet créé, copiez l'adresse pour cloner votre projet en cliquant sur le bouton `code` puis sur l'icône de copie `ssh`.




— 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.



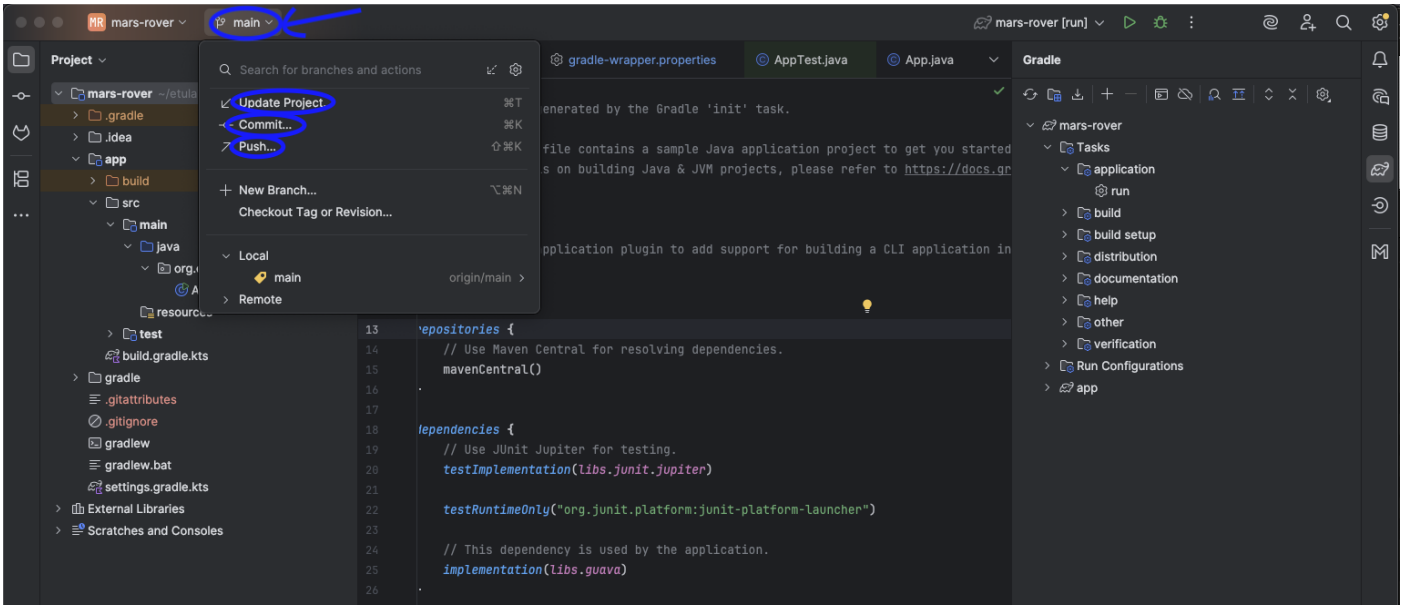

— Collez l'adresse de votre dépôt distant dans le champ `URL` et validez l'ajout en cliquant sur le bouton `OK` deux fois (une fois pour la fenêtre `Define remote` et une autre fois pour la fenêtre `Git remotes`).



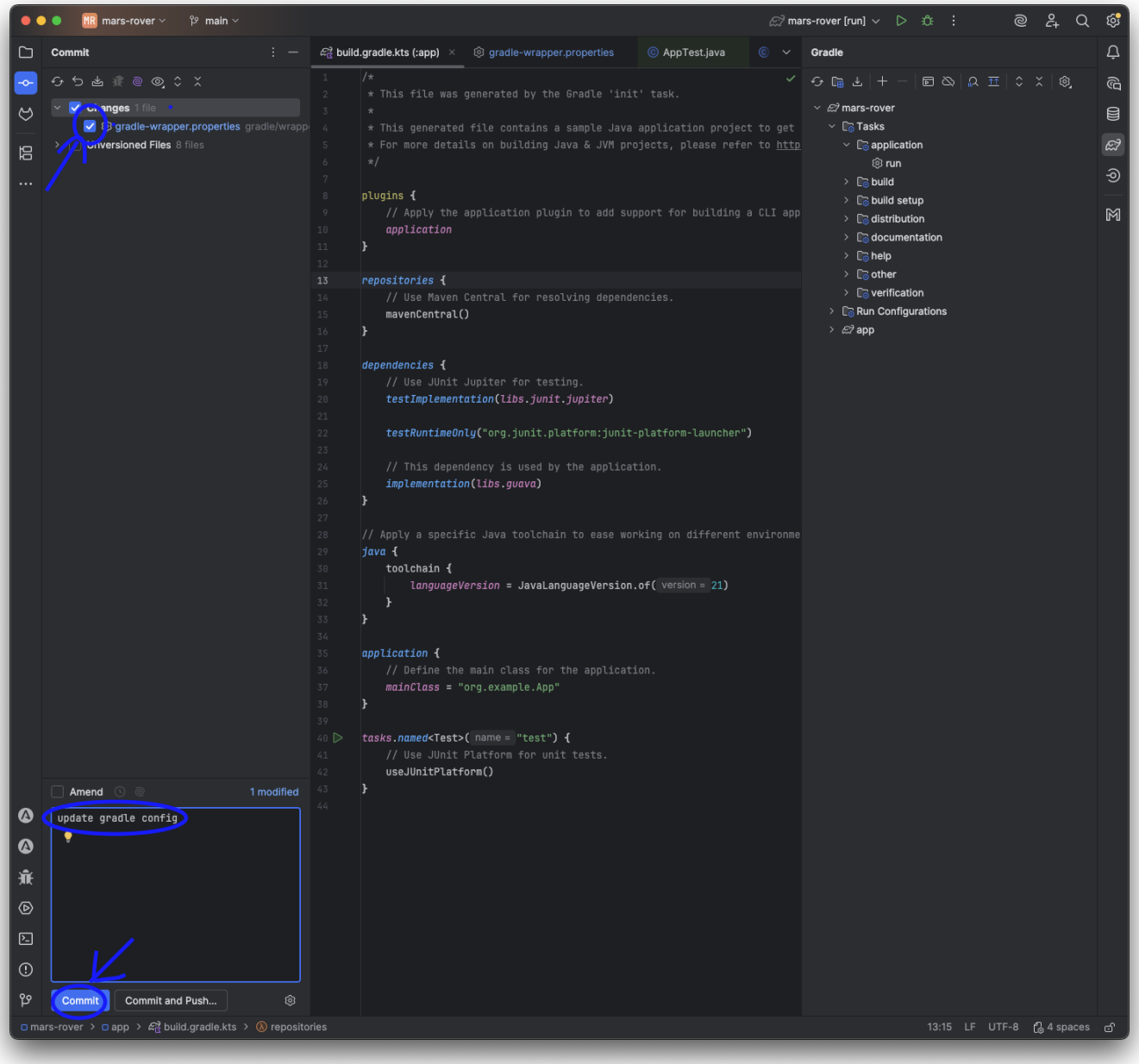

### 3.2.4 Opérations de base de *git* sous IntelliJ IDEA

Une fois votre projet créé, vous avez à partir du menu dédié les trois opérations de base de *git* :

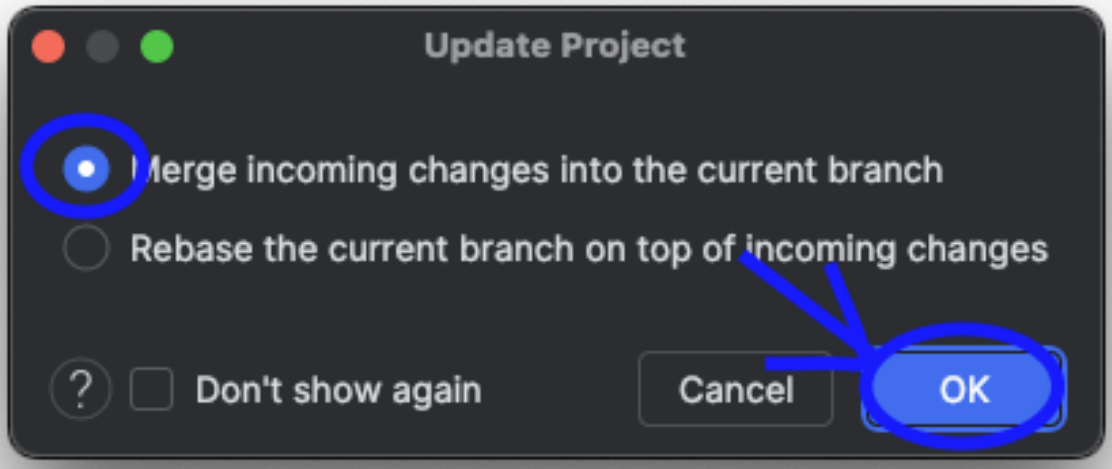
- *commit* pour créer une mise à jour du code avec un message
- *update project* pour mettre à jour la version locale du projet à partir de celle du serveur (récupère les *commits* du serveur)
- *push* pour mettre à jour la version du serveur à partir de la version locale (pousse les *commits* sur la version du serveur)



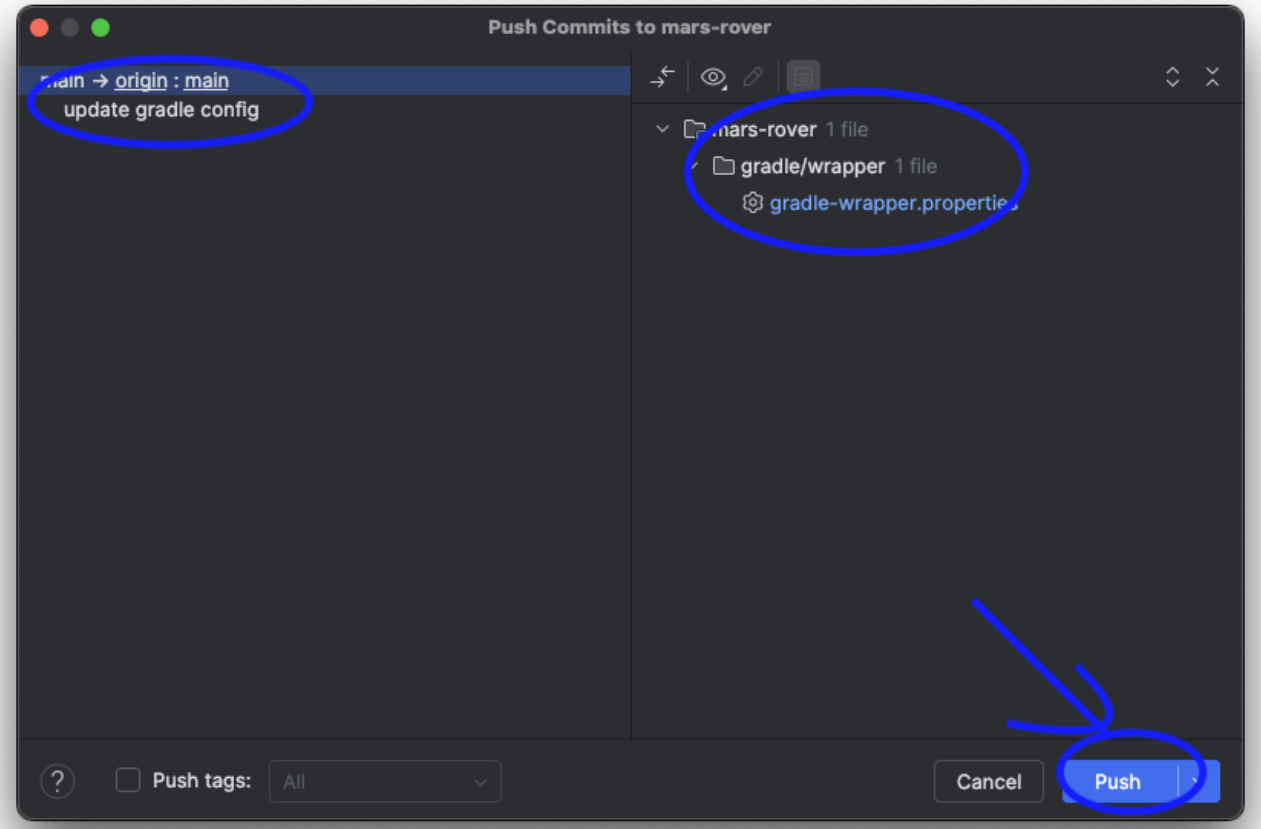

— Pour faire un *commit*, il vous suffit de :
  — cocher les fichiers que vous voulez modifier (*changes*) ;
  — de donner un message explicite des modifications faites dans le code (texte en bas) ;
  — et finalement de valider en cliquant sur le bouton *commit*.




— pour faire un *update project*, il vous suffit de :
  — de choisir entre *merge* et *rebase* (généralement vous pouvez vous contenter du *merge*) :
    — Le *merge* préserve l'historique complet du développement en créant de nouveaux *commits* qui combinent les branches sans modifier les commits existants.
    — Le *rebase* réécrit l'histoire en rejouant les *commits* d'une branche sur une autre.
  — de donner un message explicite des modifications faites dans le code (texte en bas) ;
  — et finalement de valider en cliquant sur le bouton *ok*




— pour faire un *push*, il vous suffit de :
  — de vérifier que la liste à gauche contient bien tous les messages des commits que vous souhaitez pousser sur le serveur ;
  — de vérifier que la liste à droite contient les fichiers modifiés par les commits
  — et finalement de valider en cliquant sur le bouton *push*.




## 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*.

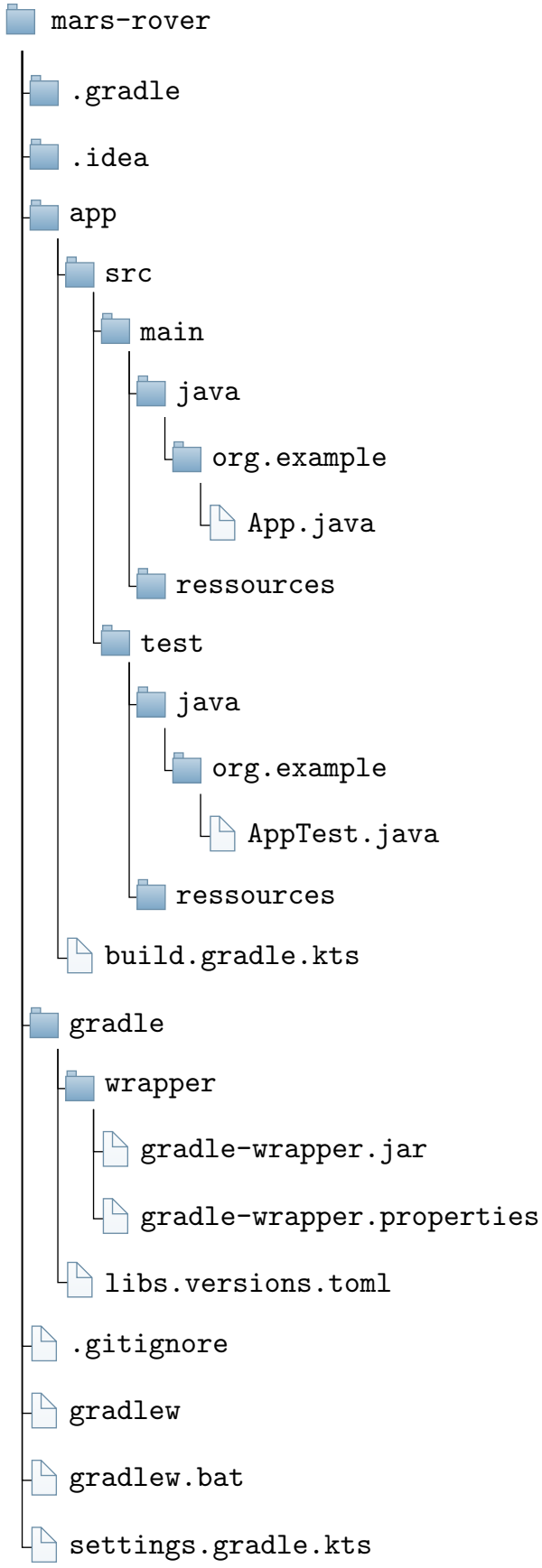
- Créez le répertoire qui va contenir votre projet qui devra être nommé `mars-rover`.
- Pour commencer la création de votre projet lancez la commande `gradle init` alors que vous êtes dans le répertoire `mars-rover`.
  - On vous demande tout d'abord quel type de projet vous souhaitez créer. Tapez `1` 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

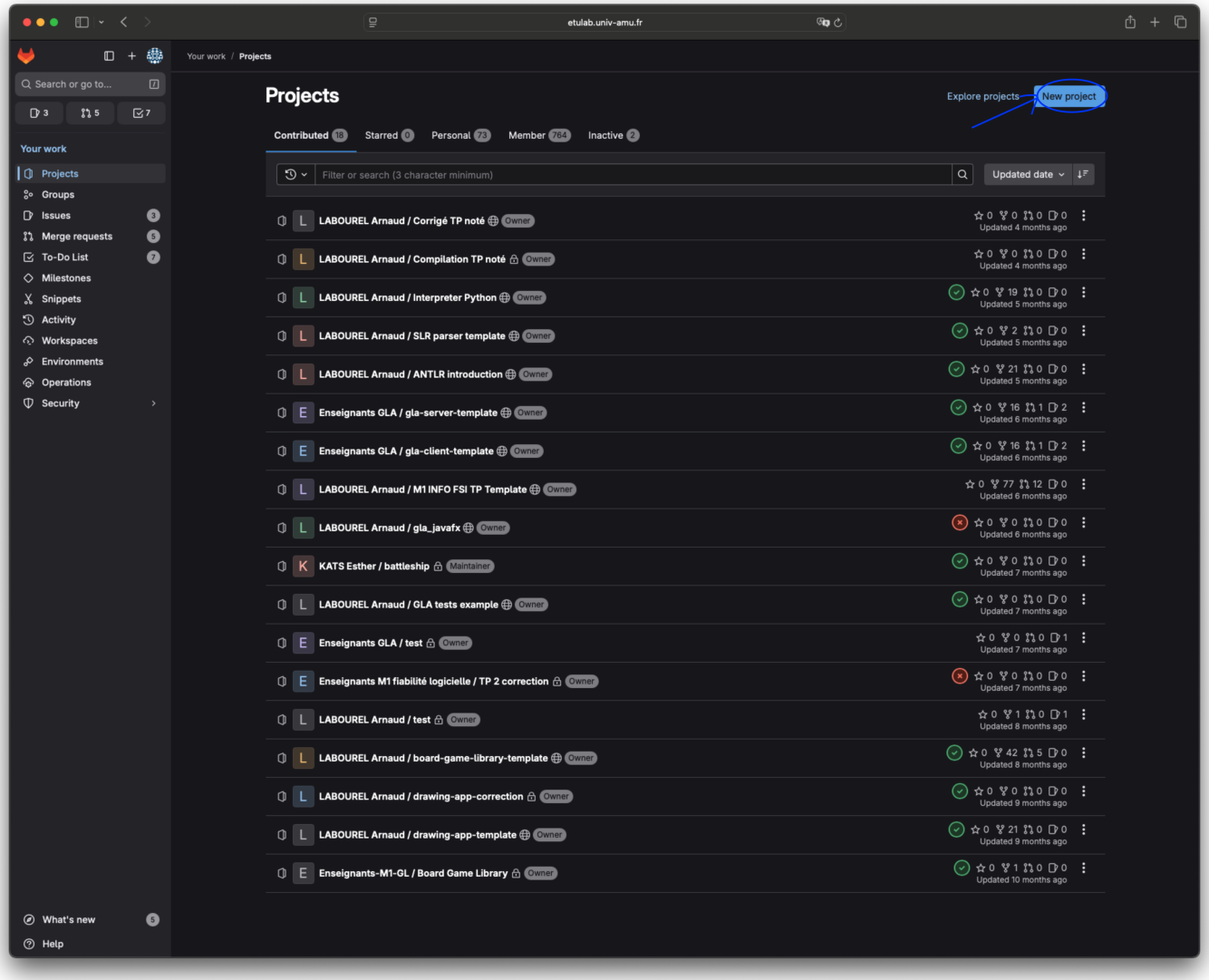
— Le répertoire `.gradle` contient les fichiers de configuration de *gradle*.
— Le répertoire `.idea` contient les fichiers de configuration d'*IntelliJ Idea*.
— Le répertoire `gradle` contient le gradle wrapper qui permet de fixer la version de *gradle* (la téléchargeant au besoin).
— Le sous-répertoire `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`

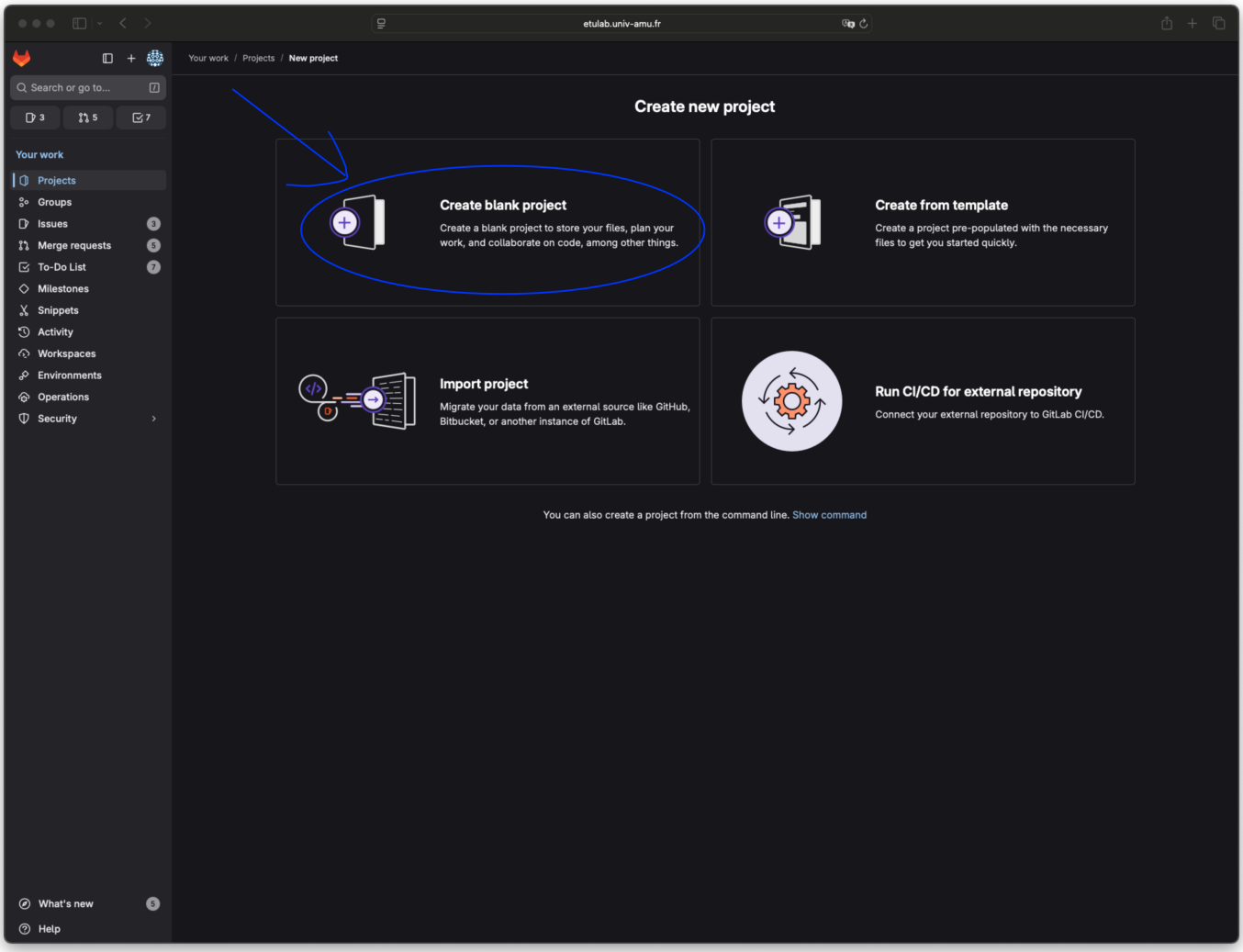
— Dans le répertoire contenant votre projet `gradle`, exécutez la commande `git init --initial-branch=main` pour créer un dépôt `git` local.
— Dans le répertoire contenant votre projet `gradle`, exécutez la commande `git add non_de_fichier` pour ajouter les fichiers de votre projet à votre prochain `commit`.
— 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

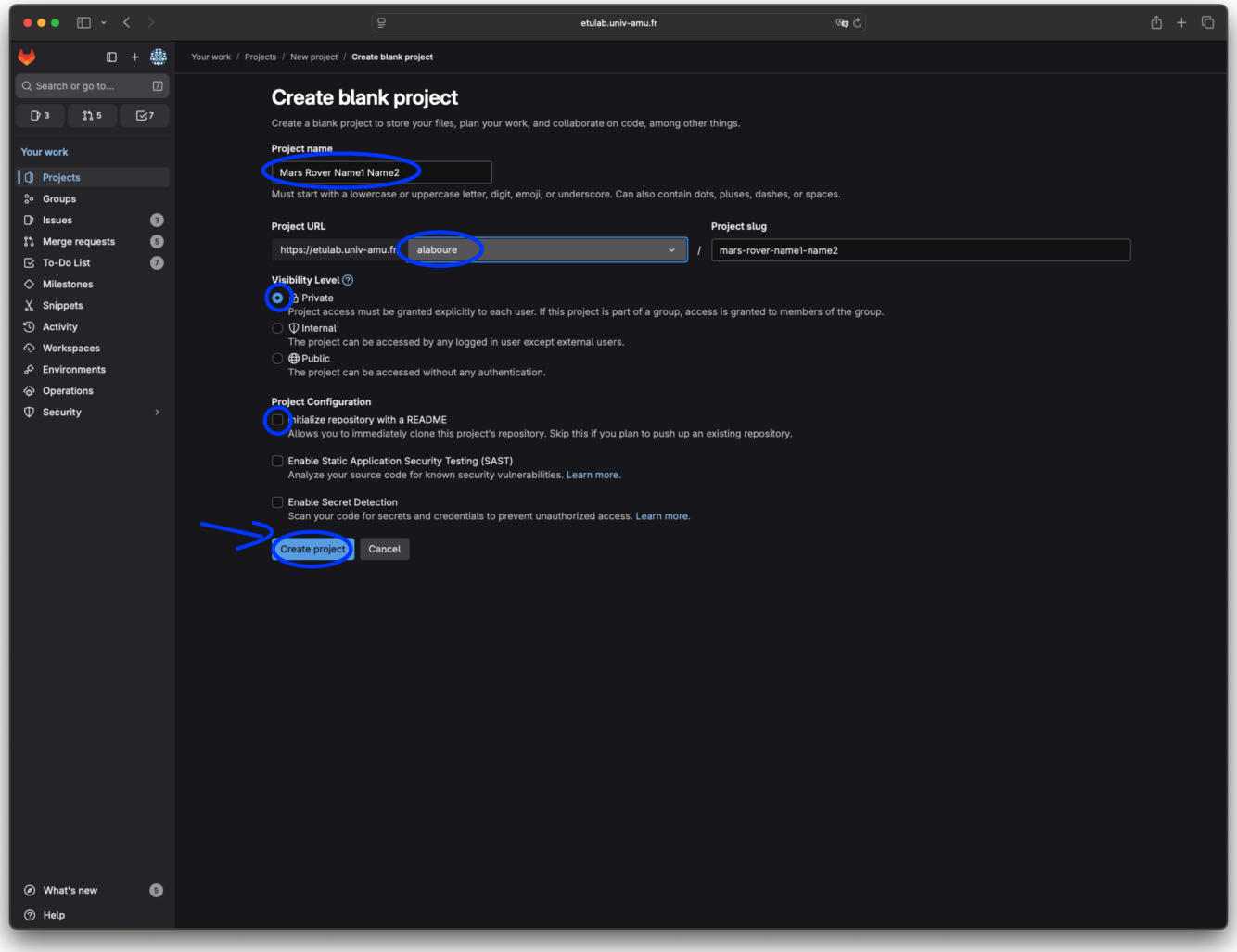
— Connectez-vous via à un navigateur à votre compte etulab.
— Commencez la création d'un nouveau projet en cliquant sur `menu` puis `Create new project`.



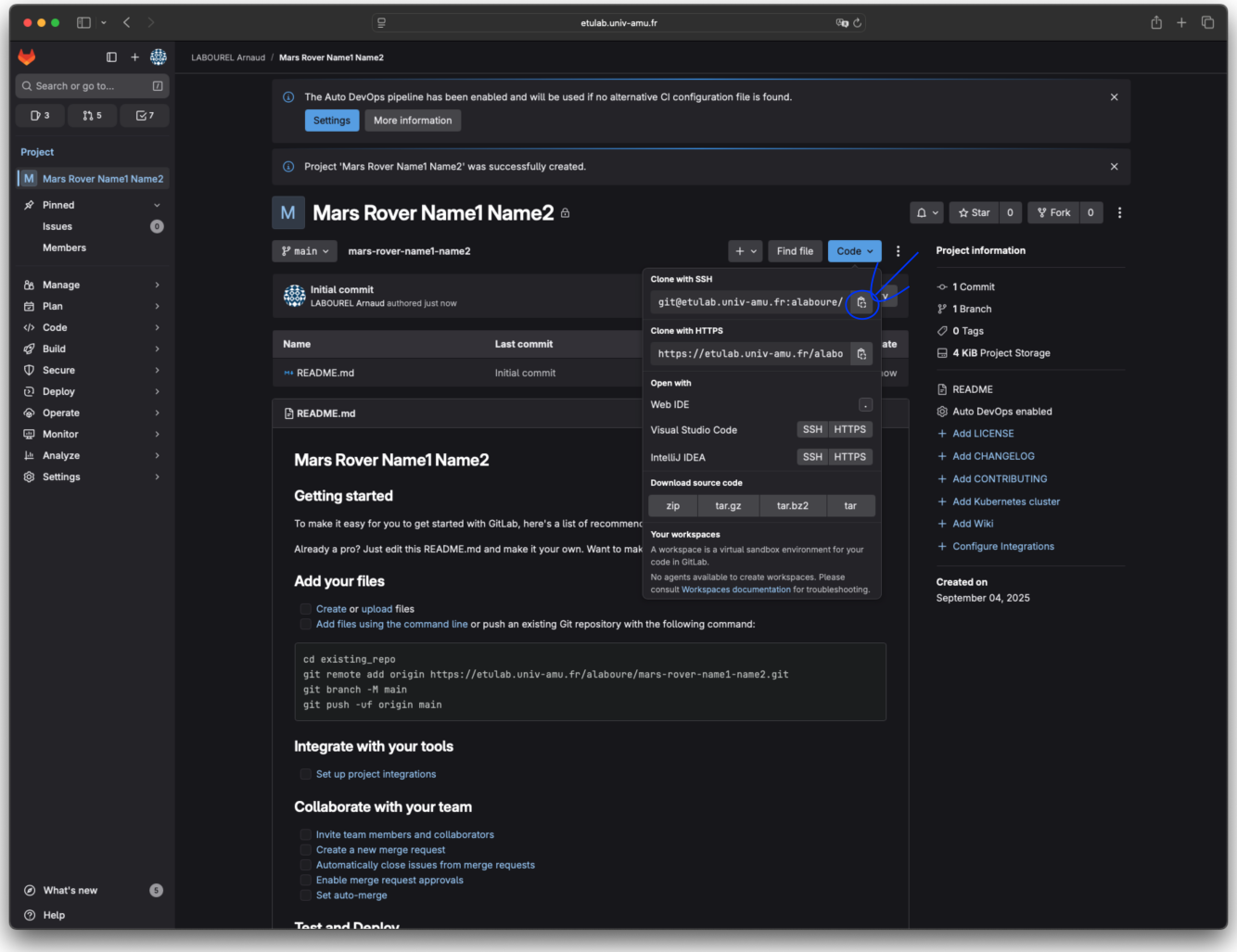

— Choisissez `Create blank project`.




— Remplissez le formulaire de la façon suivante :
  — Choisissez comme nom de projet (*project name*) `Mars Rover` suivi par les noms complets (nom suivi du prénom) du ou des deux étudiants dans le projet ;
  — Indiquez comme *namespace* le *namespace* correspondant à votre compte (numéro d'étudiant précédé de la première de votre nom) ;
  — Vérifier que le *project slug* est `mars-rover` suivi du nom du ou des étudiants du projet séparés par des tirets ;
  — Décochez la création du `README.md` (case à côté de *Initialize repository with a README*) ;
  — Validez la création en cliquant sur le bouton `Create project`.




— Une fois Copiez l'adresse pour cloner votre projet en cliquant sur le bouton `code` puis sur l'icône de copie `ssh`.

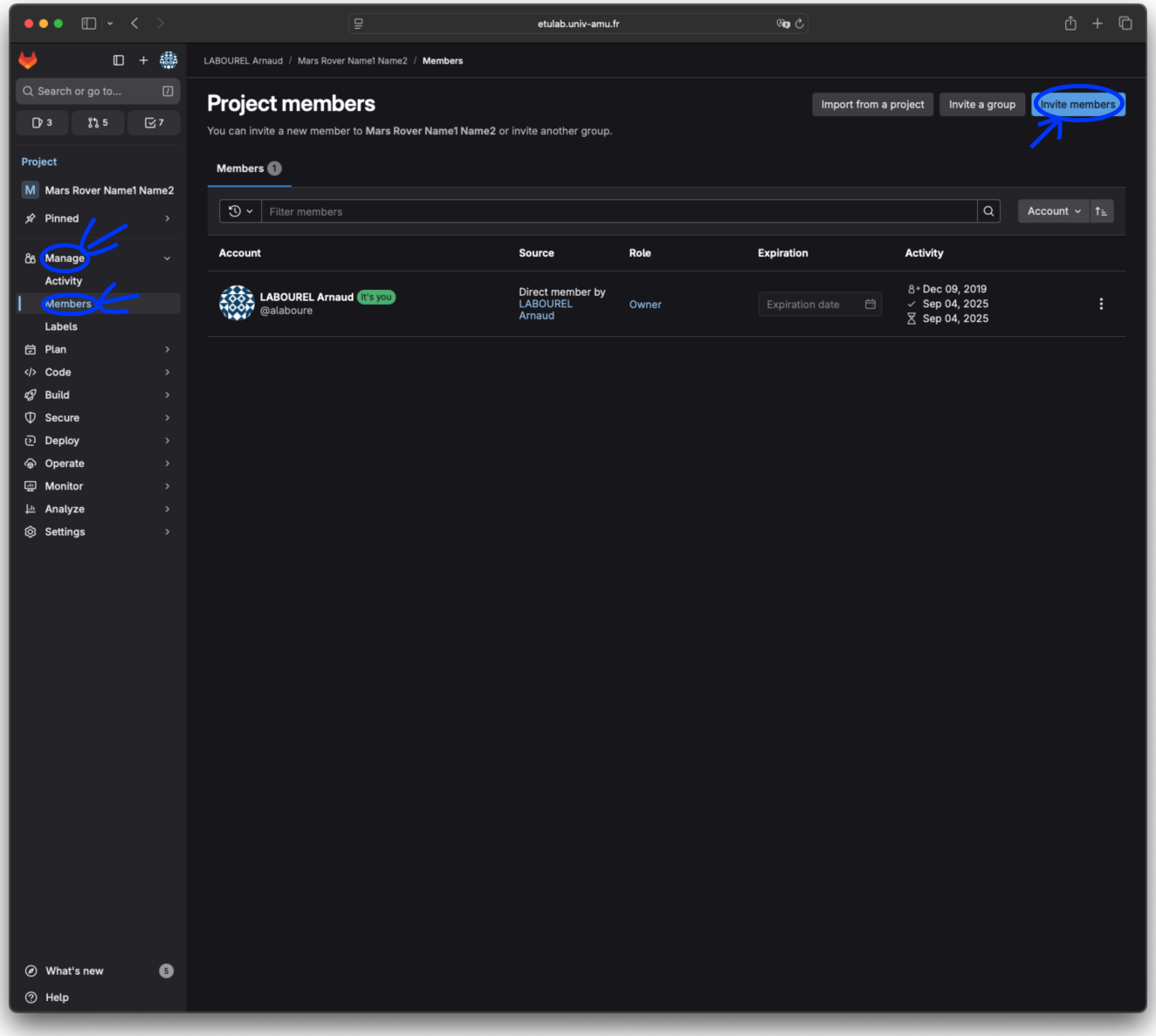



— Utiliser la commande suivante pour rajouter l'adresse du dépôt distant à votre dépôt local. Il vous faudra bien évidemment remplacer le dernier argument de la commande par l'adresse de votre projet que vous avez préalablement copié : `git remote add origin adresse_projet`.

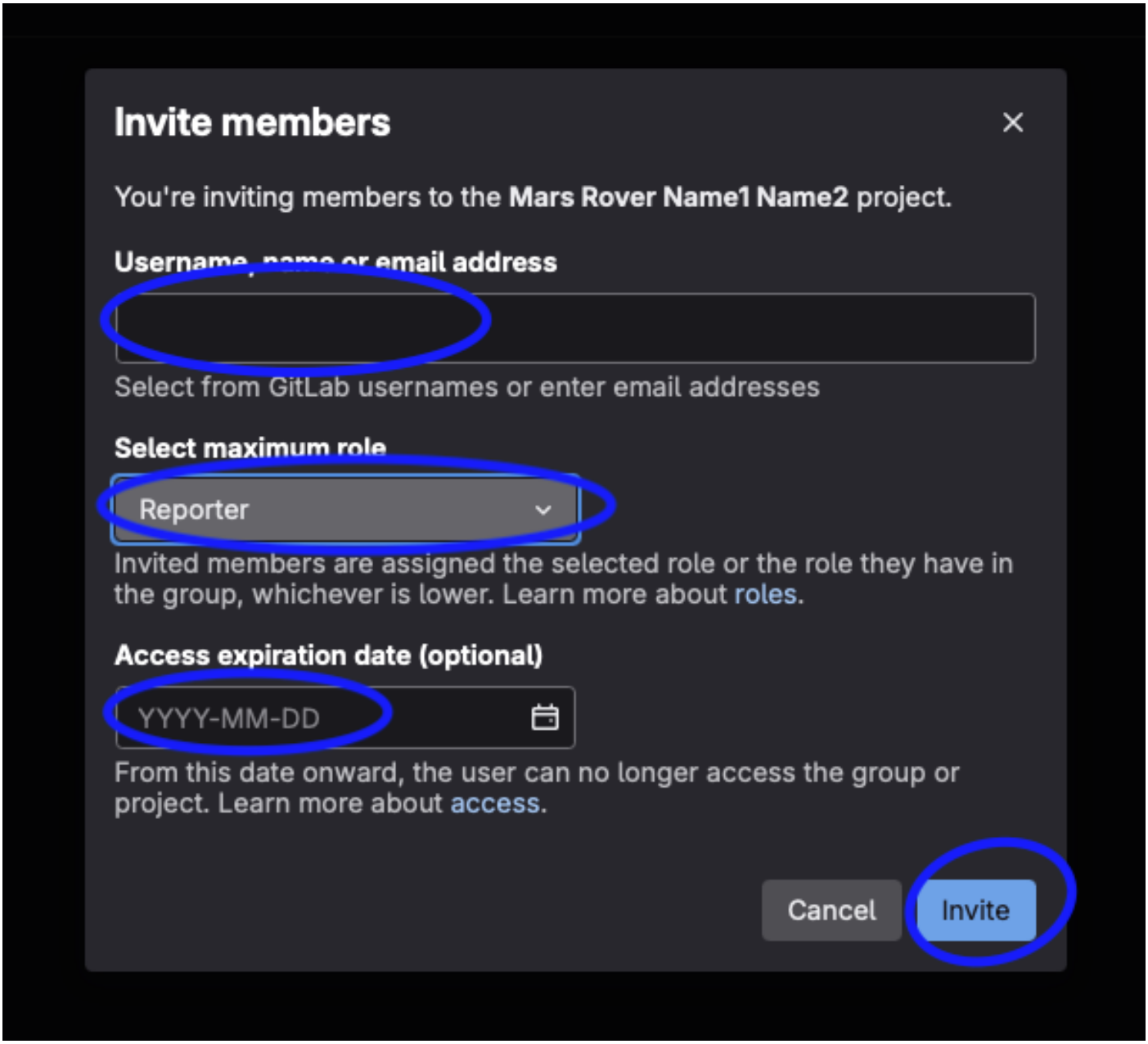
— Faites le premier `push` à l'aide de la commande `git push -u origin`. Pour les `push` suivants, vous pourrez simplement utiliser la commande `git push`, car vous avez configuré les branches.

# 4 Ajout des membres au projet

— Pour ajouter un membre dans le projet, il faut aller sur votre projet sur etulab puis cliquer sur `manage` puis `member` et finalement cliquer sur le bouton `invite members`.




— Pour ajouter un membre, il vous faut:
  — chercher un compte via le nom d'utilisateur, le nom ou l'adresse mail ;
  — choisir son rôle (*reporter* pour donner l'accès au code et *owner* pour donner les mêmes droits que le compte créateur du projet)
  — donner une date (optionnel) de fin d'accès ;
  — finalement cliquer sur bouton `invite` pour valider l'ajout du membre.




**Tâche 3 :** Ajouter comme membre au projet :

- votre binôme (si vous faites le projet à deux) avec le rôle `owner` ;
- l'enseignant Arnaud Labourel (login `@alaboure` et mail : `arnaud.labourel@univ-amu.fr`) avec le rôle `reporter`.

## 5 Spécifications Mars rover

Maintenant que vous avez mis en place votre environnement de travail avec de la gestion de version, il est temps de s'attaquer au code. Vous trouverez ci-dessous la description des spécifications du code demandé.

Lors de l'écriture du code, on vous demande de respecter les consignes suivantes :

- Effectuer des *commits* régulièrement. Toute modification significative du code (plusieurs dizaines de lignes de code) doit faire l'objet d'un commit ;
- Mettre un message de *commit* qui correspond à la modification effectuée sur le code (description de la fonctionnalité ajoutée ou du problème corrigé) ;
- Vérifier qu'après chaque *commit* le code compile ;
- Effectuer des *push* régulièrement (au moins une fois toutes les deux heures de travail).
- Ne pas mettre dans les *commits* des fichiers inutiles. Les seuls fichiers que vous devez ajouter sont les suivants :
  - les fichiers de code java dans les sous-répertoires de `src`
  - les fichiers suivants à la racine
    - `build.gradle.kts`
    - `gradlew`
    - `gradle.bat`
    - `settings.gradle.kts`
  - les fichiers dans le répertoire `gradle`.

  Tous les autres fichiers comme ceux présents dans le répertoire `.idea` ou `.gradle` ne doivent pas être ajoutés au dépôt présent sur le serveur via des *commits*

**Tâche 3 :** Produire une première version de votre code qui respecte les spécifications ci-dessous tout en suivant les instructions ci-dessus sur l'utilisation de la gestion de version.

### 5.1 Description des spécifications

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 commandes. Les commandes possibles sont :

- *tourner à gauche* : fait tourner le *rover* de 90 degrés vers la gauche (sens inverse des aiguilles d'une montre) ;
- *tourner à droite* : font tourner le *rover* de 90 degrés vers la droite (sens des aiguilles d'une montre) ;
- *avancer* : 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. On considère deux types de grille : rectangulaire ou toroïdale.

### 5.2 Grille rectangulaire

Une grille rectangulaire correspond à la grille de base représentant un rectangle. Les coordonnées $(0, 0)$ correspondent au coin sud-ouest de la grille alors que les coordonnées $(l - 1, h - 1)$ correspondent au coin nord-est de la grille ($l$ étant la largeur de la grille et $h$ sa hauteur). On suppose que tout mouvement faisant sortir le *rover* de la grille rectangulaire entraîne sa destruction et une fois détruit il n'exécutera pas les commandes suivantes.

### 5.3 Grille toroïdale

Une grille toroïdale correspond à une topologie ou le haut et le bas de la grille sont connectés et un robot sortant par le haut réapparait en bas de la grille (passant d'une coordonnée en $y$ de $h - 1$ à 0 avec $h$ la hauteur de la grille et en gardant la même coordonnée en $x$) et inversement. De même, la gauche et la droite sont connectées et un robot sortant par la gauche réapparait à la droite de la grille (passant d'une coordonnée en $x$ de 0 à $l - 1$ avec $l$ la largeur de la grille et en gardant la même coordonnée en $y$) et inversement.

Le tableau suivant donne pour une grille $4 \times 4$ ($x \in \{0, 1, 2, 3\}$, $y \in \{0, 1, 2, 3\}$) la position après un mouvement dans la grille :

| Position\Opération | NORTH | SOUTH | EAST | WEST |
|---|---|---|---|---|
| (0, 0) | (1, 0) | (3, 0) | (0, 1) | (0, 3) |
| (1, 0) | (2, 0) | (0, 0) | (1, 1) | (1, 3) |
| (1, 1) | (2, 1) | (0, 1) | (1, 2) | (1, 0) |
| (2, 0) | (3, 0) | (1, 0) | (2, 1) | (2, 3) |

## 6 Spécification des nouvelles entrées/sorties

Le format de fichier d'entrée pour la configuration des *rovers* est modifié pour être un fichier de type YAML.

Le fichier d'entrée a deux nœuds principaux :

- un nœud `grid` correspondant à la grille et donnant la largeur (`width`), hauteur (`height`) et type (`kind`) de la grille, le type de la grille peut être `RECTANGULAR` (rectangulaire), `TOROIDAL` (toroïdal) ou `SPHERICAL` (sphérique) ;
- un nœud `rovers` correspondant aux *rovers* et donnant pour chaque *rover* sous la forme d'une liste
  - sa position avec des coordonnées entières `x`, `y` et une `orientation` (`NORTH`, `EAST`, `WEST` ou `SOUTH`) et

— les commandes à exécuter sous forme d'une liste d'éléments `LEFT`, `RIGHT` ou `MOVE` (même signification que les lettres `L`, `R` et `M`).

Par exemple, une configuration correspondant à une grille rectangulaire de taille $5 \times 5$ 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 config.yml suivant :

```yaml
grid:
  width: 5
  height: 5
  kind: RECTANGULAR
rovers:
  - position:
      coordinates:
        x: 1
        y: 2
      orientation: NORTH
    commands:
      - LEFT
      - MOVE
      - LEFT
      - MOVE
      - LEFT
      - MOVE
      - LEFT
      - MOVE
      - MOVE
  - position:
      coordinates:
        x: 3
        y: 3
      orientation: EAST
    commands:
      - MOVE
      - MOVE
      - RIGHT
      - MOVE
      - MOVE
      - RIGHT
      - MOVE
      - RIGHT
      - RIGHT
      - MOVE
```

Le fichier de sortie a deux nœuds principaux :

— un nœud `percentageExplored` correspondant au pourcentage de la grille explorée au format nombre entier suivi de `%` ;
— un nœud `finalRoverStates` correspondant à la liste des positions finales des *rovers* (coordonnées entières `x`, `y` et une `orientation`)

```
percentageExplored: 28 %
finalRoverStates:
- isDestroyed: false
  position:
    coordinates:
      x: 1
      y: 3
    orientation: NORTH
- isDestroyed: false
  position:
    coordinates:
      x: 4
      y: 3
    orientation: EAST
```

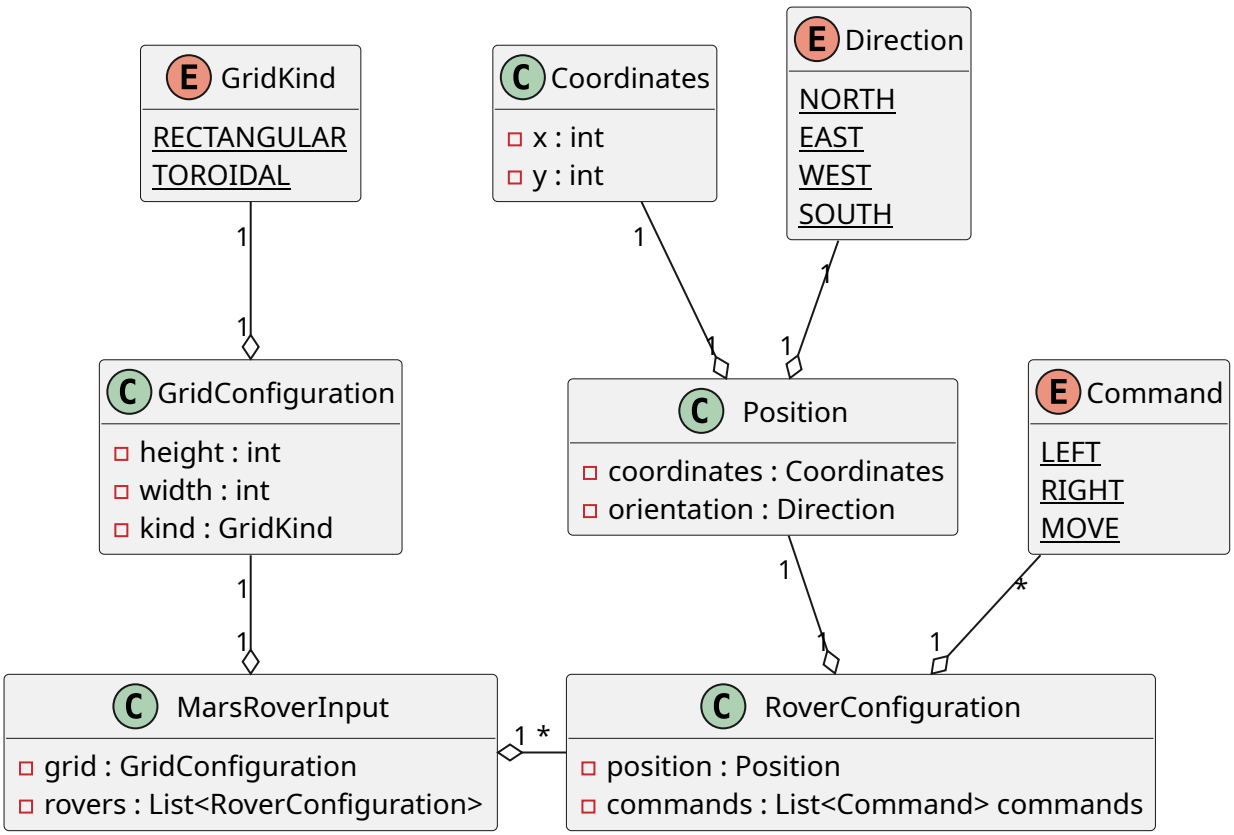
## 6.1 Utilisation de la bibliothèque Jackson

Pour la lecture des fichiers d'entrées et l'écriture des fichiers de sortie, on vous conseille fortement d'utiliser la *library* Jackson. Cette *library* permet de lire et écrire des fichiers au format *YAML* (et aussi *json*). Pour pouvoir l'utiliser, il vous suffit de rajouter la dépendance ci-dessous dans le fichier `build.gradle.kts` de votre projet.

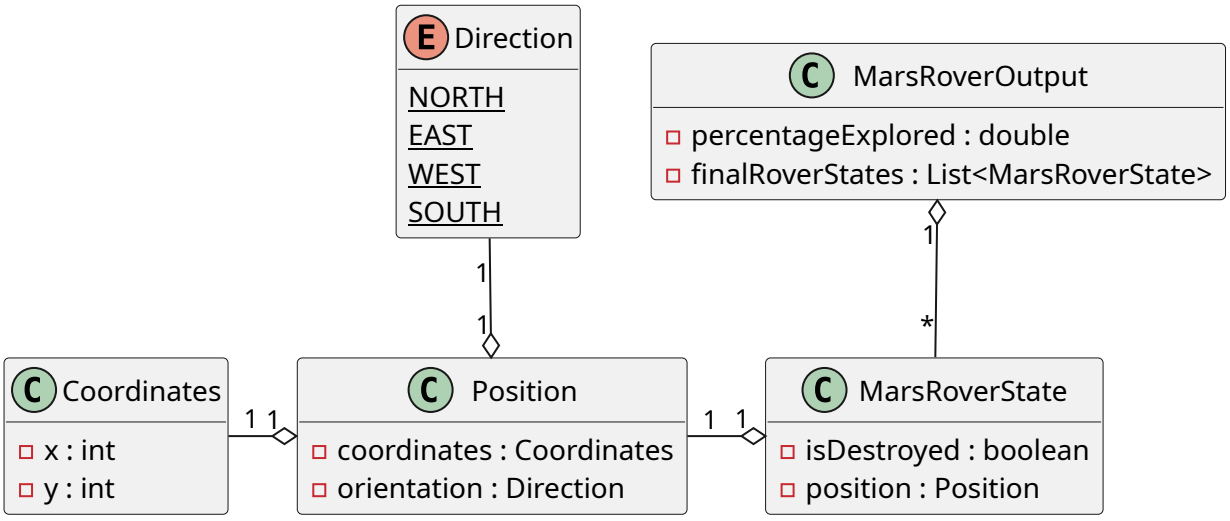
```
dependencies {
    implementation("com.fasterxml.jackson.dataformat:jackson-dataformat-yaml:2.20.0")
}
```

### 6.1.1 Lecture de fichier

Pour lire un fichier, il faut d'abord créer une structure de classes correspondant à la structure en arbre du format de fichier *YAML*. Pour le l'entrée du problème sera récupérer sous la forme d'une instance de `MarsRoverInput`. La structure de cette classe doit être la suivante (les constructeurs évidents et les *getters* pour chaque attribut, avec le même nom, sont omis pour éviter de surcharger la figure) :




La sortie du problème devra être donné sous la forme d'une instance de la classe `MarsRoverOutput` ayant la structure suivante :




Afin de vous faciliter la déclaration de ces classes, on vous conseille d'utiliser les classes *record* de Java. Les *records* sont des classes spéciales qui permettent de définir succinctement des classes qui agrègent des valeurs de manière immuable. Pour les définir la syntaxe est similaire à celle des classes normales en remplaçant `class` par `record` et en spécifiant les attributs de la classe entre parenthèses après le nom du *record*.

Par exemple, on peut définir un record `Rectangle` avec deux attributs `length` et `width` avec le code suivant :

```
public record Rectangle(double length, double width) { }
```

Un *record* possède les caractéristiques suivantes :

- chaque élément de sa description est un attribut `private` et `final` ;
- un *getter* public est défini pour chaque élément ;
- un constructeur public qui possède la même signature que celle de la description qui initialise chaque élément avec la valeur correspondante fournie en paramètre ;
- une redéfinition des méthodes `equals()` et `hashCode()` qui garantit que deux instances du record sont égales si elles sont du même type et qu'elles contiennent les mêmes attributs égaux.

Le record `Rectangle` est équivalent à la classe Java suivante :

```
public final class Rectangle {
    private final double length;
    private final double width;

    public Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }

    double length() { return this.length; }
    double width()  { return this.width; }

    // Implementation of equals() and hashCode(), which specify
    // that two record objects are equal if they
    // are of the same type and contain equal field values.
    public boolean equals...
    public int hashCode...

    // An implementation of toString() that returns a string
    // representation of all the record class's fields,
    // including their names.
    public String toString() {...}
}
```

Une fois que vous avez créé les classes, pour lire un fichier afin de créer un objet il faut :

1. Créer un `ObjectMapper` à partir d'une `YAMLFactory` ;
2. Ouvrir le fichier YAML à lire dans un `InputStream` ;
3. Lire le fichier afin d'obtenir l'objet correspondant à la racine du fichier *YAML* à l'aide de la méthode `readValue` d'`ObjectMapper` en donnant en argument l'`InputStream` et la classe de l'objet.

Cela donne par exemple le code suivant pour ouvrir un fichier `config.yml` situé dans le répertoire `ressources` du projet :

```
final ObjectMapper objectMapper = new ObjectMapper(new YAMLFactory());
try {
  final InputStream inputStream = App.class.getResourceAsStream("/config.yml");
  final GlobalConfiguration globalConfiguration =
    objectMapper.readValue(inputStream, GlobalConfiguration.class);
} catch (IOException e) {
  e.printStackTrace();
}
```

### 6.1.2 Écriture de fichier

Pour écrire un fichier *YAML*, il faut :

1. créer un `ObjectMapper` à partir d'une `YAMLFactory` (vous pouvez réutiliser celui utilisé pour la lecture) ;
2. ouvrir un nouveau fichier *YAML* à écrire dans un `OutputStream` ;
3. récupérer un `ObjectWriter` en appelant la méthode `writer` d'`ObjectMapper` ;
4. récupérer un `SequenceWriter` en appelant la méthode `writeValues` d'`ObjectWriter` en donnant l'`OutputStream` en argument ;
5. écrire le fichier en appelant la méthode `write`.

Cela donne par exemple le code suivant pour écrire un fichier `path/output.yml` avec le contenu d'un objet de type `GlobalOutput` :

```
ObjectMapper objectMapper = new ObjectMapper(new YAMLFactory());
try {
  GlobalOutput globalOutput = ...
  ObjectWriter writer = objectMapper.writer();
  ileOutputStream fos = new FileOutputStream("path/file.yml");
  SequenceWriter sw = writer.writeValues(fos);
  sw.write(globalOutput);
} catch (IOException e) {
  e.printStackTrace();
}
```

# 7 Exécution du code

Afin de pouvoir tester votre code manuellement, on vous conseille d'utiliser *gradle* pour exécuter le code avec la tâche *run* ou bien construire un *jar* avec la tâche *build*.

## 7.1 Tâche `run`

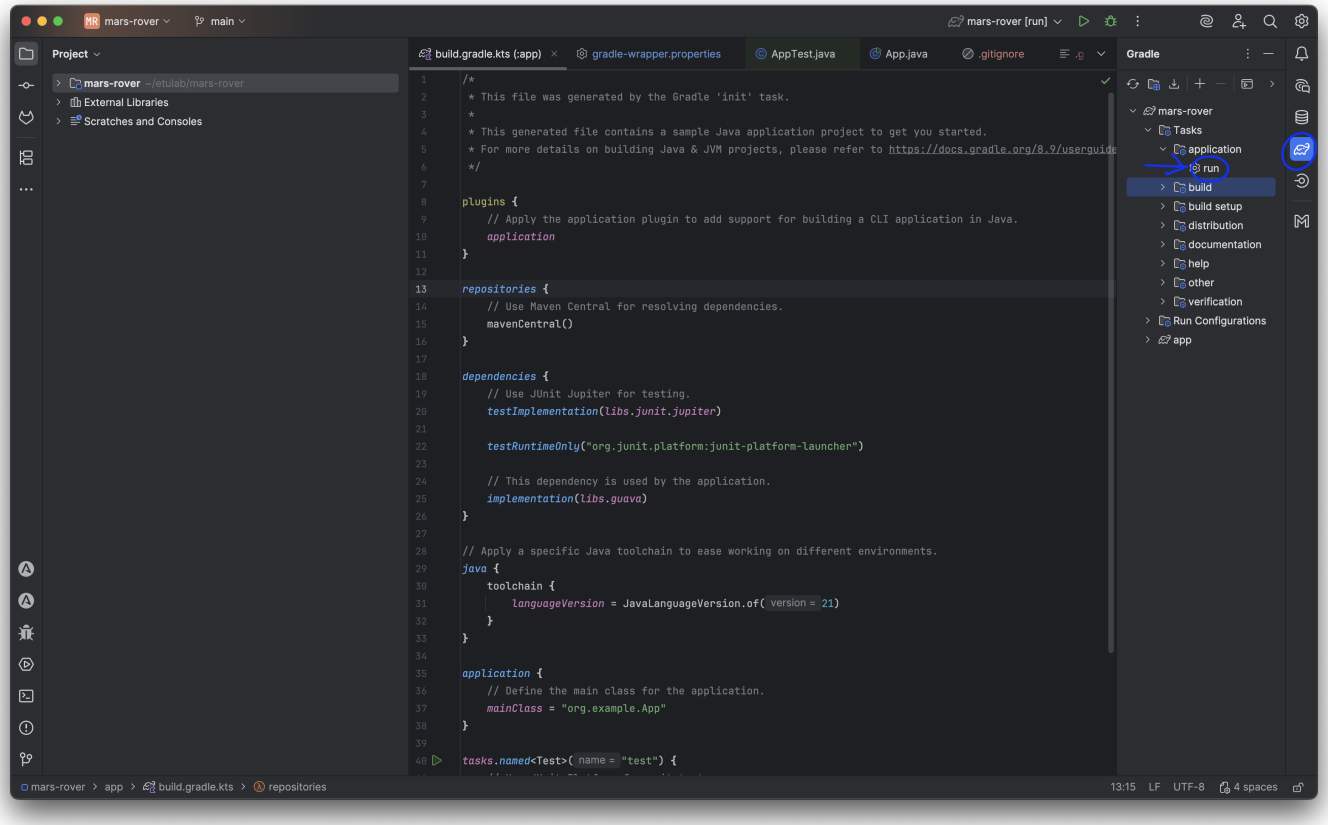
Pour configurer la tâche `run`, on va commencer par rajouter (si besoin) le plugin application dans le fichier `gradle.build.kts`. Pour cela, on rajoute la ligne suivante :

```
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"
}
```

Vous pouvez ouvrir le menu *gradle* en cliquant sur l'icône éléphant à droite. Vous avez éventuellement besoin de cliquer sur l'icône avec deux flèches en haut à gauche de l'onglet *gradle* pour faire mettre à jour la liste des tâches *gradle*. La tâche *run* se trouve dans le répertoire `application`.




## 7.2 Tâche `build`

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"
    }
}
```

La tâche *build* se trouve dans le répertoire `build`. Une fois le *jar* créé vous pouvez l'exécuter avec la commande suivante :

```
java -jar nom-du-jar.jar
```