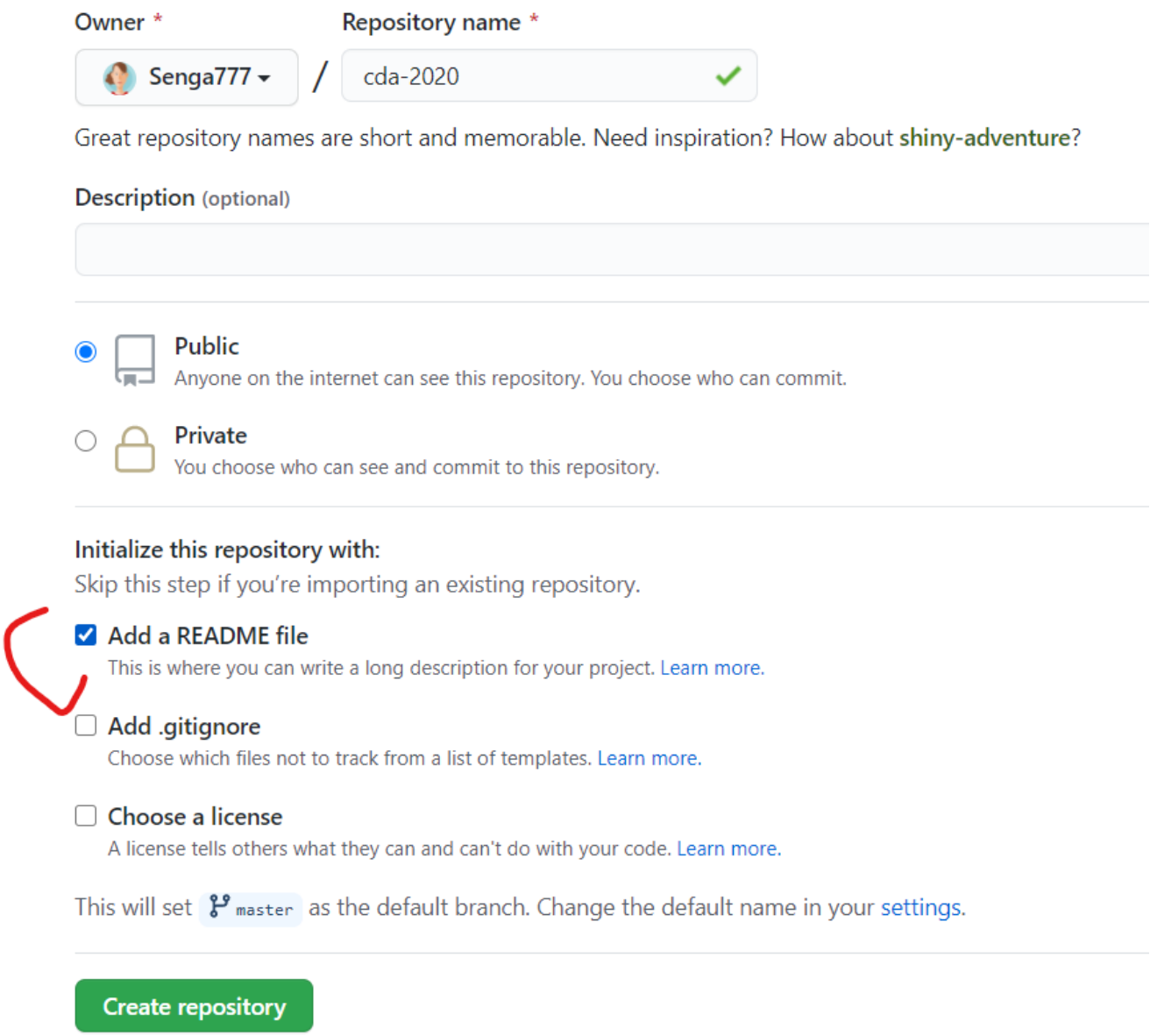
Git Manip

Dans un premier, vous allez travailler en utilisant git, de la façon, la plus simple qui soit, c'est-à-dire ... tout seul ! Vous allez travillez sur une seule branche, la branche (dite branche `master`). Cette première approche a pour but de vous familiariser. Plus tard, nous verrons comment utiliser le git en collaboration.

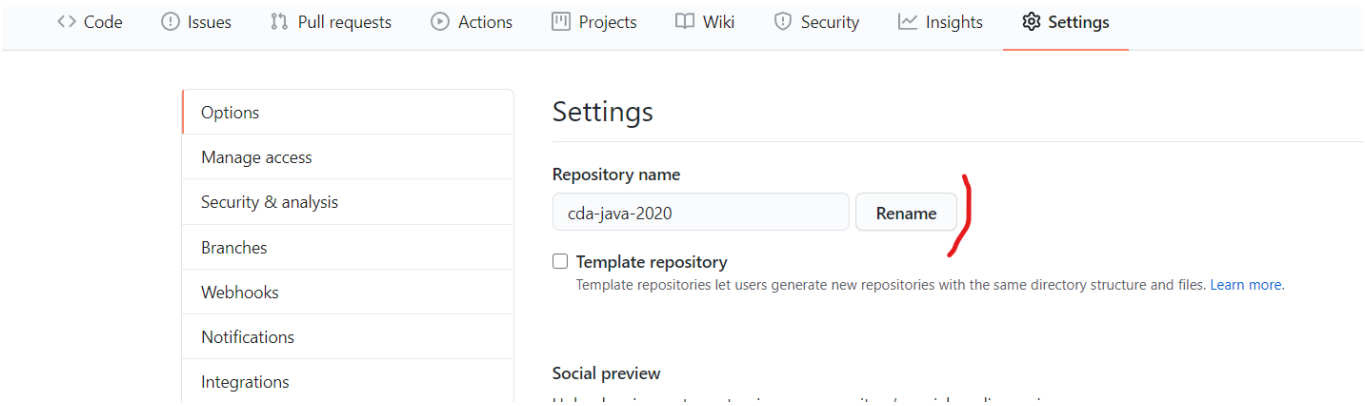
Etape 1

Créer un repository (abrégé en **repo**) sur GitHub, avec un ReadMe. Appelons par exemple : `cda-2020`



le repo `cda-2020` est distant, il est en ligne sur github.

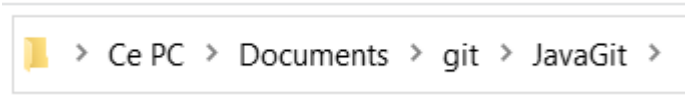
Un repository se comporte comme un dossier. On peut renommer un repository :



Renommer le pour `cda-java-2020`

Etape 2

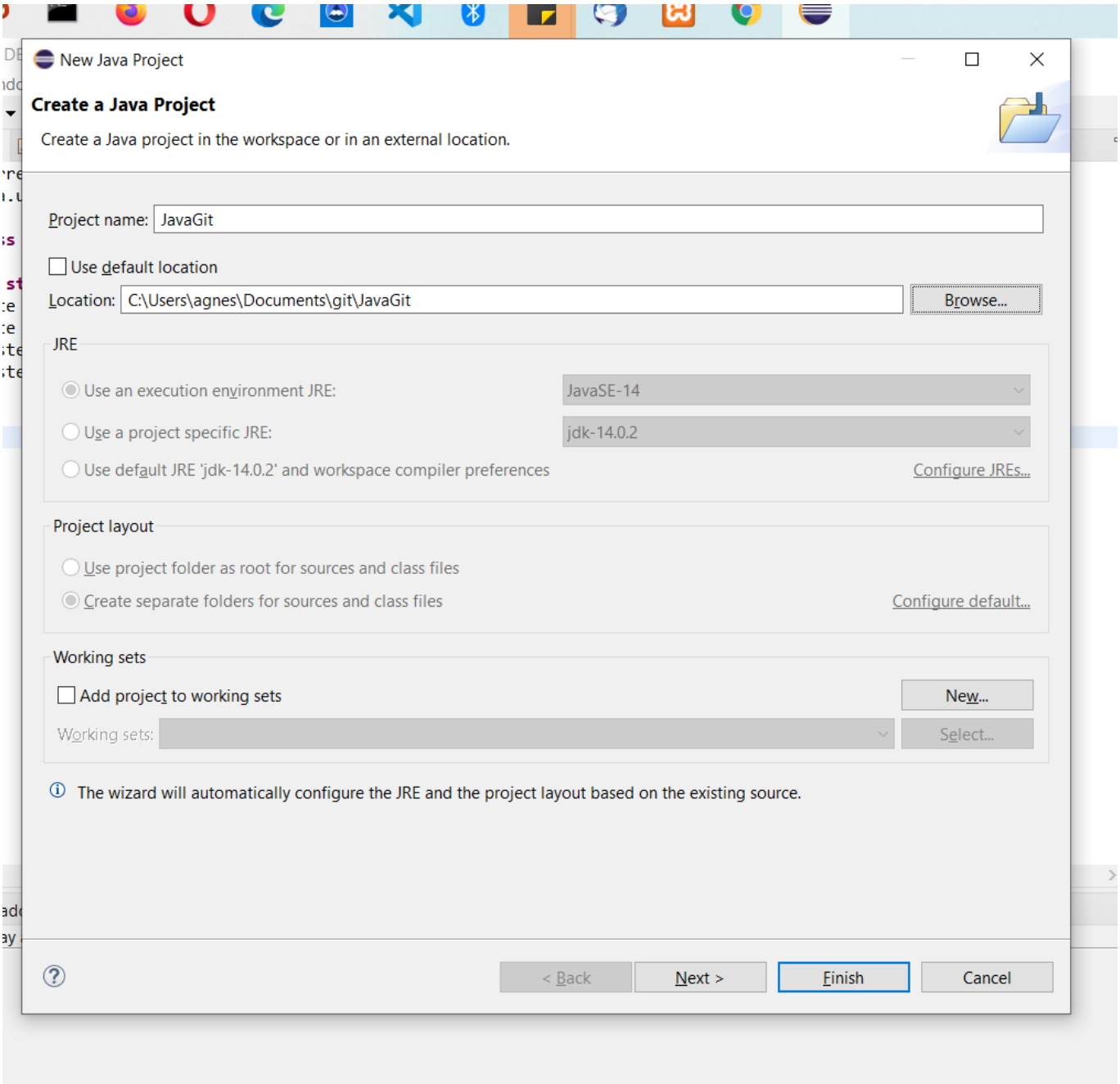
Creer un dossier `git` dans documents, qui contiendra tous les "repo" git. Dans ce dossier, créer un dossier `javaGit` pour notre "repo" qu'on vient de créer, cda-java-2020.



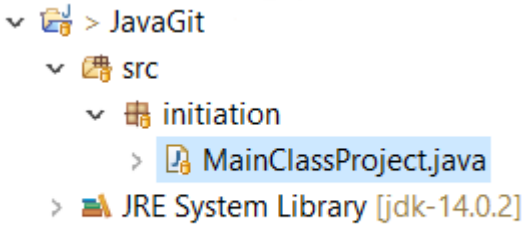
Le dossier `javaGit` est en local sur vos machines.

Etape 3

Dans Eclipse, créer un nouveau projet Java. Par exemple, je l'appelle `javaGit`. C'est le nom du projet que je veux partager.



Ceci permet d'avoir la structure d'un projet Java. On va créer une classe pour vérifier que cela fonctionne.



Par exemple, vous pouvez afficher un "hello world".

Donc on a :

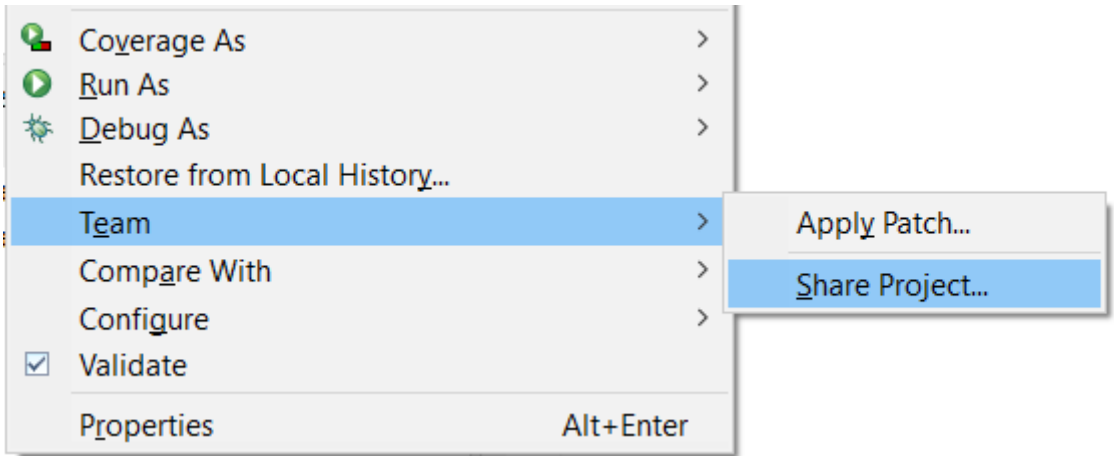
- un dossier en local, simple, qui contient un projet Java qui fonctionne.
- un dossier distant, cda-java-2020, qui contient rien pour le moment, mis à par un fichier `readme`

Etape 4

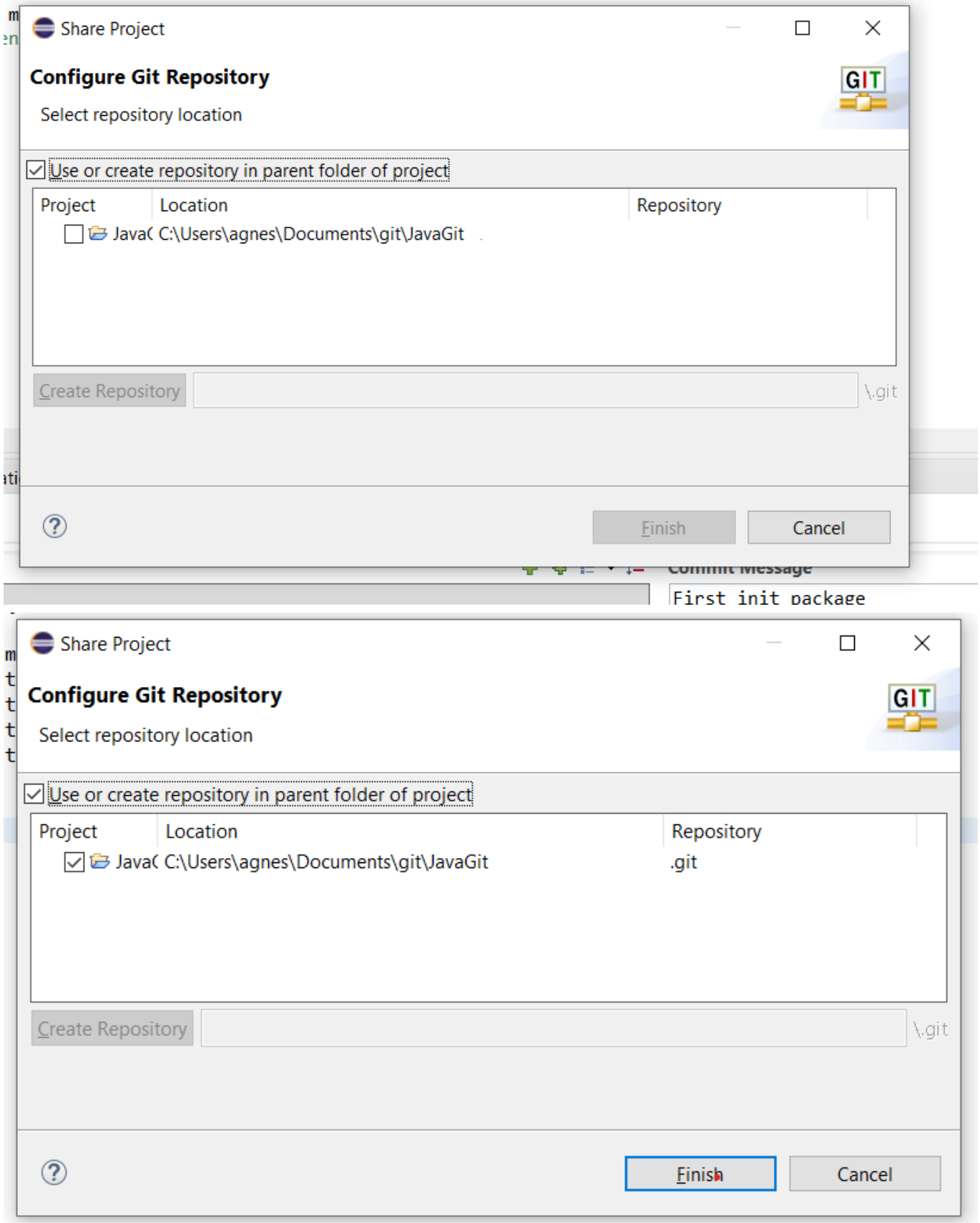
Maintenant que tout est pret, il faut initialiser `git` dans ce projet.

Pour cela :

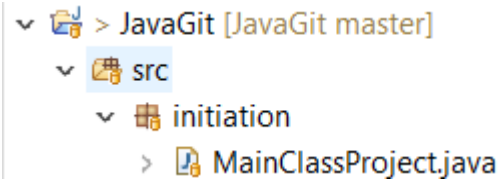
- Click droit que le nom du projet,
- choisir `team`
- puis `share project`



Il faut cocher la clause : `Use or create repository in parent folder of project`

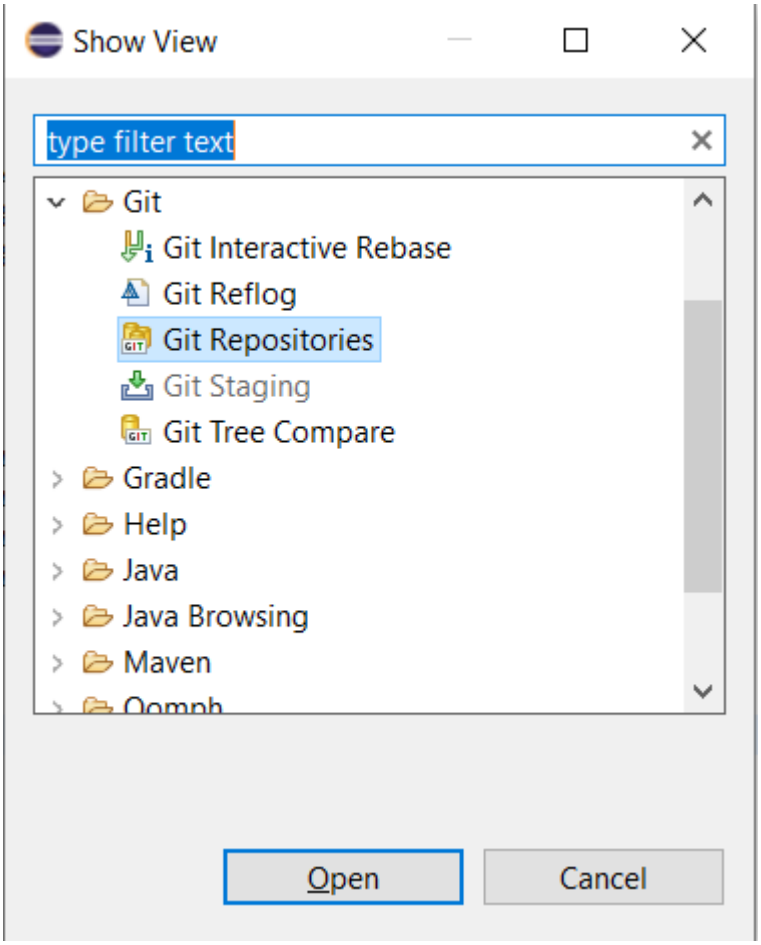


Lorsque vous voyez ce `.git`, cela signifie que le git est pret pour ce projet. Vous devez avoir à côté du projet : `[JavaGit Master]`



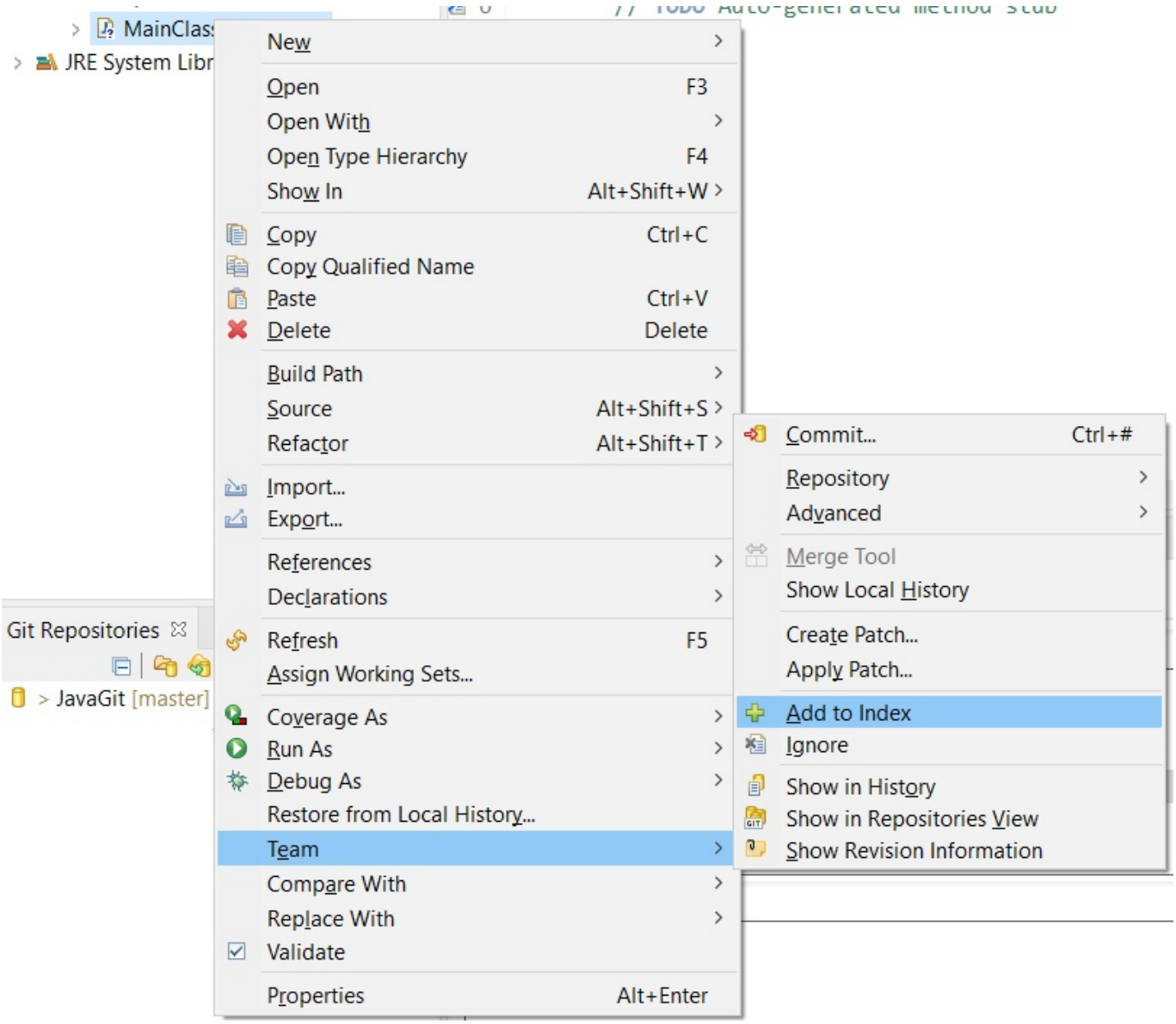
Etape 5

Ajouter la vue pour git : `window/Show View` puis choisir :



Etape 6

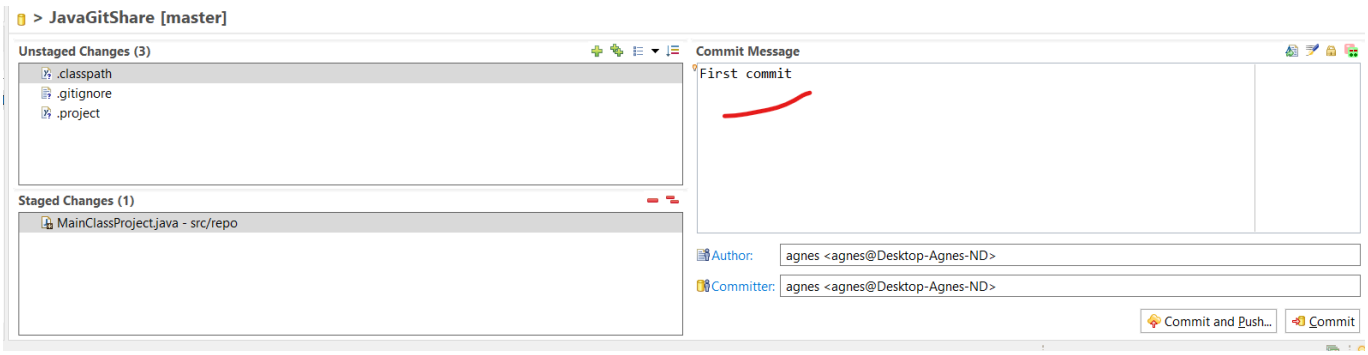
Nous allons indiquer les changements à mettre à jour. Pour cela on ajoute un index sur la classe `MainClassProject.java`



Etape 7

Une fois qu'on a indiqué les éléments à surveiller pour la modification (c'est-à-dire qu'on ajouter notre index).

On doit soumettre nos changements : le mot clé pour cette action est `commit`



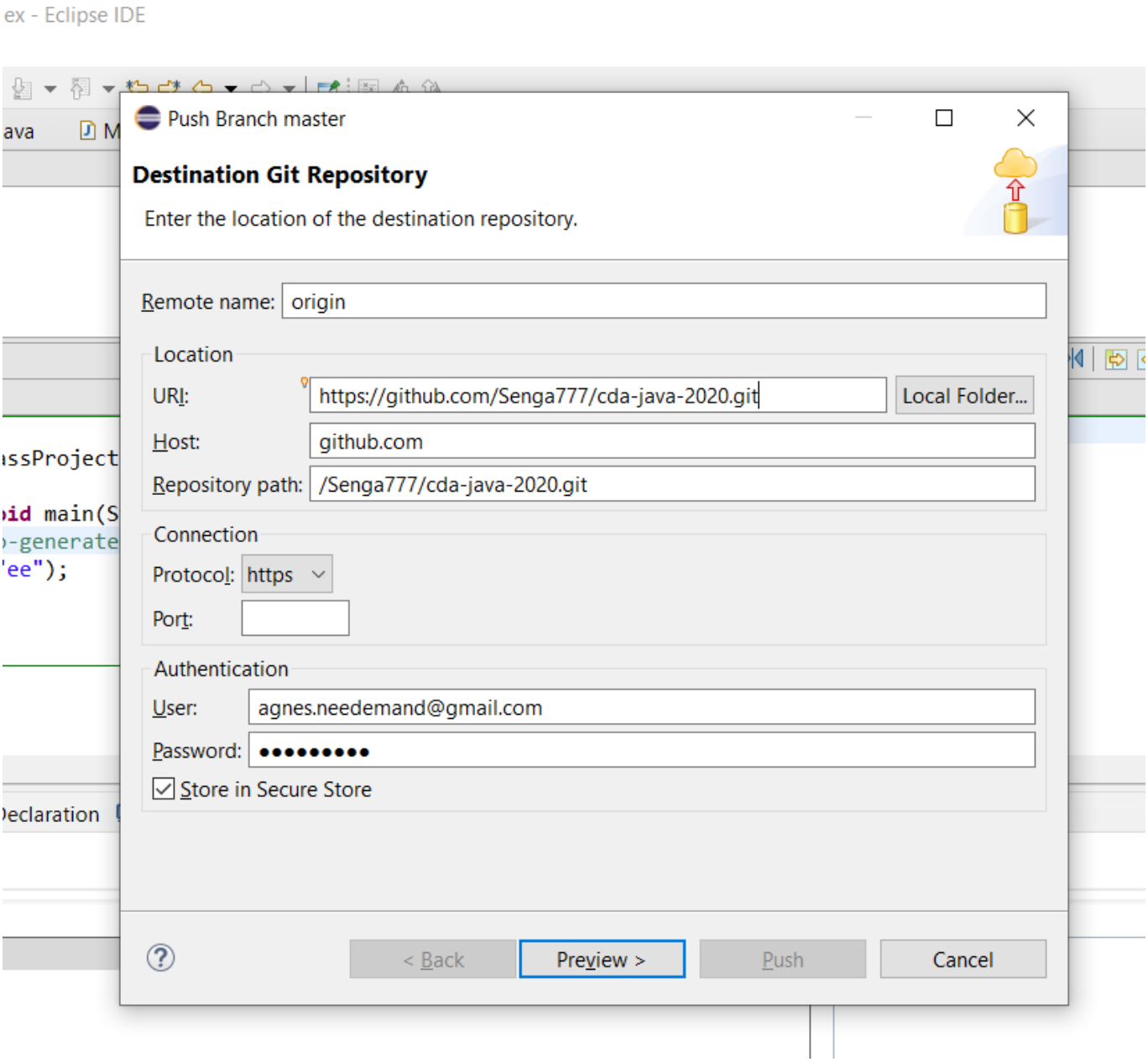
Lorsqu'on effectue un `commit` il faut impérativement ajouter un commentaire.

Ensuite vous devez choisir `commit and push`

`Push` : Signifie que l'on va faire correspondre le local avec le distant.

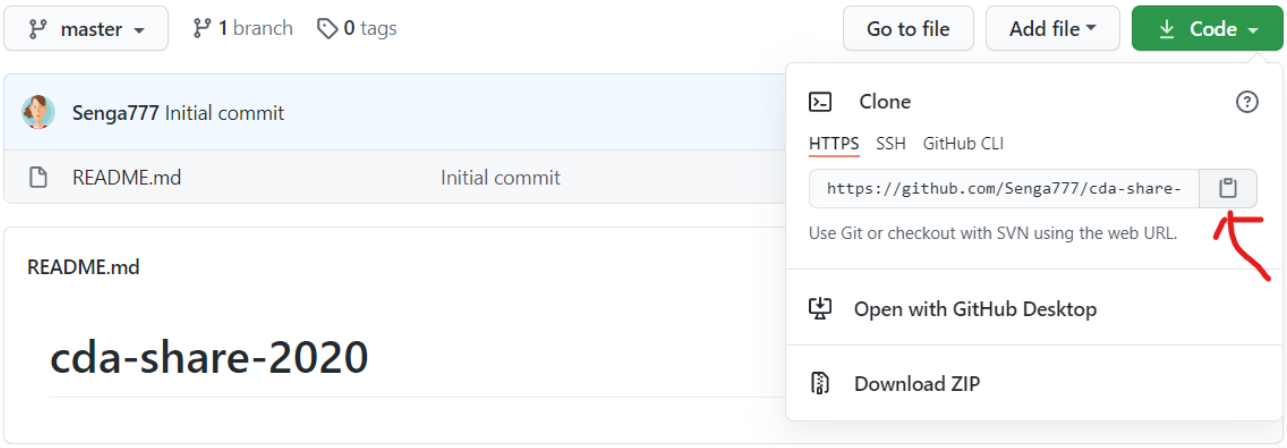
La première fois, que l'on fait un `push`, il faut lui dire ou il doit "pousser" le code.

Préciser le repo de destination.



L'adresse URI vous la récuperer sur le site GitHub, ici :

Copier l'adresse du repo



Vous "copiez-coller" cette adresse dans le champs URI. Ensuite, on vous demandera votre login et votre mot de passe.

Etape 8

Normalement, vous devez avoir une **erreur** du type `no-fast-forward`. Cette erreur signifie qu'il y a une différence entre le repot distant et le repo local.

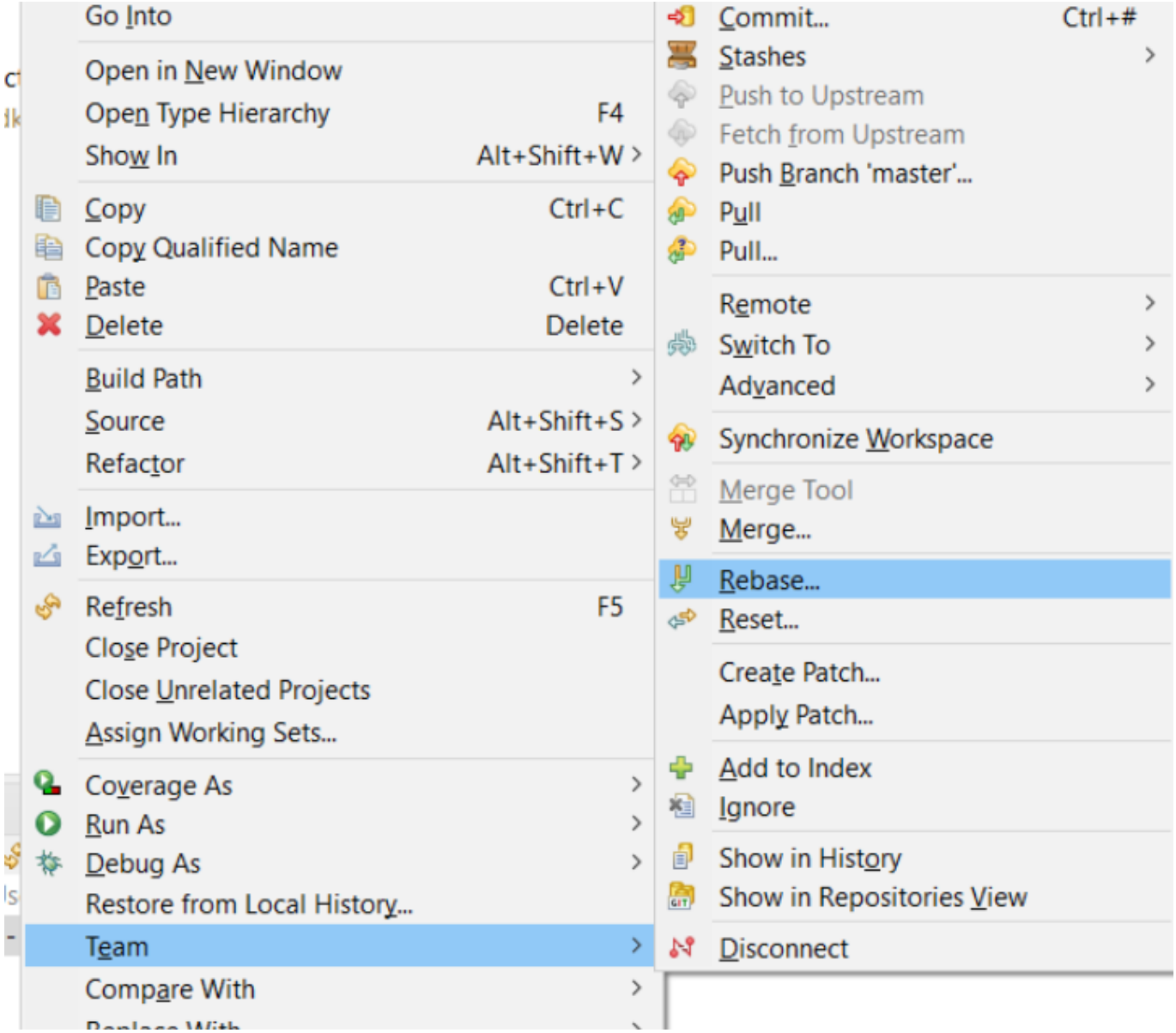
En effet, sur le distant, il ya un fichier `ReadMe`. Et en local, il y a notre classe `MainClassProject`.

Dans ce cas, git, ne peut pas gérer les changements.

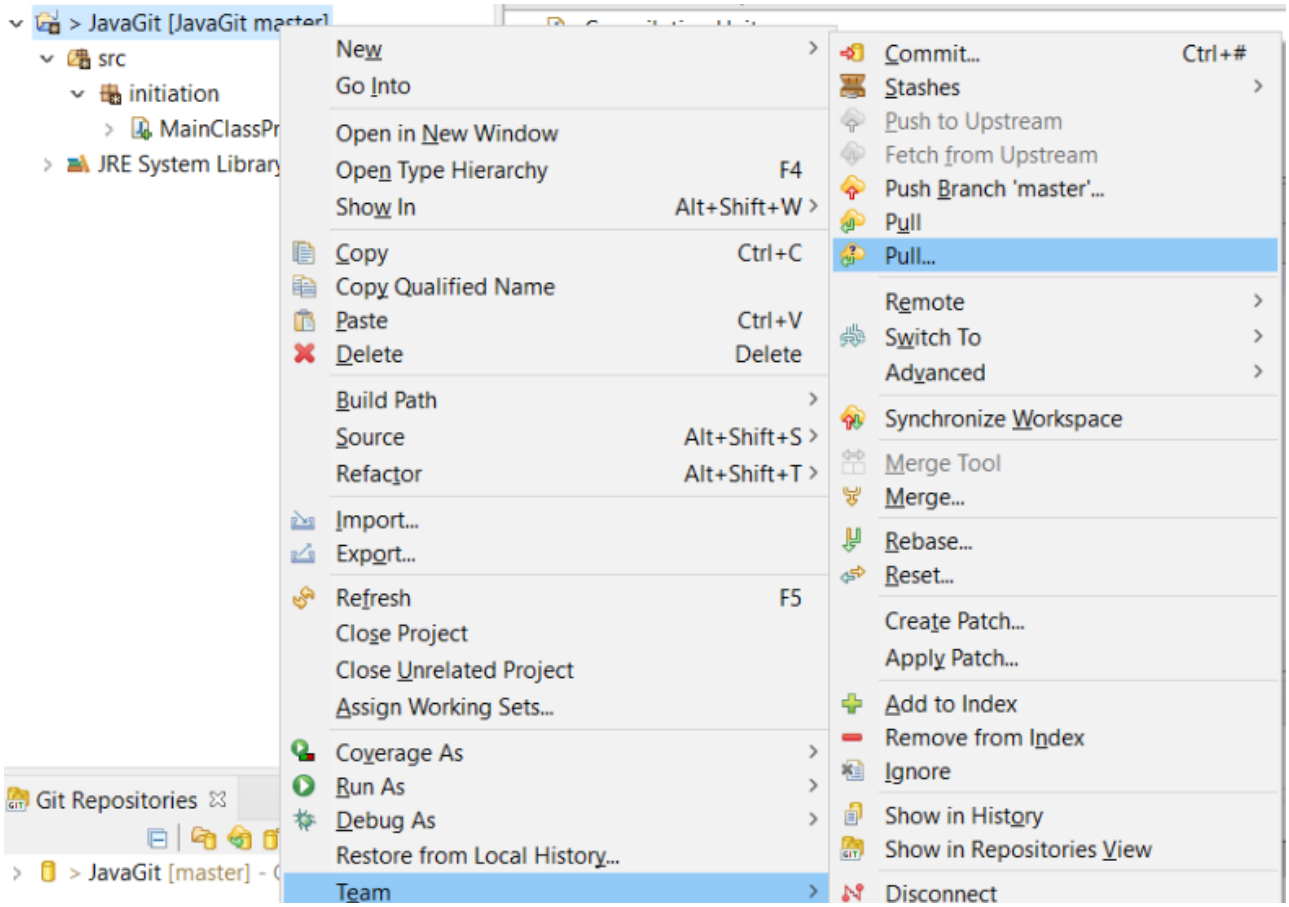
Etape 9

Pour corriger cette erreur, nous allons :

- `rebase` : On repart sur des bonnes bases.


Go Into
Open in New Window
Open Type Hierarchy F4
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
Refresh F5
Close Project
Close Unrelated Projects
Assign Working Sets...
Coverage As
Run As
Debug As
Restore from Local History...
Team
Compare With
Commit... Ctrl+#
Stashes
Push to Upstream
Fetch from Upstream
Push Branch 'master'...
Pull
Pull...
Remote
Switch To
Advanced
Synchronize Workspace
Merge Tool
Merge...
Rebase...
Reset...
Create Patch...
Apply Patch...
Add to Index
Ignore
Show in History
Show in Repositories View
Disconnect

- `pull` : On récupère les élements distants, pour les mettre en local.



  Normalement, après cette opération, le fichier `readme` present en local.

Remarque : Les commandes se trouvent toujours dans `Team`

> !important : Il faudra veiller constament que le projet local est à jour. Donc quand vous codez, vous devrez toujours `pull` le code distant.

Là vous êtes à jour, en local.

Etape 10

Enfin, il faut `push`. C'est à dire, que l'on envoie le code local sur le distant. (`push` = pousser) ... Toujours dans Team.

Là vous êtes à jour, en local, et à distance.

Conclusion

Bonnes pratiques

- `pull` = tirer. Vous "tirez" le code du serveur distant. Comme ça vous êtes à jour.
- Ensuite vous produisez du code ... normal.
- Puis, quand le code fonctionne, vous soummettez votre code. avec `add index` et `commit`, en ajoutant un commentaire à chaque fois.
- Quand vous voulez mettre à jour votre code distant, vous poussez le code, grace à la commande `push`.

En général, c'est en fin de sujet, ou en fin de journée. On ne `push` pas, ensemble, plusieurs travaux effectués qui ne sont pas en relation.

Par exemple, si vous devez faire un algorithme, qui permet de convertir un nombre en chiffre romain. Quand l'exercice est fini, que vous allez travailler sur autre chose. Vous effectuez un `push`, avec le commentaire adéquat.

En cas d'erreur

Eventuellement, si vous avez de nouveau un problème de type : `no-fast-forward` vous pouvez éventuellement, refaire un `rebase`. Pour les autres erreurs, il faudra voir au cas par cas.