GIT

PLAN DU COURS

- Outils de Control de Version (CVS, SVN, **GIT**)
- Repositories Git (Local, Distant)
- Commandes GIT avec Git Bash
- Commandes Git avec une IDE
- Projet : Travail en Equipe (Ajouter / Récupérer un Projet de GitHub, Branches, Conflit, …)

Outils de Control de Version (CVS, SVN, GIT)

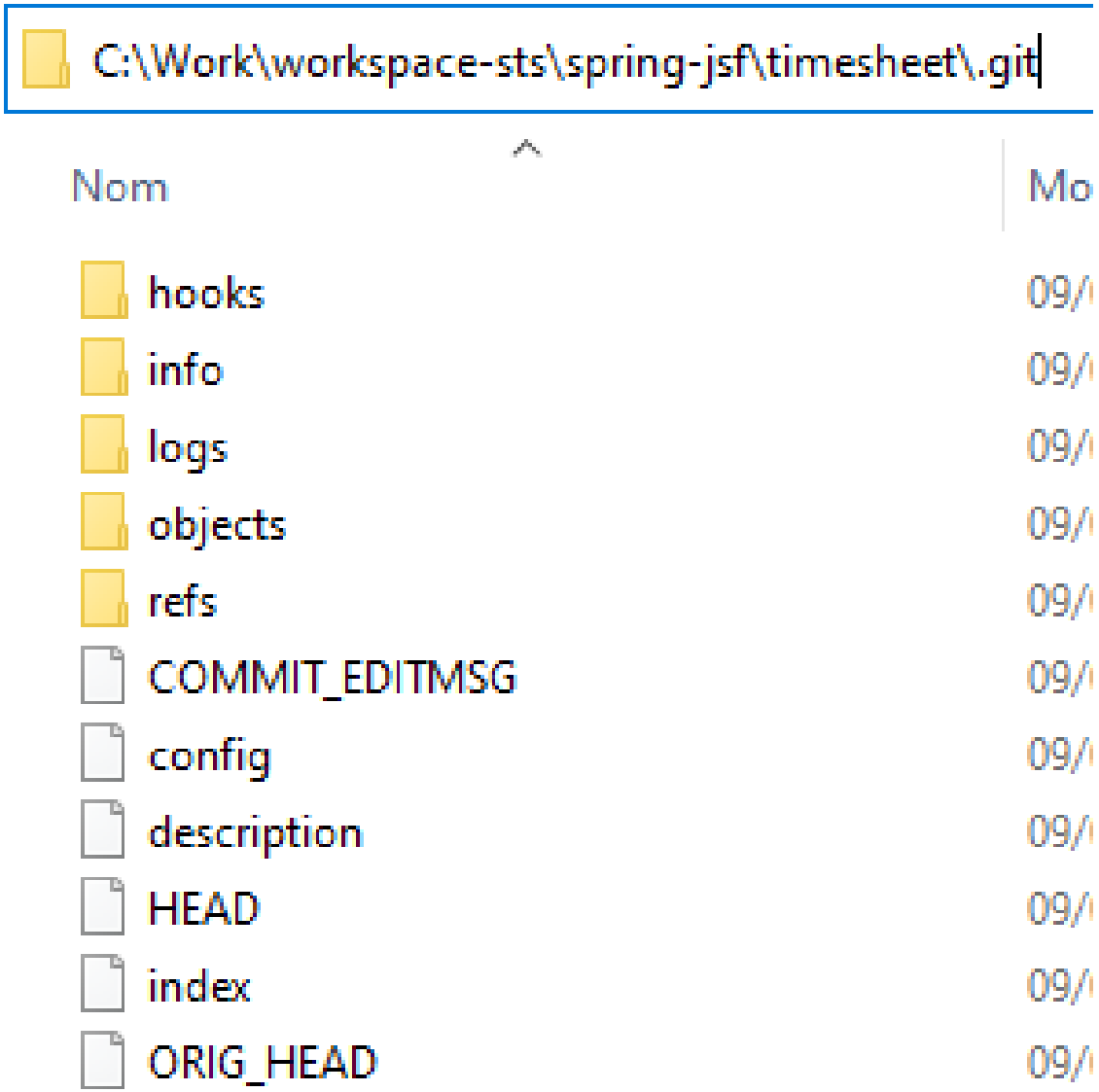
- Outils de control de version : versionner, historiser, faire travail l'équipe ensemble, traçabilité, ….
- Exemples : CVS, SVN, GIT
- Git présente un avantage majeur qui a fait que c'est l'outil le plus utilisé maintenant :

Git

- **Git** est un système de contrôle de version **distribué** gratuit et open source conçu pour tout gérer (historique, traçabilité des modifications, versionning, travail en équipe, …), des petits projets aux très grands projets avec rapidité et efficacité.

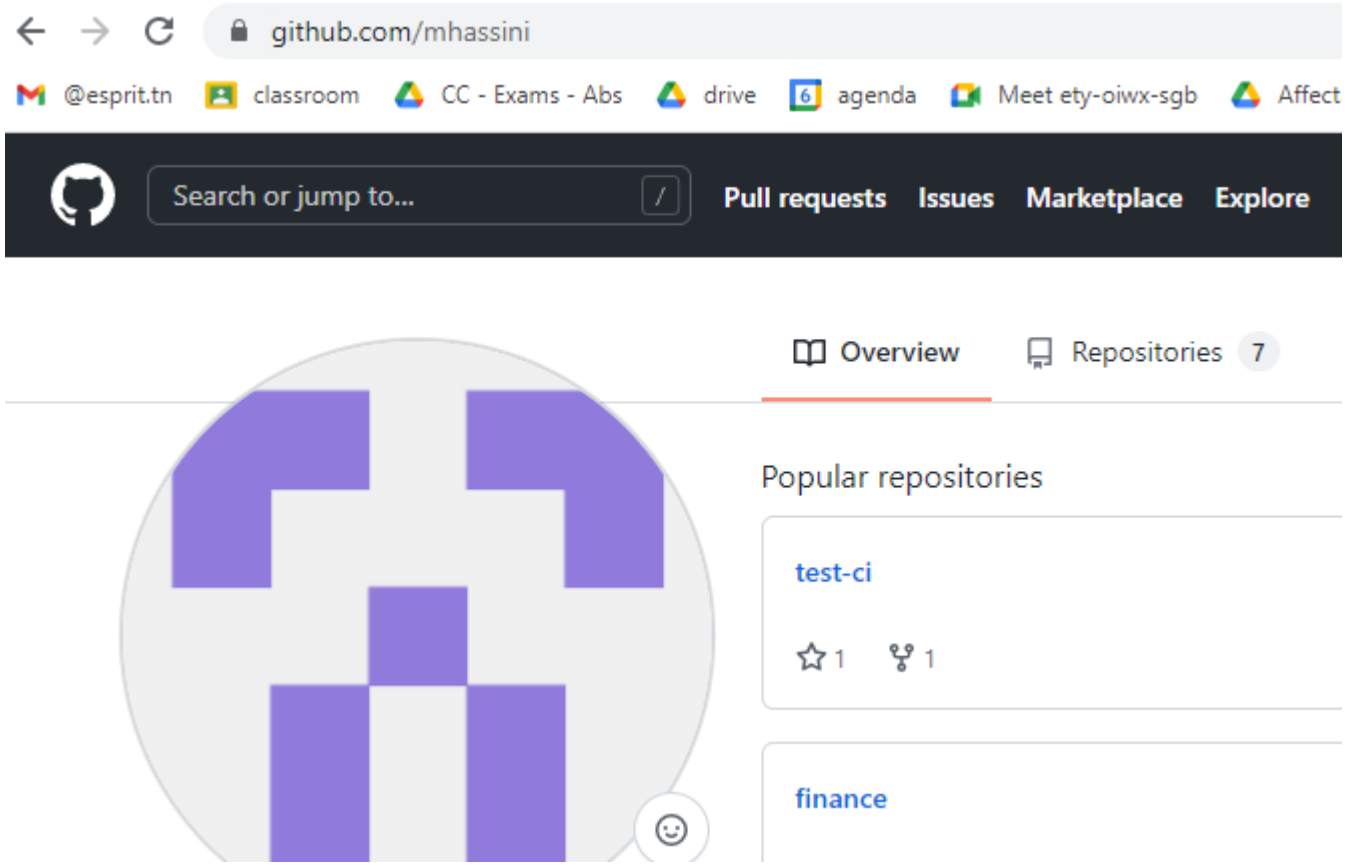
Repositories Git

- Local : .git



Repositories Git

- Distant : Github, Gitlab, Bitbucket, …
- Créer un compte sur Github : github.com



Initialisation de GIT

- Installer Git à partir du site officiel **https://git-scm.com**
- SCM : Source Control Management
- Ouvrir **Git Bash** et lancer **git --version** pour vérifier que Git est bien installé :

```
Mourad HASSINI@M104 MINGW64 ~
$ git --version
git version 2.16.2.windows.1
```

Initialisation de GIT

- Toujours sur Git Bash, définir ton identité, avec **git config**, pour éviter que Git vous demande cela à chaque action

```
$ git config --list
```

```
$ git config --global user.name "Mourad HASSINI"
```

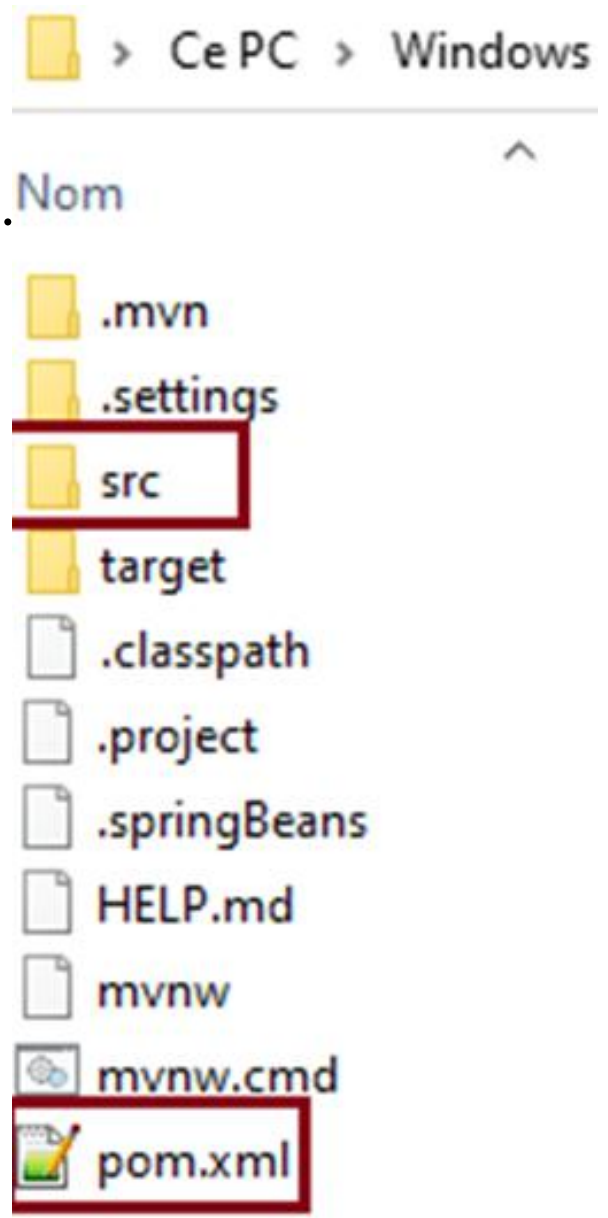
```
$ git config --global user.email "mourad.hassini@esprit.tn"
```

Initialisation de GIT

Chacun fera les manipulations sur un exemple de projet simple se trouvant dans votre workspace IntelliJ.

Vous pouvez sinon utiliser le projet simple timesheet-devops se trouvant dans le Drive (téléchargez-le et dézippez le dans votre workspace IntelliJ).

Seuls le dossier **src** et le fichier **pom.xml** seront gérés par Git :



Initialisation de GIT

- Aller dans le projet timesheet-devops déjà existant sur votre workspace, et initialiser Git pour pouvoir l'utiliser sur ce projet (**git init** puis **git status**) :

```
$ cd C:/Work/workspace-intellij/timesheet-devops

$ git init
Initialized empty Git repository in C:/Work/workspace-
intellij/timesheet-devops/.git/

$ git status
nothing added to commit but untracked files present (use "git add"
to track)
```

Le Commit

- Sélectionner les fichiers à commiter (dossier src et fichier pom.xml uniquement) avec la commande **git add**
- Puis un **git status**
- Puis faire un **git commit** . Cela revient à dire à Git de prendre une photo de ton projet.

```
$ git add pom.xml src
$ git status
```

(ne pas inclure les fichiers : .classpath .mvn/ .project .settings/ .springBeans HELP.md mvnw mvnw.cmd : ces fichiers sont locaux et créés automatiquement, le développer ne le changera jamais).

```
$ git commit -m "initialisation du projet timesheet-devops"
 [master (root-commit) 6257aab] initialisation du projet timesheet-devops
 82 files changed, 2923 insertions(+)
 …
```

Le Commit

- Après chaque modification d'un ou de plusieurs fichiers, vous pouvez refaire la même action pour commiter vos modifications : git status puis git add puis git commit (avec un autre commentaire bien sûr).
- Modifier le contenu d'un fichier (User.java par exemple) et refaites les actions ci-dessous :

```
$ git status
$ git add src
$ git commit -m "ajout commentaire"
[master 5386a19] ajout commentaire
 1 file changed, 1 insertion(+)
```

L'historique

- Lancer la commande **git log** pour voir les deux commit (les deux photos de ton projet) :

```
$ git log
commit 5386a198346eb13131c60ea9be651a3da667c4c2 (HEAD -> master)
Author: Mourad HASSINI <mourad.hassini@esprit.tn>
Date:   Fri Apr 9 23:18:29 2021 +0100
    ajout commentaire

commit 6257aabac843778b7e7599f86b72f7b3bddf4c1a
Author: Mourad HASSINI <mourad.hassini@esprit.tn>
Date:   Fri Apr 9 23:10:06 2021 +0100
    initialisation du projet timesheet-devops
```

Les branches

- Créer une branche (**git branch**) et lister les branches (le * indique sur quelle branche nous sommes actuellement) :

```
$ git branch user-management

$ git branch
* master
  user-management
```

Les branches

- Basculer sur la branche user-management :

```
$ git checkout user-management
Switched to branch 'user-management'

$ git branch
  master
* user-management
```

Les branches

- Faite une modification sur votre fichier (User.java par exemple) et commiter la modification sur cette branche :

```
$ git add src

$ git commit -m "modification 2"
[user-management 042579c] 2ème ajout commentaire
 1 file changed, 2 insertions(+), 1 deletion(-)
```

- Quelle commande pour voir l'historique :

Les branches

```
$ git log

commit 690f70f416800f25b66554c641480e344230f8f9 (HEAD -> user-management)
Author: Mourad HASSINI <mourad.hassini@esprit.tn>
Date:   Fri Apr 9 23:25:29 2021 +0100

    2ème ajout commentaire

commit 5386a198346eb13131c60ea9be651a3da667c4c2 (master)
Author: Mourad HASSINI <mourad.hassini@esprit.tn>
Date:   Fri Apr 9 23:18:29 2021 +0100

    ajout commentaire

commit 6257aabac843778b7e7599f86b72f7b3bddf4c1a
Author: Mourad HASSINI <mourad.hassini@esprit.tn>
Date:   Fri Apr 9 23:10:06 2021 +0100

    initialisation du projet timesheet-devops
```

Les branches

- Récupérer ce travail sur la branche principale : basculer sur la branche qui va recevoir (master dans notre cas), faire un merge et supprimer la branche qui vient d'être mergée :

Les branches

```
$ git checkout master
Switched to branch 'master'
```

```
$ git branch
* master
  user-management
```

```
$ git merge user-management
Updating 5386a19..042579c
Fast-forward
 src/main/java/tn/esprit/spring/entities/User.java | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)
```

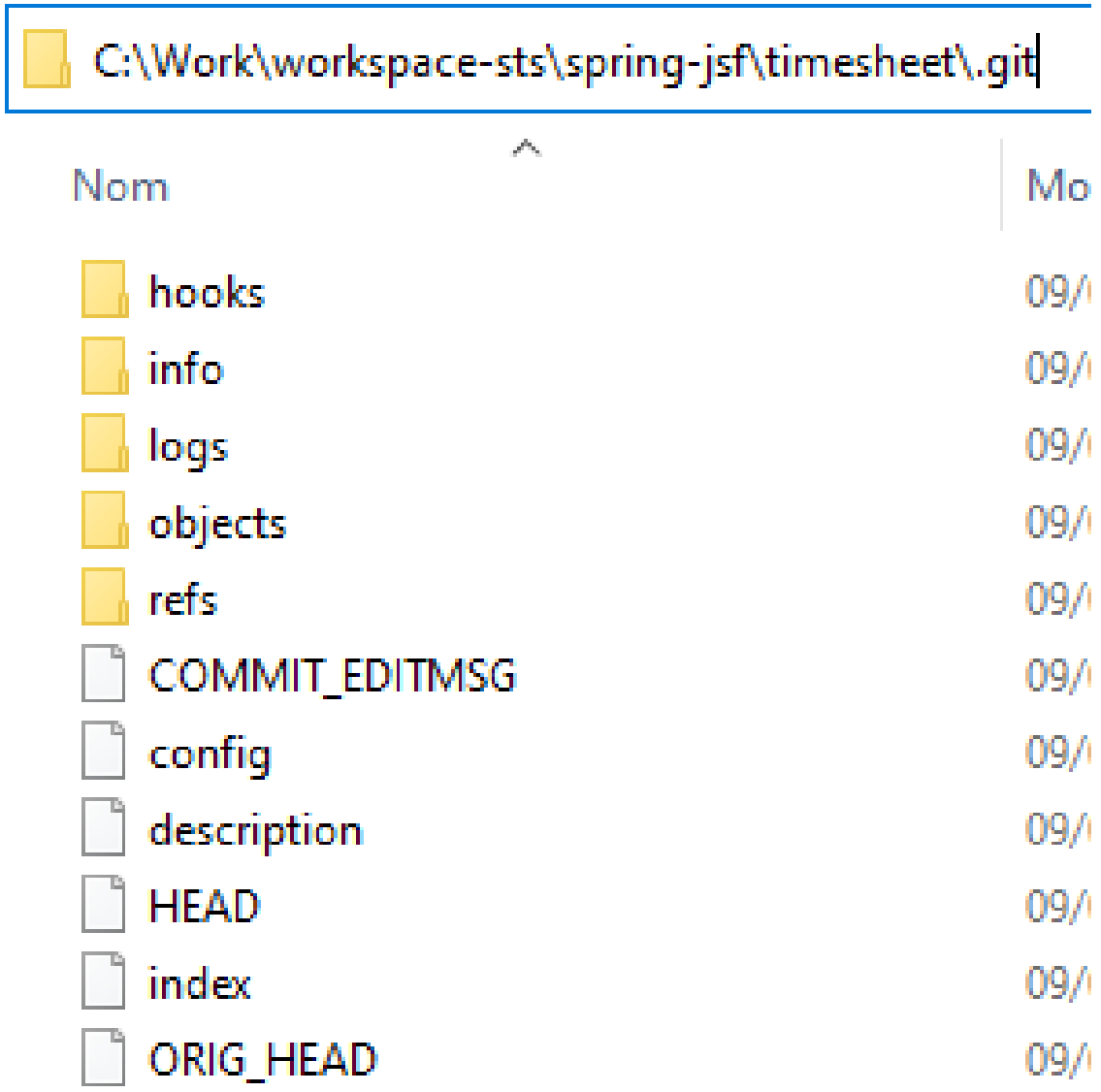
Les branches

```
$ git log
commit 042579c2d194e357f6670eff05f214f8f7325136 (HEAD -> master, user-management)
Author: Mourad HASSINI <mourad.hassini@esprit.tn>
Date:   Fri Apr 9 23:25:29 2021 +0100
    2ème ajout commentaire
commit 5386a198346eb13131c60ea9be651a3da667c4c2
Author: Mourad HASSINI <mourad.hassini@esprit.tn>
Date:   Fri Apr 9 23:18:29 2021 +0100
    ajout commentaire
commit 6257aabac843778b7e7599f86b72f7b3bddf4c1a
Author: Mourad HASSINI <mourad.hassini@esprit.tn>
Date:   Fri Apr 9 23:10:06 2021 +0100
    initialisation du projet timesheet-devops

$ git branch -d user-management
Deleted branch user-management (was 0b4990e).

$ git branch
* master
```

Dépôt distant GitHub

Jusque là, Git nous a permis de travailler seul sur notre projet en local (versionning, historisation, …).

Nous avons utiliser le **Dépôt Git local** : dossier (**.git**) :

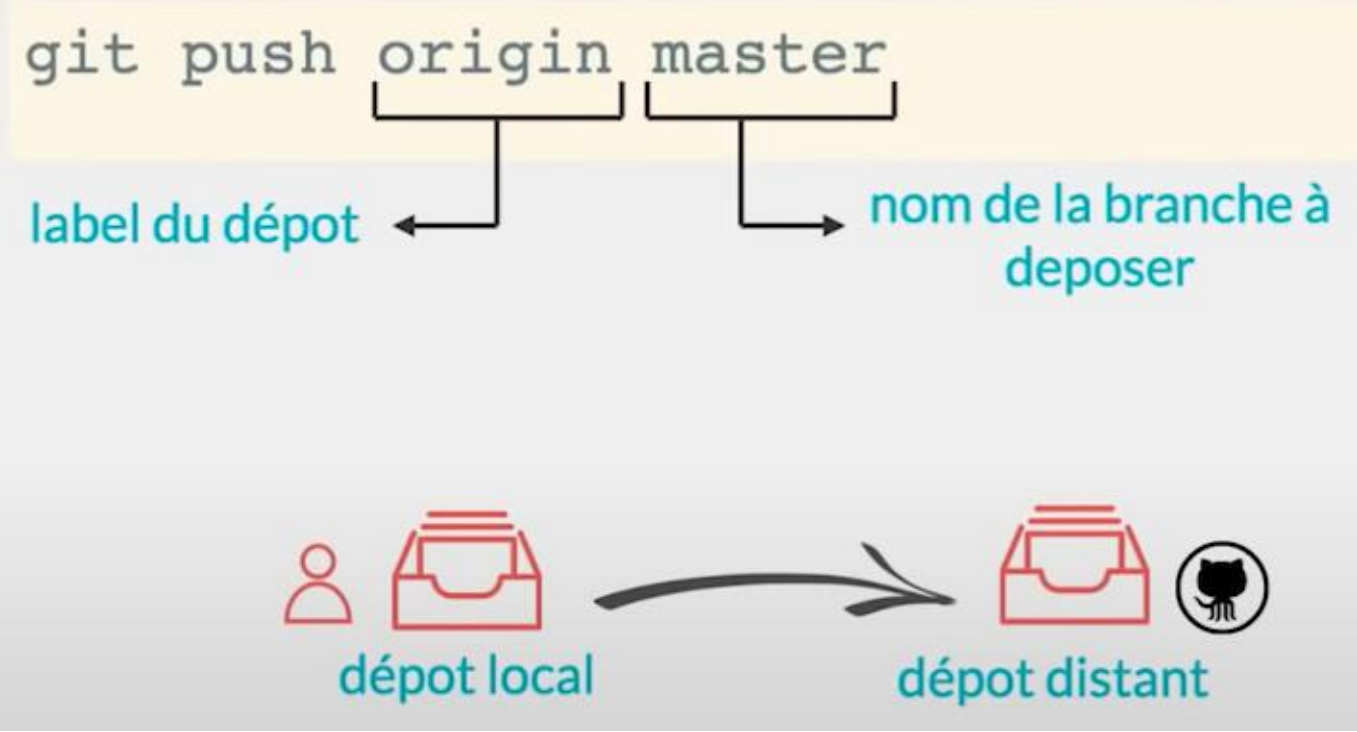


Dépôt distant GitHub

- Si on souhaite travailler en équipe :

Dépôt distant GitHub

- Créer un compte sur GitHub github.com
- Créer un Repository
- Associer le dépôt distant à notre dépôt local (nom du dépôt distant origin)
- Enfin déposer votre projet sur le dépôt distant :



Dépôt distant GitHub

```
Mourad HASSINI@M104 MINGW64 /c/Work/workspace-intellij/timesheet-devops (master)
$ git remote add origin https://github.com/mhassini/timesheet-devops.git


$ git remote
origin


$ git push origin master
Counting objects: 137, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (116/116), done.
Writing objects: 100% (137/137), 58.56 KiB | 768.00 KiB/s, done.
Total 137 (delta 24), reused 0 (delta 0)
remote: Resolving deltas: 100% (24/24), done.
To https://github.com/mhassini/timesheet-devops.git
 * [new branch]      master -> master
```

Dépôt distant GitHub

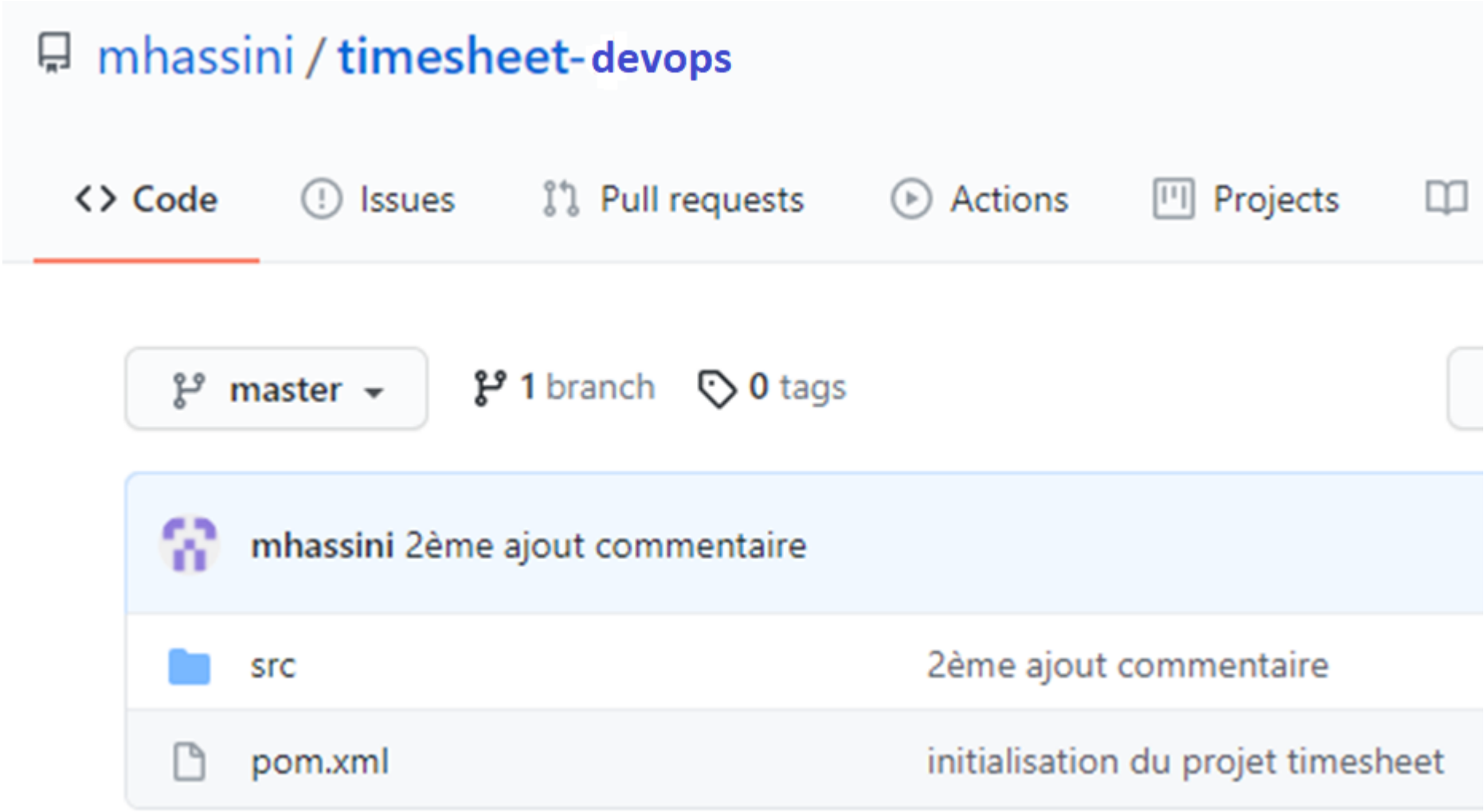
Attention au Password (si erreur) :

- Mourad HASSINI@M104 MINGW64 /c/Work/workspace-eclipse/maven/avec-maven (master)
- $ git push origin master
- Logon failed, use ctrl+c to cancel basic credential prompt.
- Username for 'https://github.com': mourad.hassini@esprit.tn
- remote: Support for password authentication was removed on August 13, 2021. Please use a personal access token instead.
- remote: Please see https://github.blog/2020-12-15-token-authentication-requirements-for-git-operations/ for more information.
- fatal: Authentication failed for 'https://github.com/mhassini/avec-maven.git/'

Solution : Créaton de token :

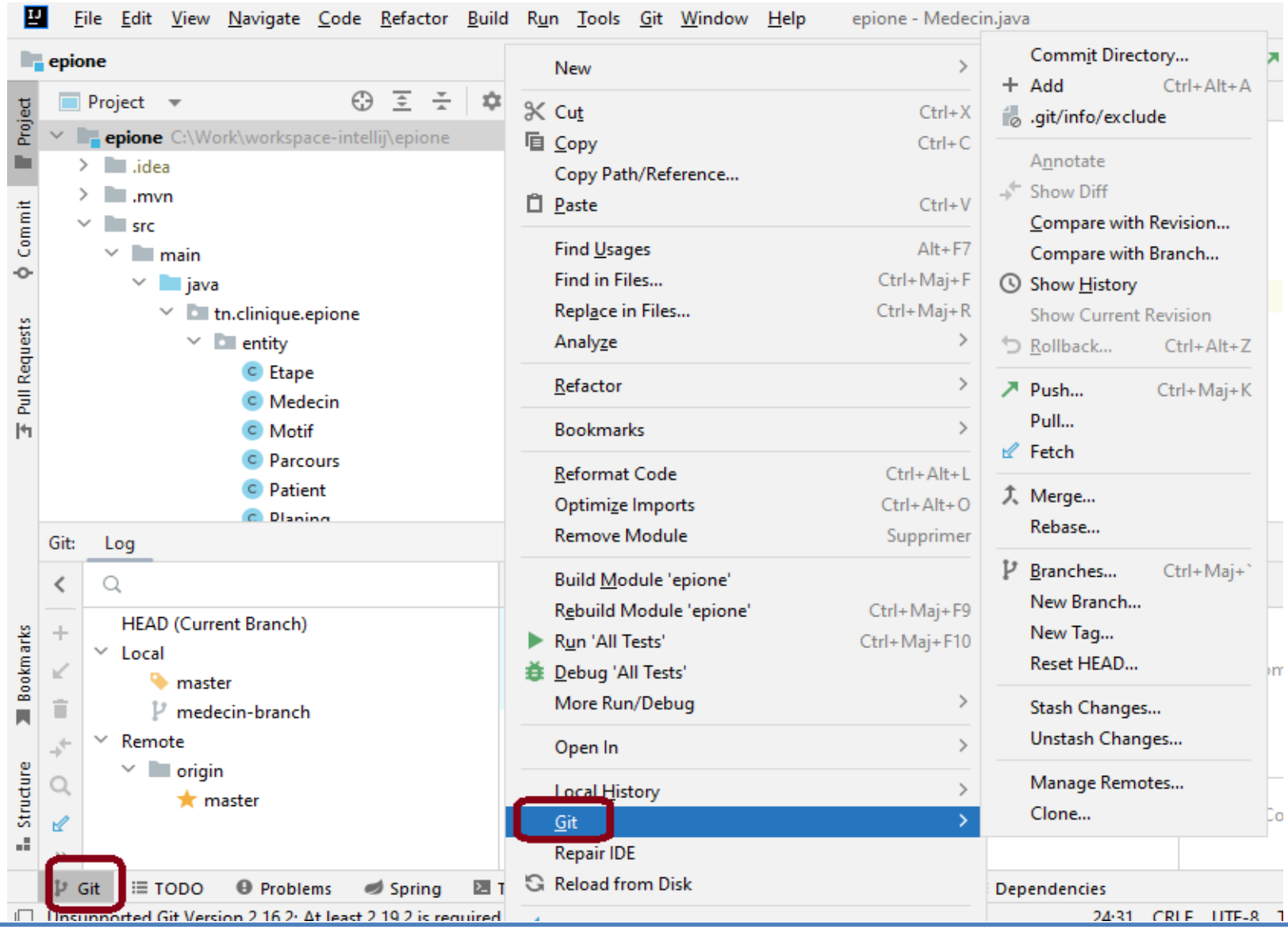
https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token

Dépôt distant GitHub



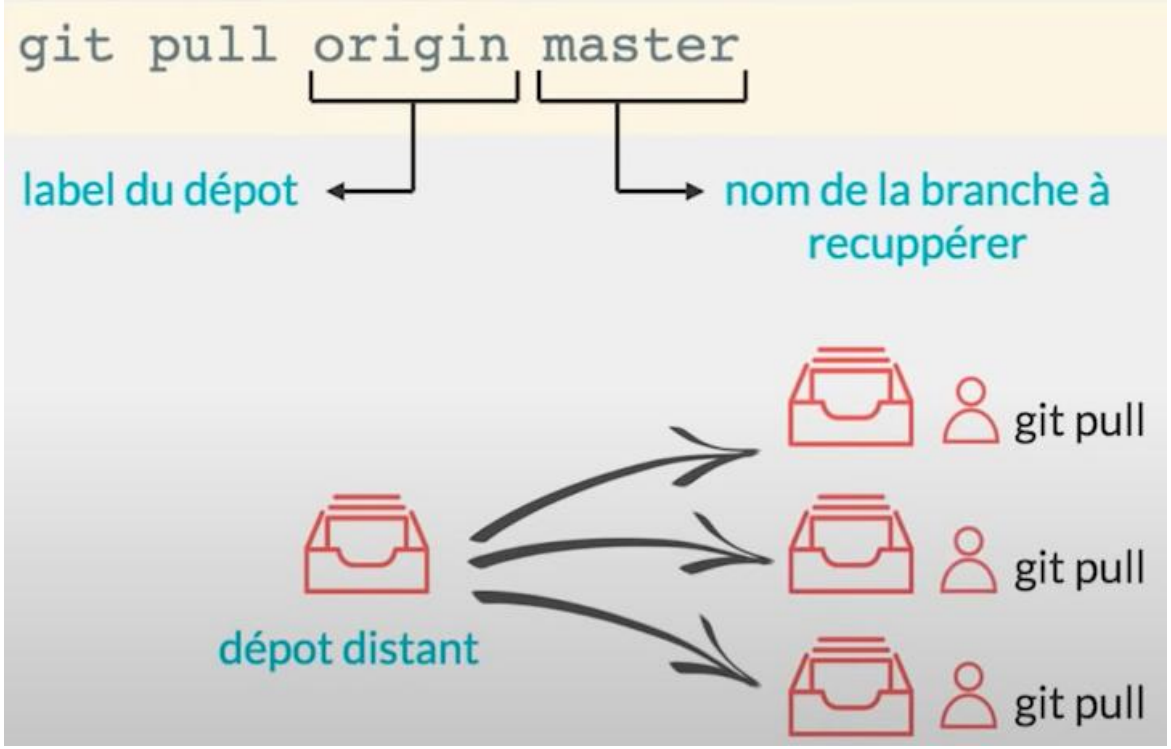
Git avec IDE

- Maintenant vous pouvez utilisez Git en ligne de commande ou sur IntelliJ :



Dépôt distant GitHub

- Comment l'équipe récupère ce projet?
- Celui qui a mis le projet sur GitHub doit donner les accès à ces collègues (Sur l'interface Github: **Collaborators -> Settings**)
- Les collègues doivent se connecter au dépôt distant (git remote add …) et faire un pull :



Dépôt distant GitHub

- Comme vous allez travailler en équipe, les autres membres d'un même groupe, récupèrent le code de GitHub avec la commande ci-dessous :

- **git clone git://github.com/mhassini/timesheet-devops.git**

- C'est équivalent à :

- git init
- git remote add origin git://github.com/mhassini/timesheet-devops.git
- git pull origin master

TP1 : GIT

Vous avez tous fait des manipulations sur le projet exemple timesheet-devops. Maintenant, vous allez travailler en équipe sur le **vrai projet** :

1- Un seul membre par équipe récupère **le projet déjà prêt du Drive, dossier code-source** , le dézippe dans le workspace IntelliJ en local et le pousse sur Github, en suivant les étapes ci-dessus.

2- Celui qui a mis le projet sur GitHub doit donner les accès à ces collègues (Sur l'interface github: **Collaborators -> Settings**)

3- Les autres membres de chaque équipe récupèrent ce projet de github (et non du drive).

4- Chacun créé sa propre branche et la pousse sur github.

5- Mettez à jour le fichier XLS partagé par votre enseignant sur le Classroom avec la liste des membres de votre groupe.

6- Invitez votre enseignant sur le repository du groupe sur github.com

TP2 : GIT – MAVEN – JENKINS

7- Chacun crée un Job de type Pipeline su son Jenkins « **Prenom_NOM_CLASSE** », avec **deux étapes** :

- **Récupération du code de sa propre branche**
- **Lancement de la commande Maven qui nettoie le projet et le compile**