

Outil de travail collaboratif (Git)



Bureau E204

Plan du cours

- Système de contrôle de version
- SVN
- GIT
- Installation GIT
- Configuration GitHub
- Commandes GIT
- Conflit de merge
- Commande avancée GIT
- Travail à faire

Système de contrôle de version

- Les systèmes de contrôle de version sont des outils logiciels qui permettent aux équipes de développement de gérer les changements apportés au code source au fil du temps.
- Avec l'accélération des environnements de développement, les systèmes de contrôle de version aident les équipes de développement à travailler plus rapidement et plus intelligemment. Ils sont particulièrement utiles pour les équipes DevOps, car ils leur permettent de réduire le temps de développement et d'assurer le succès des déploiements.

Système de contrôle de version

- Les logiciels de contrôle de version gardent une trace de chaque changement apporté au code dans un type spécial de base de données. Si une erreur est commise, les développeurs peuvent revenir en arrière et comparer les versions antérieures du code, ce qui leur permet de corriger l'erreur tout en minimisant les perturbations pour tous les membres de l'équipe.

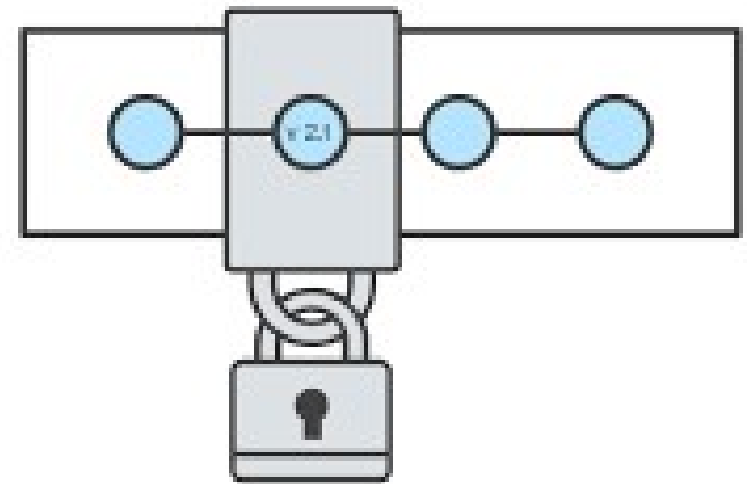
Qui utilise l'outil de gestion de versionnig ?

Système de contrôle de version

- L'idée que l'outil de gestion de version est utilisé uniquement par les développeurs est **fausse**.
- En effet, cet outil est utile pour toute personne impliquée dans le domaine de l'IT (architecte, administrateur, équipe infrastructure, etc..).
- L'équipe infrastructure ainsi que l'équipe opérationnelle utilise cet outil pour sauvegarder et tracker les fichiers de configurations.

Système de contrôle de version

- Avantages des systèmes de contrôle de version:
 - ✓ Avoir un historique complet des changements à long terme de chaque fichier.
 - ✓ Branching et merges.
 - ✓ Traçabilité.



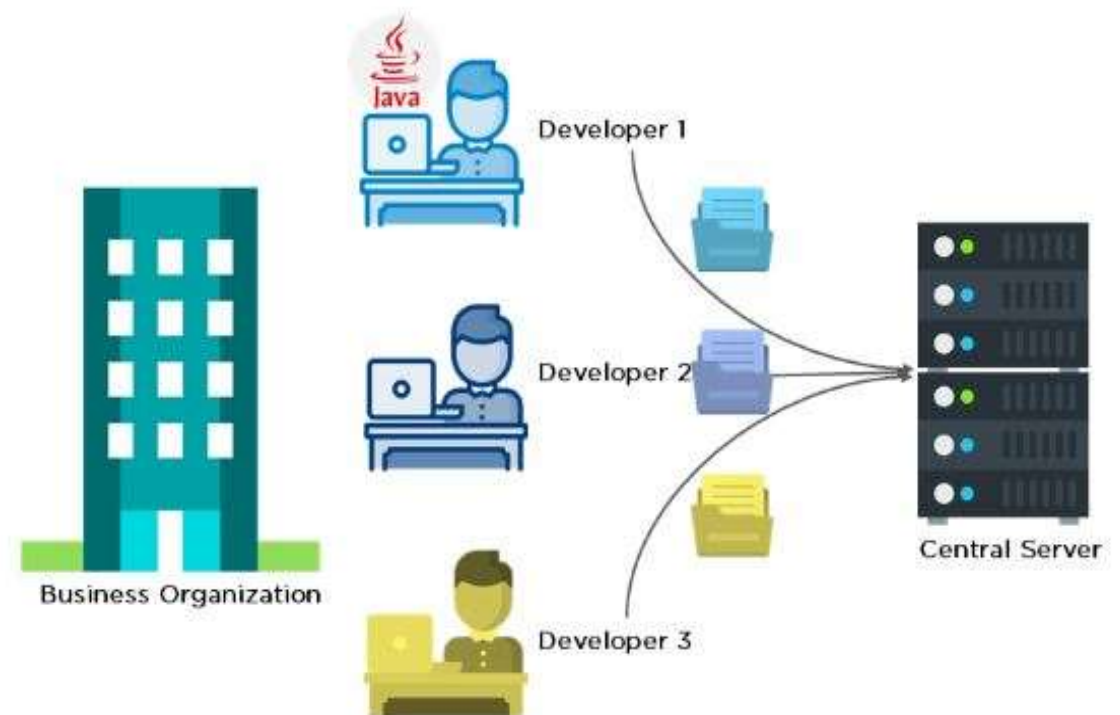
→ Outils de Système de contrôle de version: versionner, historiser, faire travail l'équipe ensemble, traçabilité,

→ Exemples: SVN, GIT, ...

SVN

- SVN = Apache Subversion
- C'est un outil de versionning centralisé
- On a une architecture client-serveur. Le projet se trouve dans un repo central.

Cours réalisé



SVN



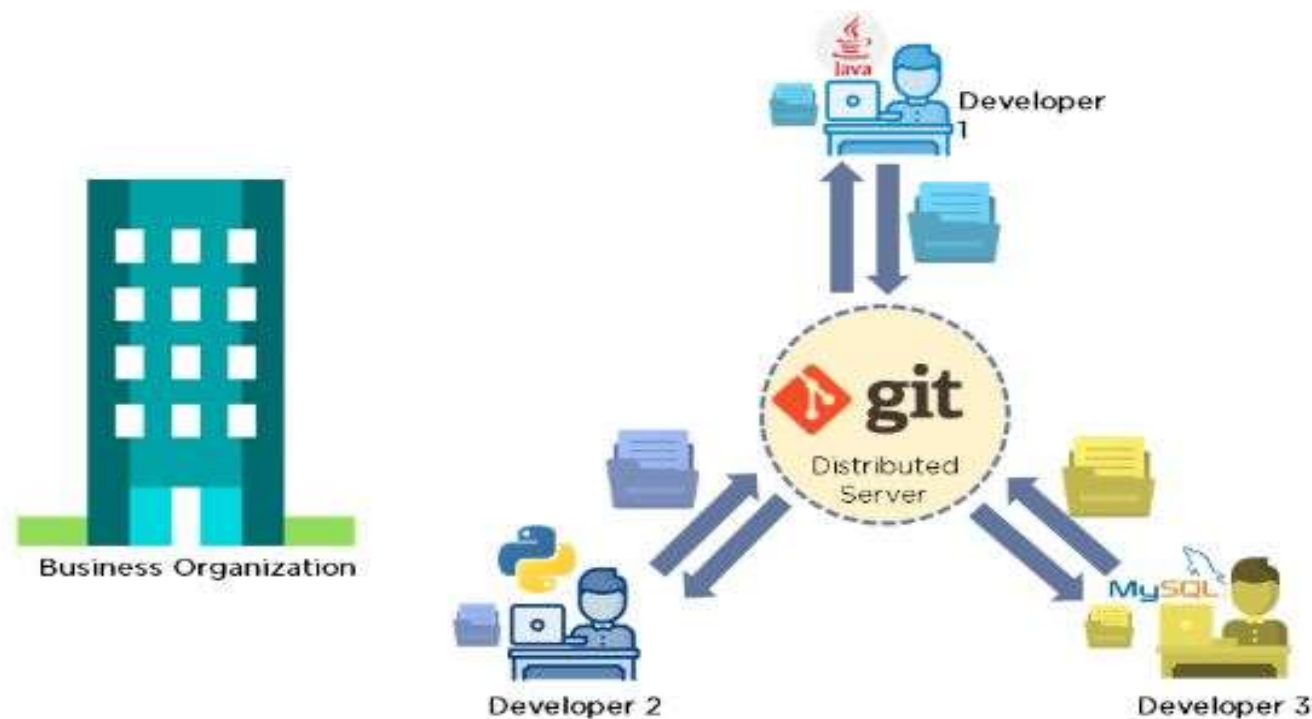
- Avec SVN, le développeur extrait le projet en local et il modifie ses propres fichiers puis il fait le commit au repo central.
 - Difficulté de se placer sur une version antérieure: Toute modification sur une classe engendre l'impossibilité de retour en arrière (à l'exception de la sauvegarde des différents versions localement)
 - Il doit toujours communiquer avec le repo central (nécessite toujours une connexion réseau) pour être à jour.
 - Beaucoup de requetes entre le serveur et l'utilisateur

GIT

- C'est un outil de versioning décentralisé ou bien distribué open source crée par Linus Torvalds le créateur de Linux.
- Chaque développeur a une copie complète du projet en local.
- Git suit l'évolution des fichiers sources et garde les anciennes versions de chacun d'eux.
- Git permet de retrouver les différentes versions d'un fichier ou d'un lot de fichiers connexes.
- Chaque modification dans le code source peut être suivi par les autres développeurs.
- Communication régulière entre les développeurs.
- Git supporte **le développement linéaire** à travers les différentes branches.
- GIT est le système de contrôle de version le plus utilisé.

GIT

- Un système de gestion des conflits
- Toutes les données sur notre machine
- La plupart des opérations ne nécessitent pas de connexion réseau, car elles ne travaillent que sur votre clone du référentiel.



GIT

- Dans git, nous distinguons deux repositories:

→ Local  **git**: Le dossier .git (Dossier caché)

→ Distant

Nom	Modifié le	Type	Taille
.git	11/10/2022 16:10	Dossier de fichiers	
.idea	12/10/2022 09:35	Dossier de fichiers	

tpAchatProject > .git >			
Nom	Modifié le	Type	Taille
hooks	04/10/2022 12:44	Dossier de fichiers	
info	04/10/2022 12:44	Dossier de fichiers	
logs	04/10/2022 12:44	Dossier de fichiers	
objects	11/10/2022 16:10	Dossier de fichiers	
refs	04/10/2022 12:45	Dossier de fichiers	
COMMIT_EDITMSG	10/10/2022 22:34	Fichier	1 Ko
config	04/10/2022 20:49	Fichier	1 Ko
description	04/10/2022 12:44	Fichier	1 Ko
FETCH_HEAD	06/10/2022 13:26	Fichier	1 Ko
HEAD	04/10/2022 12:45	Fichier	1 Ko
index	11/10/2022 16:10	Fichier	8 Ko
ORIG_HEAD	06/10/2022 13:26	Fichier	1 Ko

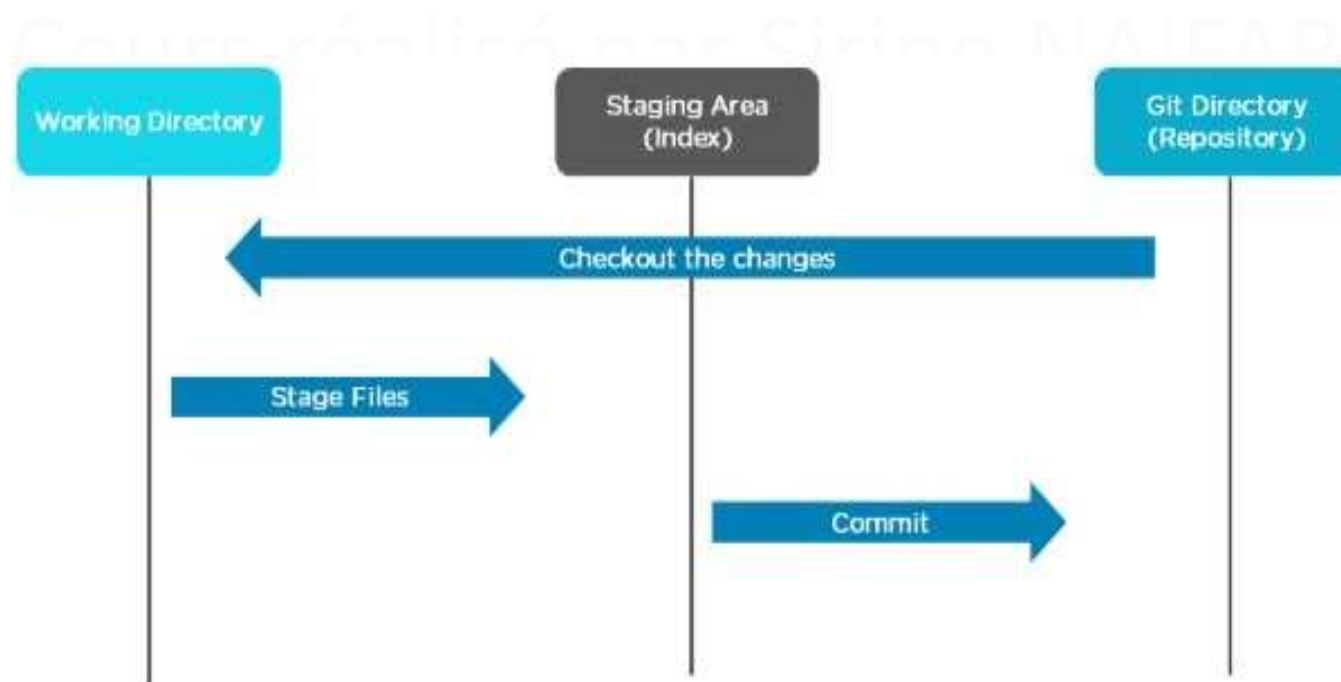


- Dans git, nous distinguons trois zones (se trouve dans .git) :

Working Area : Modification des fichiers sur la machine locale.

Stagging Area : Préparer des copies et ajouter des snapshots de ces fichiers.

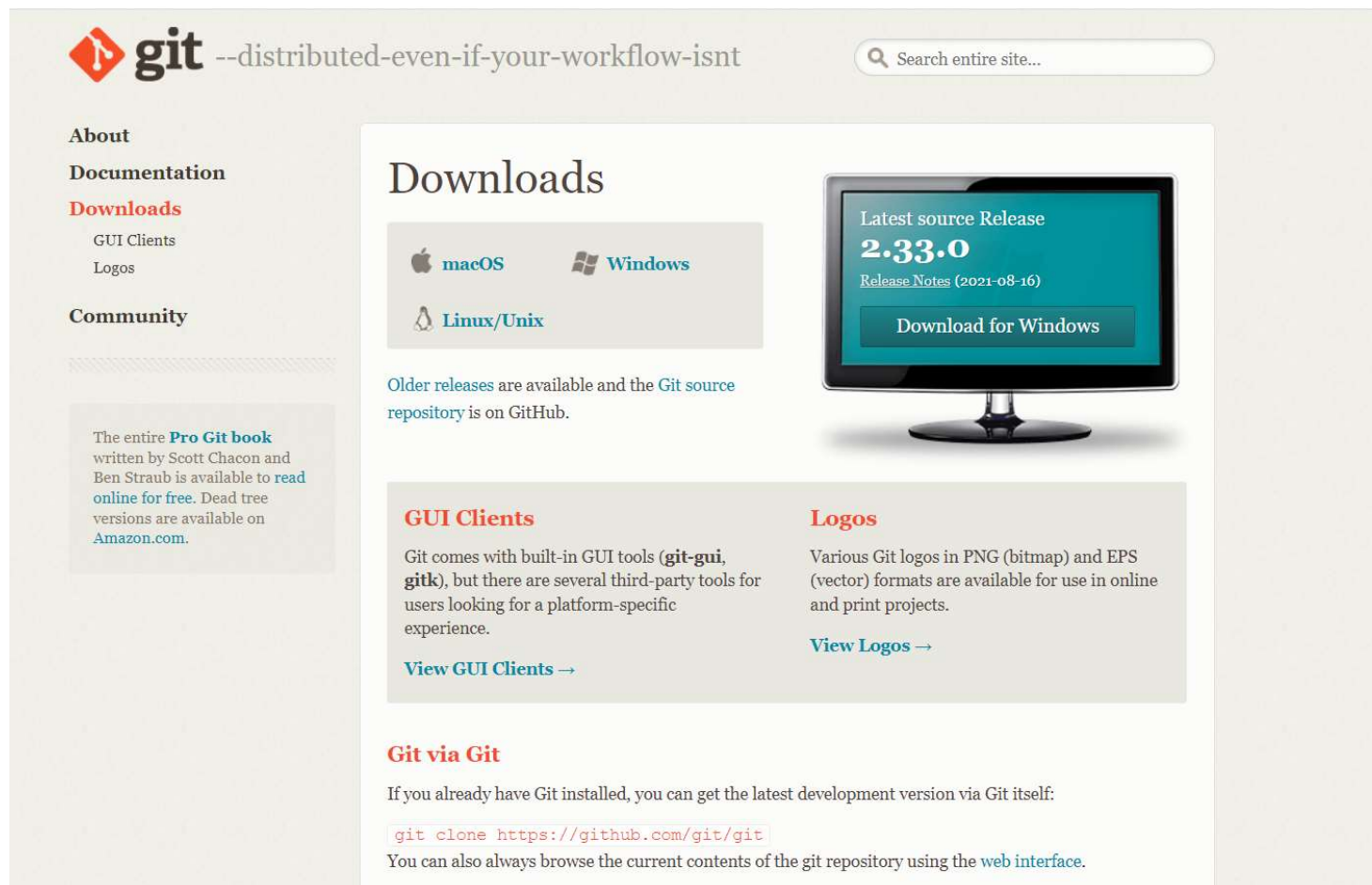
Git Area (repository) : Commiter les snapshots d'une façon définitive sur le répertoire de GIT.



Installation GIT - Windows



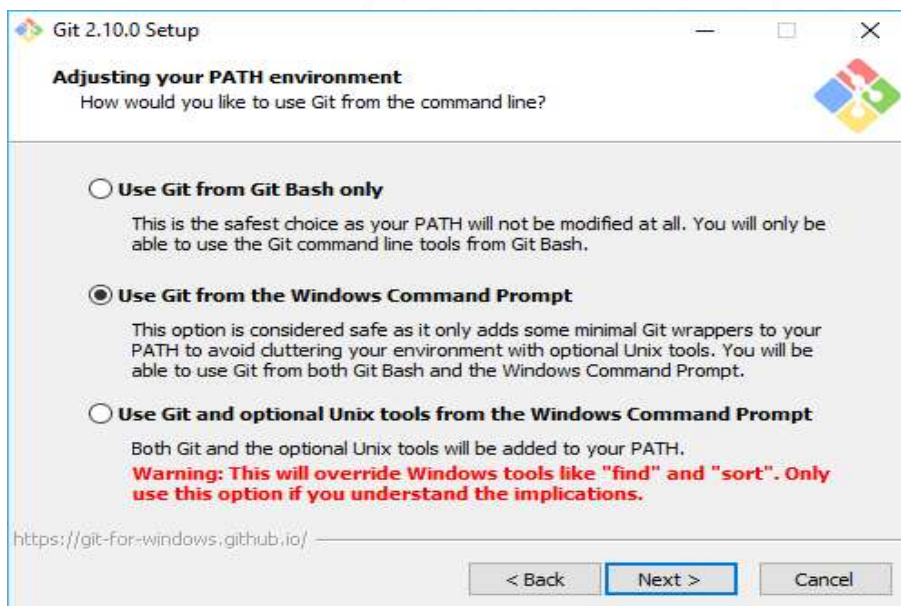
- Télécharger GIT depuis le site officiel <https://git-scm.com/downloads>



Installation GIT - Windows



- Lancer l'exécutable



Vérifier que GIT est installé sur votre machine:

```
C:\Users\sirin>git version  
git version 2.30.0.windows.2
```

Ouvrir git bash et lancer

```
sirin@LAPTOP-CH750RI6 MINGW64 ~  
$ git --version  
git version 2.30.0.windows.2
```

Installation GIT - Ubuntu



- Pour installer Git, vous devez exécuter la command suivante:
 - `sudo apt install git`
- Pour vérifier l'installation, vous devez exécuter la commande
 - `git --version`

Cours réalisé par Sirine NAIFAR

```
[root@localhost vagrant]# git --version
git version 1.8.3.1
```



GitHub

GitHub est une plateforme de "codes" open-source où nous pouvons collaborer sur un projet ou accéder à un code source public.

Github permet d'utiliser le versioning de Git sans avoir à apprendre Git qui se gère à la ligne de commande.



Configuration GitHub

- Accéder à l'adresse <http://www.github.com> et faire l'inscription

Welcome to GitHub!
Let's begin the adventure

Enter your email

→

Continue

Would you like to receive product updates and announcements via email?
Type "y" for yes or "n" for no

✓ n

Verify your account

✓

Create account



Configuration GitHub

- Finaliser le process de la création de votre compte

A screenshot of the GitHub account verification screen. It has a dark blue background. The text 'You're almost done!' is at the top in white. Below it, 'We sent a launch code to' is also in white. Then, '→ Enter code' is in a light purple color. Below this is a row of six square input boxes; the first box contains a white vertical line. At the bottom, in a smaller white font, it says 'Didn't get your email? [Resend the code](#) or [update your email address](#).'

You're almost done!

We sent a launch code to

→ Enter code

Didn't get your email? [Resend the code](#) or [update your email address](#).



Configuration GitHub

- Afficher l'interface GitHub

The screenshot shows the GitHub homepage. At the top is a dark navigation bar with the GitHub logo, a search bar, and links for Pull requests, Issues, Marketplace, and Explore. On the left sidebar, there's a section 'Create your first project' with buttons for 'Create repository' and 'Import repository', and a 'Recent activity' section. The main content area features a large green box titled 'Learn Git and GitHub without any code!' with buttons for 'Read the guide' and 'Start a project'. Below this is an 'Introduce yourself' section with a text area containing a template for a README file. On the right, there's a 'Save the Date!' notification for GitHub Universe.

Create your first project
Ready to start building? Create a repository for a new idea or bring over an existing repository to keep contributing to it.

[Create repository](#) [Import repository](#)

Recent activity
When you take actions across GitHub, we'll provide links to that activity here.

Learn Git and GitHub without any code!
Using the Hello World guide, you'll create a repository, start a branch, write comments, and open a pull request.

[Read the guide](#) [Start a project](#)

All activity

Introduce yourself
The easiest way to introduce yourself on GitHub is by creating a README in a repository about you! You can start here:

```
sirineDevOps / README.md
```

```
1 - 🙋 Hi, I'm @sirineDevOps
2 - 👀 I'm interested in ...
3 - 📖 I'm currently learning ...
4 - ❤️ I'm looking to collaborate on ...
5 - 📧 How to reach me ...
6
```

[Dismiss this](#) [Continue](#)

Save the Date!
GitHub Universe is coming October 27 and 28. From product deep dives to interactive roundtables, you'll gather the tips, tools, and connections to help you do the best work of your life.

[Learn more](#)



Configuration GitHub

- Pour partager le code source entre les collaborateurs, il faut créer un repo dans le GitHub

Create your first project

Ready to start building? Create a repository for a new idea or bring over an existing repository to keep contributing to it.

Create repository

Import repository



Configuration GitHub

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?

[Import a repository.](#)

Owner *


 sirineDevOps ▾

Repository name *

/ SpringDataJPA-CrudRepo ✓

Great repository names are short and memorable. Need inspiration? How about [musical-system](#)?

Description (optional)

- ☐  **Public**
Anyone on the internet can see this repository. You choose who can commit.
- ☒  **Private**
You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

- ☐ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)
- ☐ **Add .gitignore**
Choose which files not to track from a list of templates. [Learn more.](#)
- ☐ **Choose a license**
A license tells others what they can and can't do with your code. [Learn more.](#)

Create repository



Configuration GitHub

🔒 sirineDevOps / SpringDataJPA-CrudRepo Private

👁️ Unwatch 1 ⭐ Star 0 🍴 Fork 0

<> Code 🔔 Issues 🔗 Pull requests 🔄 Actions 📁 Projects 🛡️ Security 📊 Insights ⚙️ Settings

Quick setup — if you've done this kind of thing before

📄 Set up in Desktop or HTTPS SSH `https://github.com/sirineDevOps/SpringDataJPA-CrudRepo.git` 📄

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

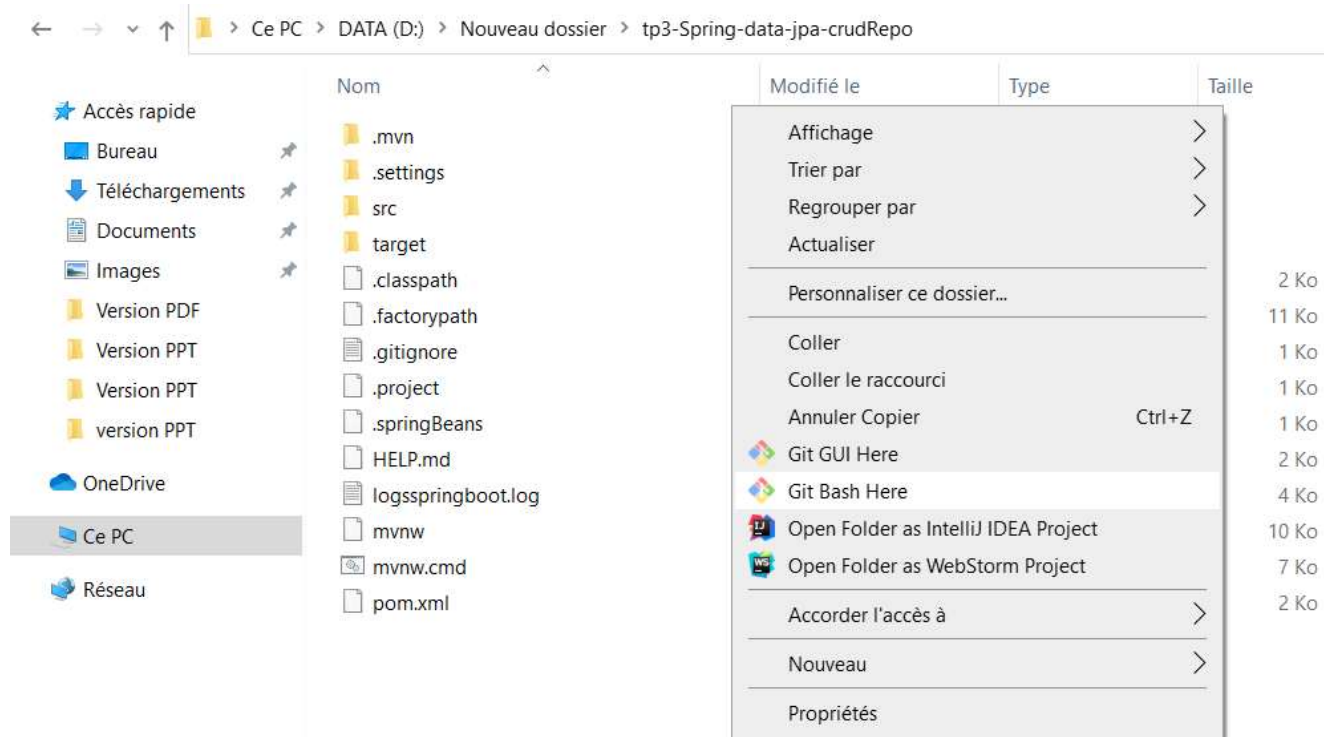
```
echo "# SpringDataJPA-CrudRepo" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/sirineDevOps/SpringDataJPA-CrudRepo.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/sirineDevOps/SpringDataJPA-CrudRepo.git
git branch -M main
git push -u origin main
```

Commandes GIT

- Maintenant, nous désirons ajouter le projet déjà prêt dans notre repo local sur notre compte GIT. Pour cela, nous devons opérer ainsi :
 - ✓ Pointer sur le dossier du projet et ouvrir l'invite de commande GitBash



→ Seuls le dossier **src** et le fichier **pom.xml** sont à envoyer sur Git

Commandes GIT – Configuration initiale

- Configurer l'utilisateur et l'email

git config --global user.name "prénom nom"

git config --global user.email "email "

```
sirin@LAPTOP-CH750RI6 MINGW64 ~  
$ git config --global user.name "Sirine.NAIFAR"  
  
sirin@LAPTOP-CH750RI6 MINGW64 ~  
$ git config --global user.email "sirine.naifer@esprit.tn"
```

- Pour vérifier la configuration

git config --list

```
sirin@LAPTOP-CH750RI6 MINGW64 ~  
$ git config --list  
diff.astextplain.textconv=astextplain  
filter.lfs.clean=git-lfs clean -- %f  
filter.lfs.smudge=git-lfs smudge -- %f  
filter.lfs.process=git-lfs filter-process  
filter.lfs.required=true  
http.sslbackend=openssl  
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt  
core.autocrlf=true  
core.fscache=true  
core.symlinks=false  
pull.rebase=false  
credential.helper=manager-core  
credential.https://dev.azure.com.usehttppath=true  
init.defaultbranch=master  
user.name=prénom nom  
user.name=Sirine.NAIFAR  
user.email=sirine.naifer@esprit.tn
```


Commandes GIT – Initialisation et Git status

- Ensuite, on va initialiser le projet comme dépôt GIT à travers la commande

git init -b main

- ✓ GIT pourra exécuter les différentes commandes et tracker les modifications dans les fichiers de ce répertoire

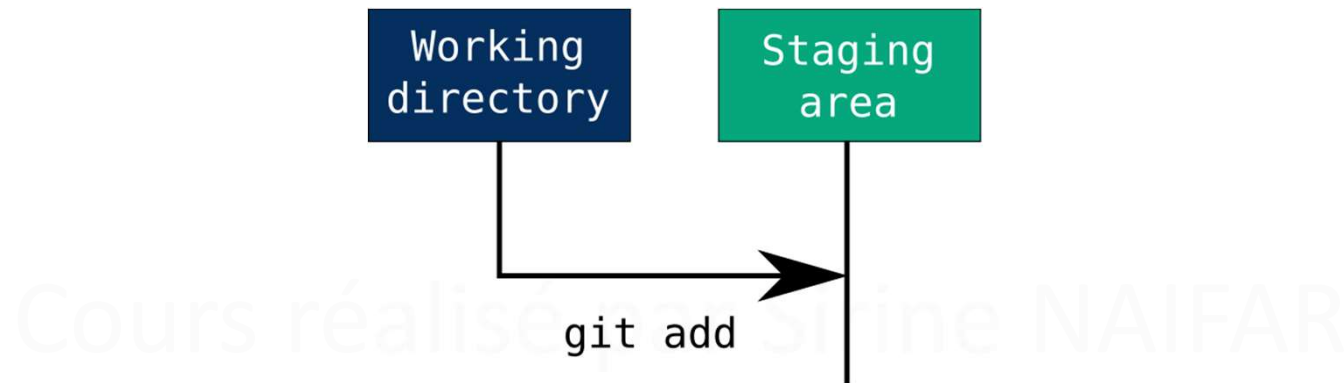
```
MINGW64:/d/Nouveau dossier/tp3-Spring-data-jpa-crudRepo  
sirin@LAPTOP-CH750RI6 MINGW64 /d/Nouveau dossier/tp3-Spring-data-jpa-crudRepo  
$ git init -b main  
Initialized empty Git repository in D:/Nouveau dossier/tp3-Spring-data-jpa-crudRepo/.git/
```

- Pour vérifier l'état des fichiers : **git status**

```
$ git status  
On branch main  
  
No commits yet  
  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
    .gitignore  
    .mvn/  
    logsspringboot.log  
    mvnw  
    mvnw.cmd  
    pom.xml  
    src/  
  
nothing added to commit but untracked files present (use "git add" to track)
```

Commandes GIT – Git add

- On va déplacer les fichiers du repo local au staging area, c'est pour cela il faut lancer la commande **git add**



- Vous pouvez ajouter un fichier individuel ou des groupes de fichiers. Pour ajouter un seul fichier, utilisez **git add <nom_fichier> <nom_fichier> ...**

Exemple: git add pom.xml src

```
sirin@LAPTOP-CH750RI6 MINGW64 /d/Nouveau dossier/tp3-Spring-data-jpa-crudRepo (main)
$ git add pom.xml
warning: LF will be replaced by CRLF in pom.xml.
The file will have its original line endings in your working directory
```

Commandes GIT – Git add

- Pour ajouter tous les fichiers que vous avez édités en même temps, vous pouvez utiliser : **git add – all**
- Si vous faites **git status**

```
sirin@LAPTOP-CH750RI6 MINGW64 /d/Nouveau dossier/tp3-Spring-data-jpa-crudRepo (main)
$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   pom.xml
    new file:   src/main/java/tn/esprit/spring/Tp3SpringDataJpaCrudRepoApplication.java
    new file:   src/main/java/tn/esprit/spring/entity/Role.java
    new file:   src/main/java/tn/esprit/spring/entity/User.java
    new file:   src/main/java/tn/esprit/spring/repository/UserRepository.java
    new file:   src/main/java/tn/esprit/spring/service/DemoService.java
    new file:   src/main/java/tn/esprit/spring/service/IUserService.java
    new file:   src/main/java/tn/esprit/spring/service/UserService.java
    new file:   src/main/resources/application.properties
    new file:   src/test/java/tn/esprit/spring/Tp3SpringDataJpaCrudRepoApplicationTests.java

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    .gitignore
    .mvn/
    logsspringboot.log
    mvnw
    mvnw.cmd
```

Commandes GIT

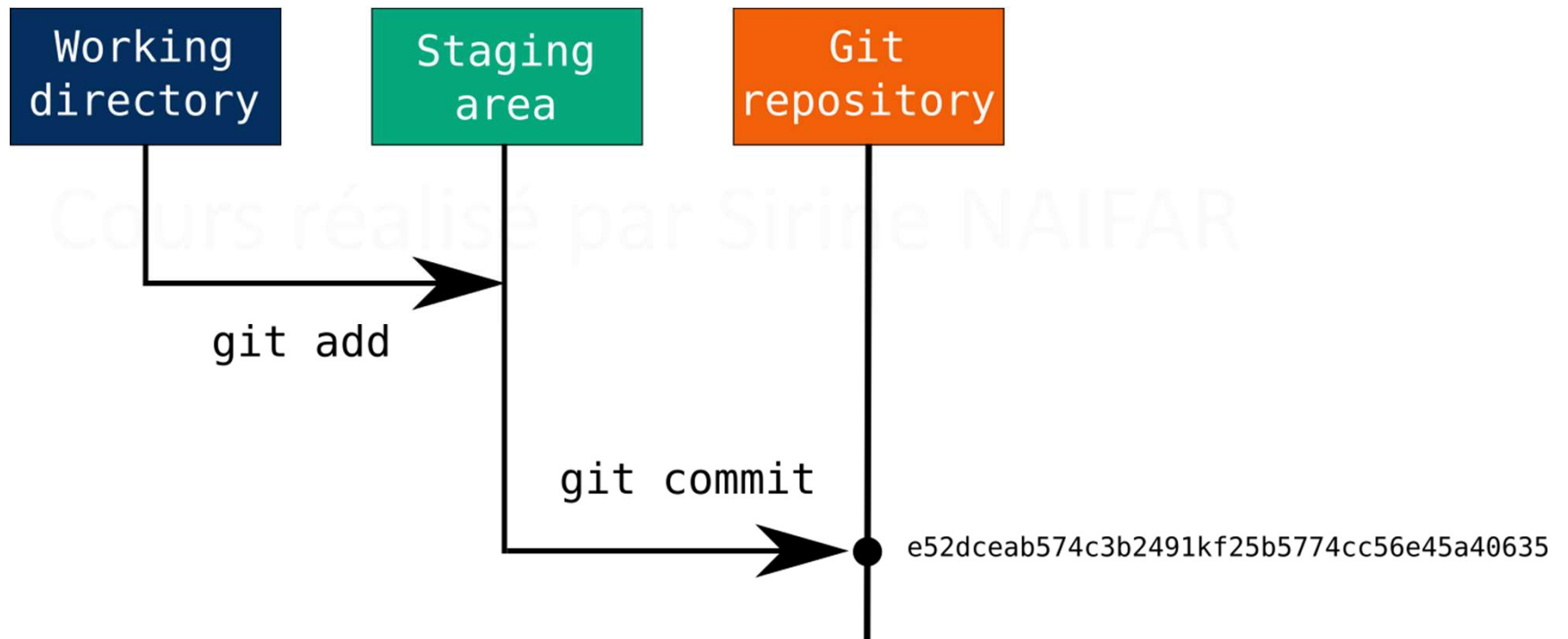
- Ne pas inclure les fichiers :

- ✓ .classpath
- ✓ .mvn/
- ✓ .project
- ✓ .settings/
- ✓ .springBeans
- ✓ HELP.md
- ✓ mvnw
- ✓ mvnw.cmd

➔ Ces fichiers sont locaux et créés automatiquement, le développeur ne le changera jamais.

Commandes GIT – Git commit

- La commande **git commit** prend toute la staging Area et crée un SnapShot permanent de l'état actuel du repository associé à un identifiant unique.



Commandes GIT – Git commit

- Première méthode :

```
sirin@LAPTOP-CH750RI6 MINGW64 ~/Desktop/testProject (sirine_branch)
$ git commit
```

MINGW64:/c/Users/sirin/Desktop/testProject

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch sirine_branch
# Changes to be committed:
#   new file:   .gitignore
#   new file:   .mvn/wrapper/maven-wrapper.jar
#   new file:   .mvn/wrapper/maven-wrapper.properties
#   new file:   mvnw
#   new file:   mvnw.cmd
#   modified:   src/main/java/com/example/entities/Animal.java
#   new file:   src/main/resources/application.properties
#   new file:   src/test/java/com/example/TestProjectApplicationTests.java
#
~
~
~
```


Commandes GIT – Git commit et Git log

- Deuxième méthode :

```
sirin@LAPTOP-CH750RI6 MINGW64 /d/Nouveau dossier/tp3-Spring-data-jpa-crudRepo (main)
$ git commit -m 'First commit'
[main (root-commit) 13966ef] First commit
10 files changed, 353 insertions(+)
create mode 100644 pom.xml
create mode 100644 src/main/java/tn/esprit/spring/Tp3SpringDataJpaCrudRepository.java
create mode 100644 src/main/java/tn/esprit/spring/entity/Role.java
create mode 100644 src/main/java/tn/esprit/spring/entity/User.java
create mode 100644 src/main/java/tn/esprit/spring/repository/UserRepository.java
create mode 100644 src/main/java/tn/esprit/spring/service/DemoService.java
create mode 100644 src/main/java/tn/esprit/spring/service/IUserService.java
create mode 100644 src/main/java/tn/esprit/spring/service/UserService.java
create mode 100644 src/main/resources/application.properties
create mode 100644 src/test/java/tn/esprit/spring/Tp3SpringDataJpaCrudRepositoryTests.java
```

- Lancer la commande **git log** pour voir les commits:

```
$ git log
commit 13966ef4dfd9da09f753e4a08afe26daecc8cb (HEAD -> main)
Author: Sirine.NAIFAR <sirine.naifer@esprit.tn>
Date: Mon Oct 4 18:07:02 2021 +0100

    First commit
```

Commandes GIT - Exemple

- 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 :

```
sirin@LAPTOP-CH750RI6 MINGW64 /d/Nouveau dossier/tp3-Spring-data-jpa-crudRepo (main)
$ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   src/main/java/tn/esprit/spring/entity/User.java

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        .mvn/
        logsspringboot.log
        mvnw
        mvnw.cmd

no changes added to commit (use "git add" and/or "git commit -a")
```


Commandes GIT - Exemple

```
sirin@LAPTOP-CH750RI6 MINGW64 /d/Nouveau dossier/tp3-Spring-data-jpa-crudRepo (main)
$ git add src/main/java/tn/esprit/spring/entity/User.java
```

```
sirin@LAPTOP-CH750RI6 MINGW64 /d/Nouveau dossier/tp3-Spring-data-jpa-crudRepo (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   src/main/java/tn/esprit/spring/entity/User.java

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        .mvn/
        logsspringboot.log
        mvnw
        mvnw.cmd
```

```
sirin@LAPTOP-CH750RI6 MINGW64 /d/Nouveau dossier/tp3-Spring-data-jpa-crudRepo (main)
$ git commit -m "modification user"
[main 4934c49] modification user
1 file changed, 2 insertions(+)
```

```
sirin@LAPTOP-CH750RI6 MINGW64 /d/Nouveau dossier/tp3-Spring-data-jpa-crudRepo (main)
$ git log
commit 4934c493708fe182b44e75b052f370737db71fc5 (HEAD -> main)
Author: Sirine.NAIFAR <naifar.sirine@gmail.com>
Date:   Mon Oct 4 18:23:34 2021 +0100

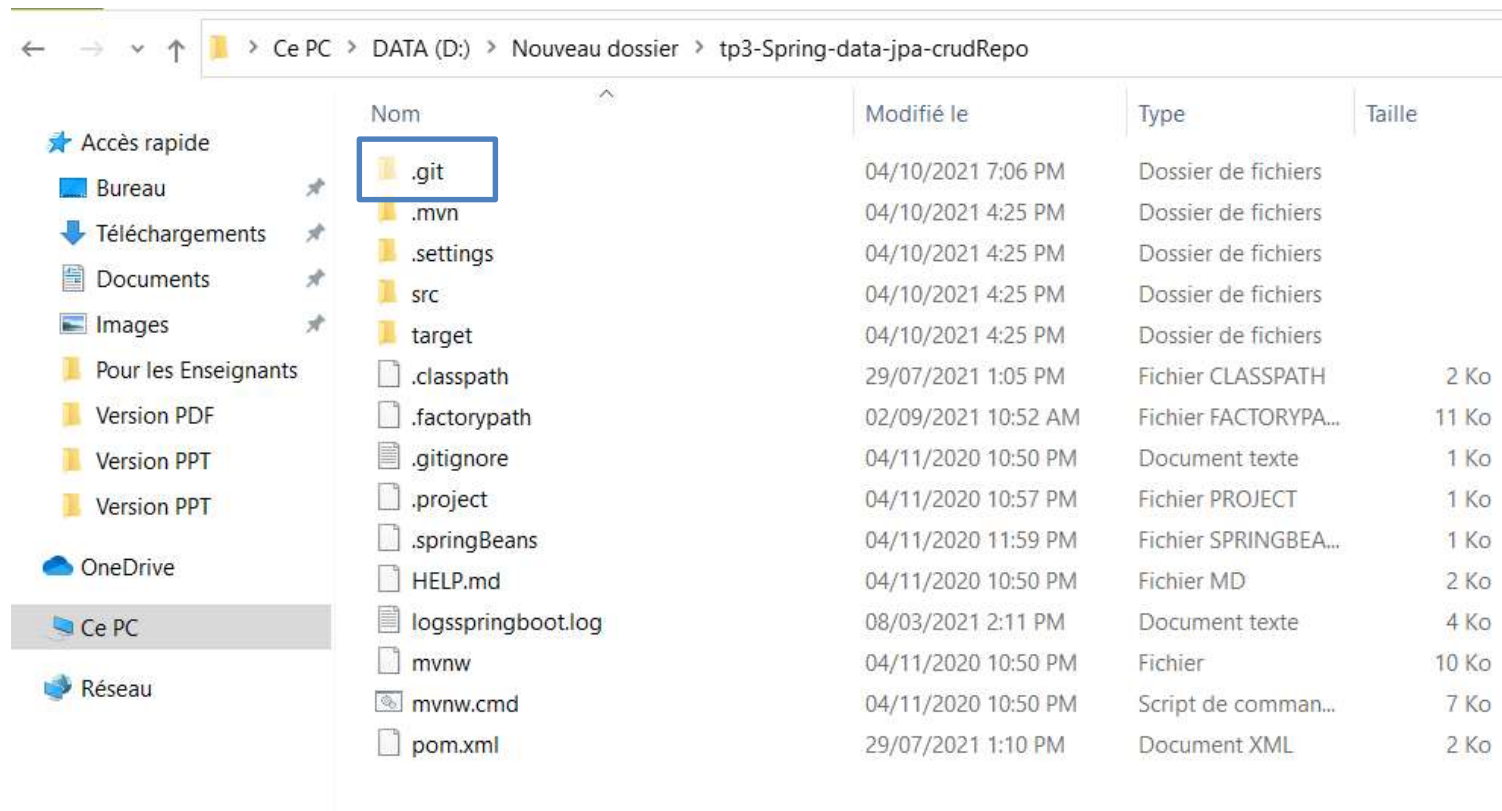
    modification user

commit 13966ef4dfd9da09f753e4a08afe26daecc8cb
Author: Sirine.NAIFAR <naifar.sirine@gmail.com>
Date:   Mon Oct 4 18:07:02 2021 +0100

    First commit
```

Remarque

- 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)




Commandes GIT – Remote

- Pour publier les changements locaux et les charger vers un dépôt centralisé (GitHub), il faut tout d'abord lier les deux repos (local et central) à travers cette commande:

git remote add origin < lien vers le repo central >

Quick setup — if you've done this kind of thing before

 Set up in Desktop

or

HTTPS

SSH

<https://github.com/sirineDevOps/SpringDataJPA-CrudRepo.git>



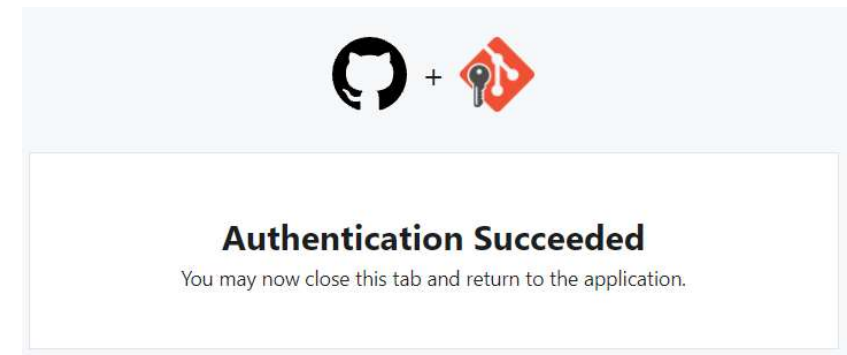
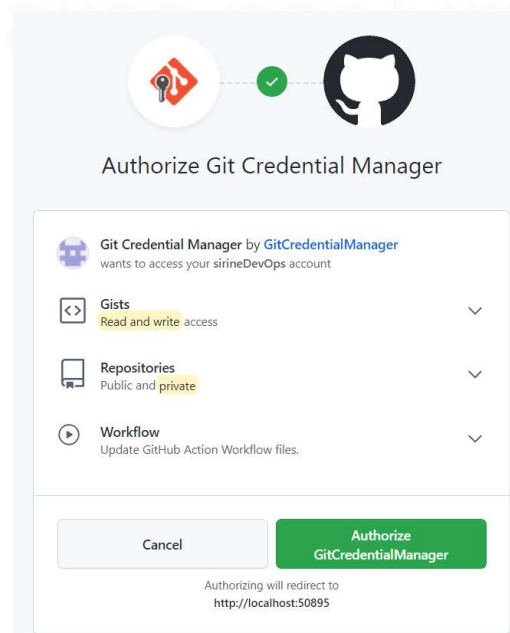
Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a README, LICENSE, and .gitignore.

```
sirin@LAPTOP-CH750RI6 MINGW64 /d/Nouveau dossier/tp3-Spring-data-jpa-crudRepo (main)
$ git remote add origin https://github.com/sirineDevOps/SpringDataJPA-CrudRepo.git
```

Commandes GIT – Push

- Après qu'un dépôt local a été modifié, un **push** est exécuté pour partager les changements avec les membres de l'équipe distants.

```
sirin@LAPTOP-CH750RI6 MINGW64 /d/Nouveau dossier/tp3-Spring-data-jpa-crudRepo (main)  
$ git push -u origin main
```



Commandes GIT – Push




```
sirin@LAPTOP-CH750RI6 MINGW64 /d/Nouveau dossier/tp3-Spring-data-jpa-crudRepo (main)
$ git push -u origin main

Enumerating objects: 47, done.
Counting objects: 100% (47/47), done.
Delta compression using up to 12 threads
Compressing objects: 100% (33/33), done.
Writing objects: 100% (47/47), 5.90 KiB | 1.47 MiB/s, done.
Total 47 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (4/4), done.
To https://github.com/sirineDevOps/SpringDataJPA-CrudRepo.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

🔒 sirineDevOps / **SpringDataJPA-CrudRepo** Private

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Security](#) [Insights](#) [Settings](#)

🔗 main ▾ 🔗 1 branch 🔗 0 tags [Go to file](#) [Add file ▾](#) [Code ▾](#)

	sirineDevOps modification user par la deuxième branche	1c1a217 31 minutes ago	🕒 3 commits
	src	modification user par la deuxième branche	31 minutes ago
	pom.xml	First commit	1 hour ago

Add a README with an overview of your project. [Add a README](#)

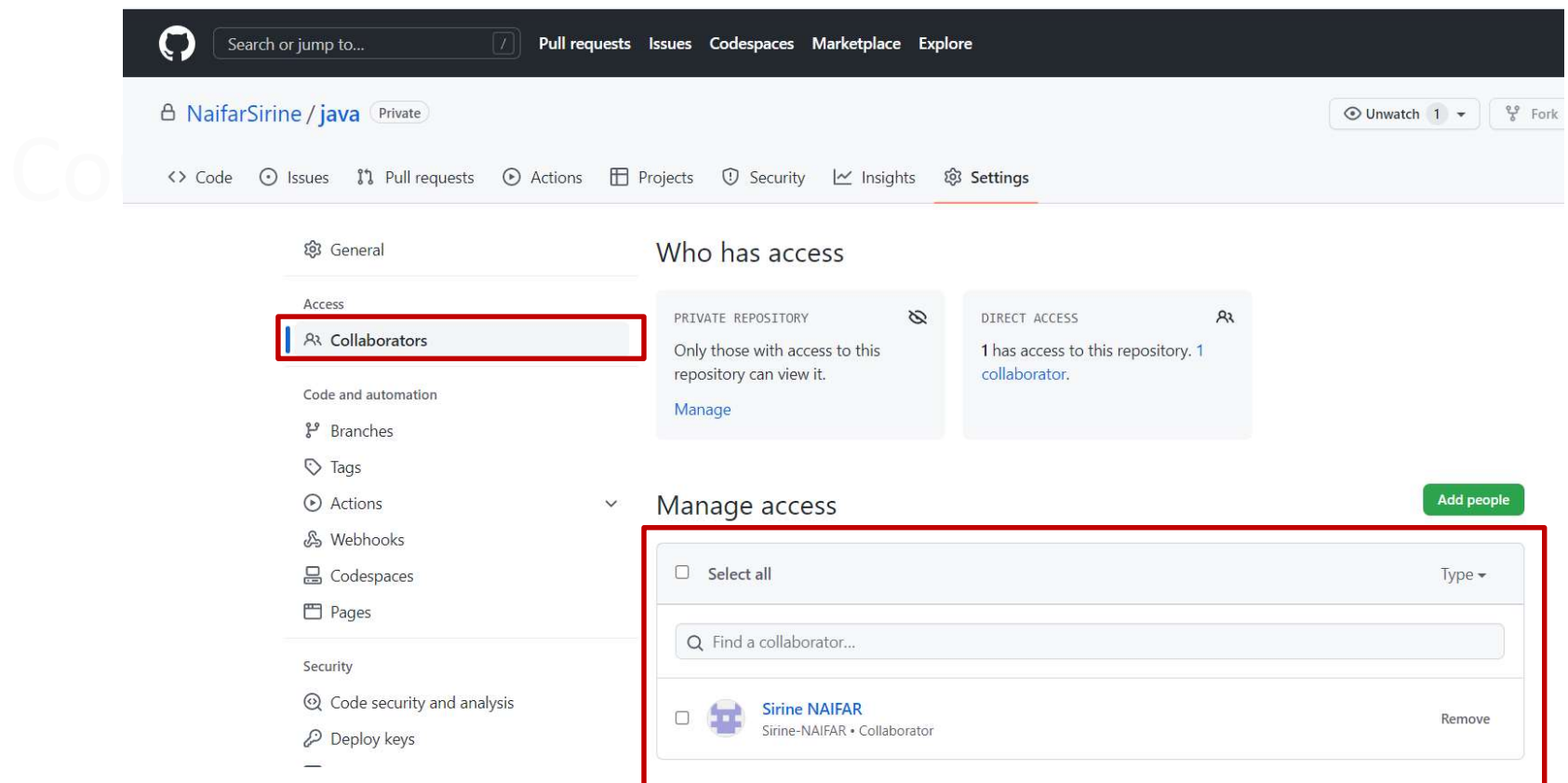
Question

Quelle est l'étape à effectuer pour assurer le partage entre les membres de l'équipe avec GIT ?



Commandes GIT – Collaborateurs

- En tant que master, je dois inviter les autres membres de l'équipe pour partager le code source de l'application. Pour envoyer l'invitation, il faut accéder au repo central et accéder aux settings:



Commandes GIT – Clone

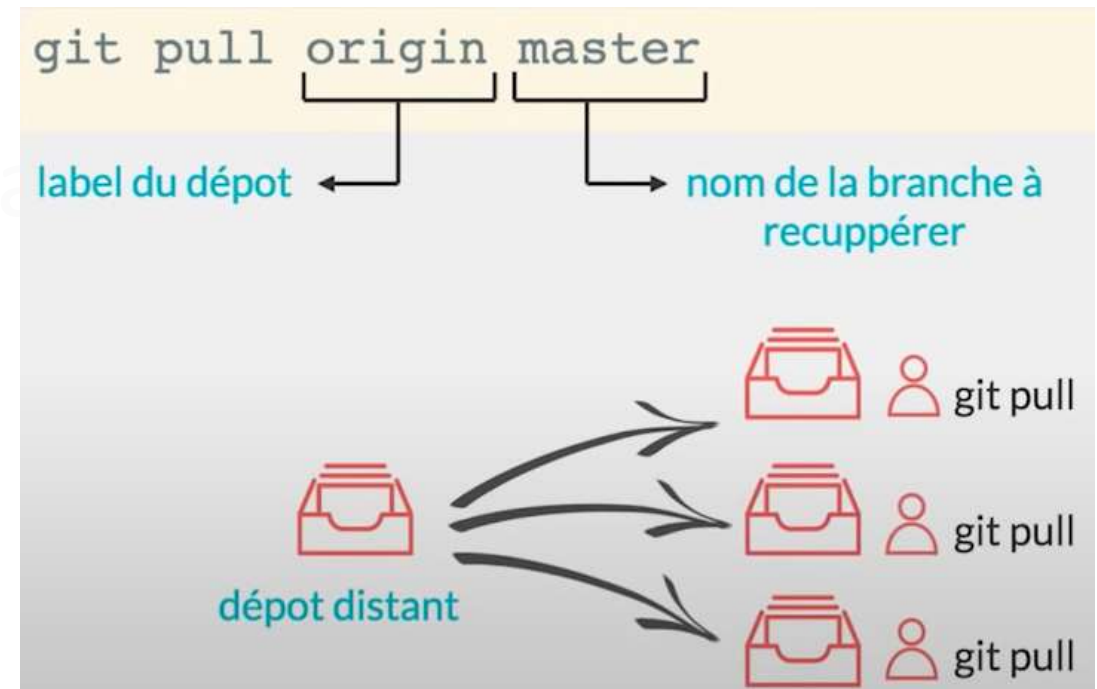
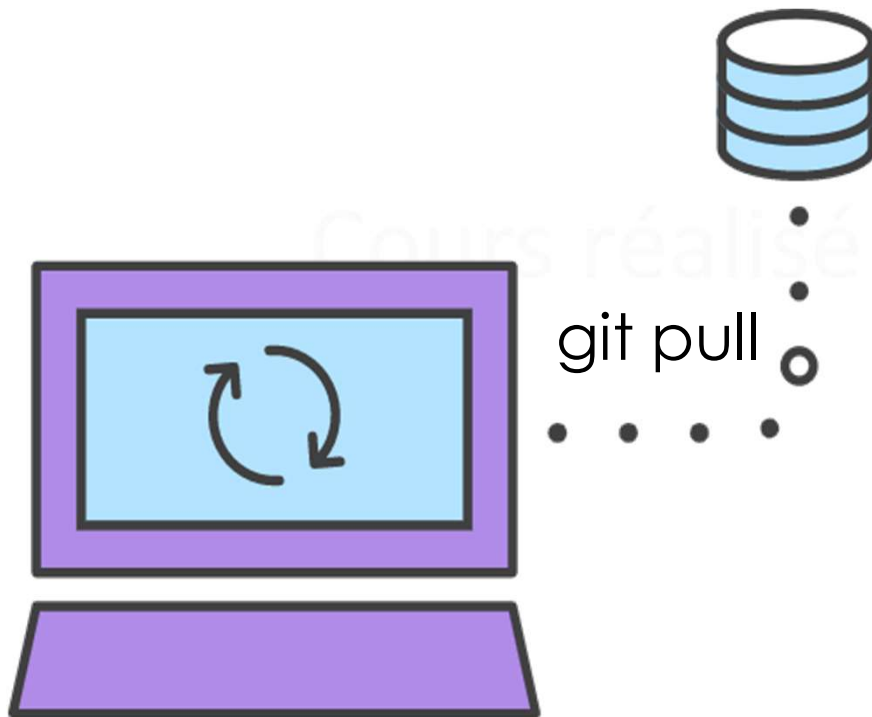
- Pour obtenir une copie locale d'un repo existant sur lequel travailler, il faut faire le clone.

```
sirin@LAPTOP-CH750RI6 MINGW64 /d/Nouveau dossier/tp3-Spring-data-jpa-crudRepo (main)
$ git clone https://github.com/sirineDevOps/SpringDataJPA-CrudRepo.git
Cloning into 'SpringDataJPA-CrudRepo'...
remote: Enumerating objects: 47, done.
remote: Counting objects: 100% (47/47), done.
remote: Compressing objects: 100% (29/29), done.
remote: Total 47 (delta 4), reused 47 (delta 4), pack-reused 0
Receiving objects: 100% (47/47), 5.90 KiB | 2.95 MiB/s, done.
Resolving deltas: 100% (4/4), done.
```

- Git clone = Git init + git remote origin <lien> + git pull origin master

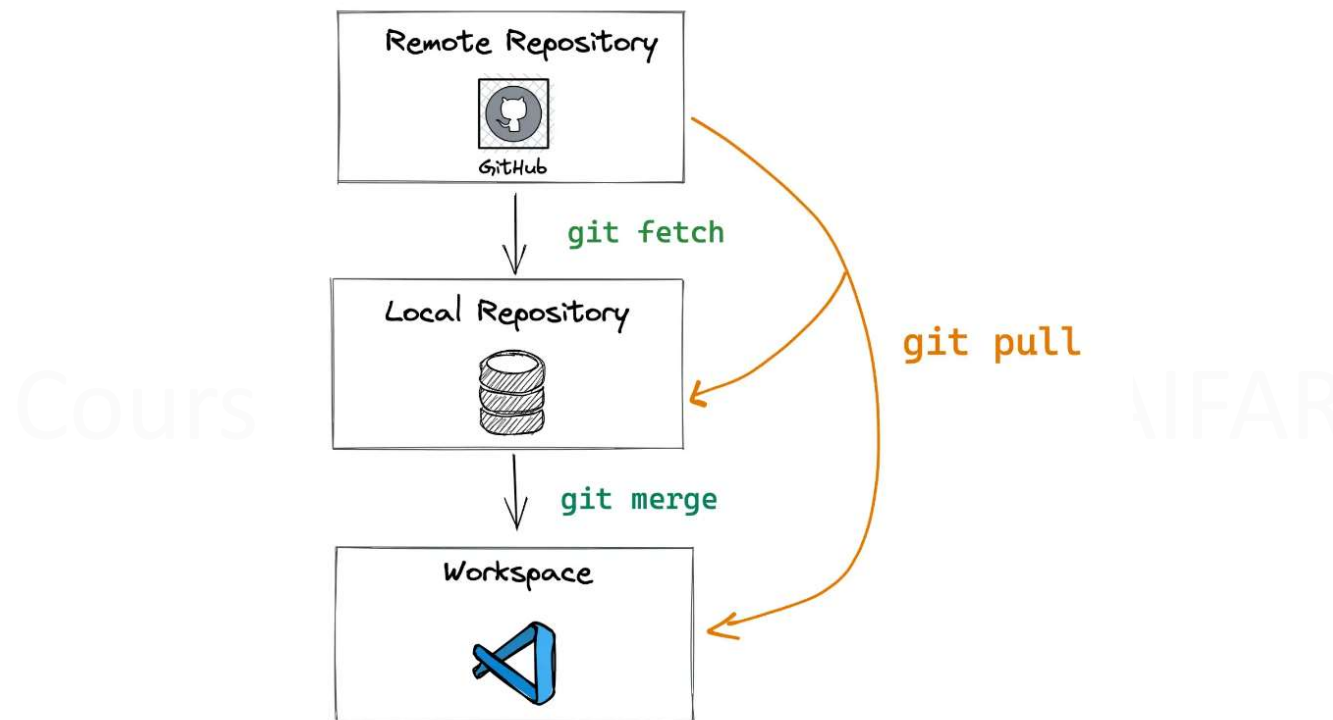
Commandes GIT – Pull

- Pour **mettre à jour cette copie** locale avec de nouveaux commits du repo central



Commandes GIT – Pull

→ `git pull` = `git fetch` + `git merge`



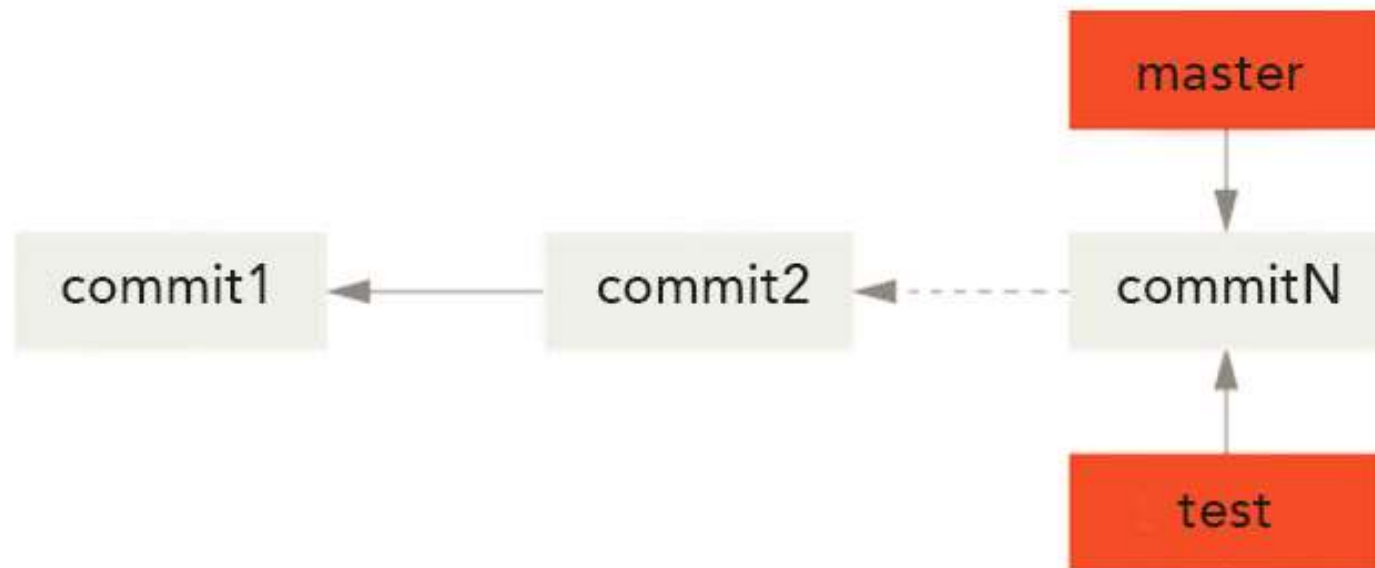
- **git fetch** : pour vérifier s'il y a des modifications dans le repository distant par rapport au contenu du repository local.
- **git merge** : Pour télécharger les mises à jour.

Commandes GIT – Clone vs Pull

- **git clone** est la façon d'obtenir une copie locale d'un dépôt existant pour travailler dessus. Il n'est généralement utilisé qu'une seule fois pour un dépôt donné, à moins que vous ne souhaitiez en avoir plusieurs copies de travail. (Ou si vous voulez obtenir une copie propre après avoir endommagé votre copie locale...)
- **git pull** est la façon de mettre à jour cette copie locale avec les nouveaux commits du dépôt distant. Si vous collaborez avec d'autres personnes, c'est une commande que vous exécuterez fréquemment.

Commandes GIT – Les branches

- Une branche dans Git est simplement un pointeur léger et déplaçable vers un de ces commits.
- La branche par défaut dans Git s'appelle main
 - Les branches permettent d'isoler certains changements pour des besoins bien spécifiques (patch, test fonction à part , etc..)



Commandes GIT – Les branches

- Pour vérifier les branches existantes , il faut taper la commande

git branch

```
sirin@LAPTOP-CH750RI6 MINGW64 /d/Nouveau dossier/tp3-Spring-data-jpa-crudRepo (main)
$ git branch
* main
```

- Pour ajouter une nouvelle branche :

git branch <nom de la nouvelle branche>

```
sirin@LAPTOP-CH750RI6 MINGW64 /d/Nouveau dossier/tp3-Spring-data-jpa-crudRepo (main)
$ git branch user-management

sirin@LAPTOP-CH750RI6 MINGW64 /d/Nouveau dossier/tp3-Spring-data-jpa-crudRepo (main)
$ git branch
* main
  user-management
```

Commandes GIT – Les branches

- Pour basculer sur la nouvelle branche:

`git checkout <nom de la branche>`

```
sirin@LAPTOP-CH750RI6 MINGW64 /d/Nouveau dossier/tp3-Spring-data-jpa-crudRepo (main)
$ git checkout user-management
Switched to branch 'user-management'

sirin@LAPTOP-CH750RI6 MINGW64 /d/Nouveau dossier/tp3-Spring-data-jpa-crudRepo (user-management)
$ git branch
  main
* user-management
```

- Nous pouvons remplacer ces deux dernières commandes par une seule commande qui crée la branche et se place sur cette branche

`git checkout -b <nom de la nouvelle branche>`

Commandes GIT – Les branches

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

```
sirin@LAPTOP-CH750RI6 MINGW64 /d/Nouveau dossier/tp3-Spring-data-jpa-crudRepo (user-management)
$ git add src/main/java/tn/esprit/spring/entity/User.java

sirin@LAPTOP-CH750RI6 MINGW64 /d/Nouveau dossier/tp3-Spring-data-jpa-crudRepo (user-management)
$ git status
On branch user-management
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   src/main/java/tn/esprit/spring/entity/User.java

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        .mvn/
        logsspringboot.log
        mvnw
        mvnw.cmd

sirin@LAPTOP-CH750RI6 MINGW64 /d/Nouveau dossier/tp3-Spring-data-jpa-crudRepo (user-management)
$ git commit -m "modification user par la deuxième branche"
[user-management 1c1a217] modification user par la deuxième branche
1 file changed, 1 insertion(+), 1 deletion(-)
```

- Quelle commande pour voir l'historique ?

Commandes GIT – Les branches

```
sirin@LAPTOP-CH750RI6 MINGW64 /d/Nouveau dossier/tp3-Spring-data-jpa-crudRepo (user-management)
$ git log
commit 1c1a2178b376aaca7560e7d922b0dfe62c574103 (HEAD -> user-management)
Author: Sirine.NAIFAR <naifar.sirine@gmail.com>
Date:   Mon Oct 4 18:51:15 2021 +0100

    modification user par la deuxième branche

commit 4934c493708fe182b44e75b052f370737db71fc5 (main)
Author: Sirine.NAIFAR <naifar.sirine@gmail.com>
Date:   Mon Oct 4 18:23:34 2021 +0100

    modification user

commit 13966ef4dfd9da09f753e4a08afe26daecc8cb
Author: Sirine.NAIFAR <naifar.sirine@gmail.com>
Date:   Mon Oct 4 18:07:02 2021 +0100

    First commit
```

- Pour supprimer une branche, il suffit de lancer cette commande:

git branch -d <nom de la branche>

```
sirin@LAPTOP-CH750RI6 MINGW64 /d/Nouveau dossier/tp3-Spring-data-jpa-crudRepo (main)
$ git branch -d user-management
Deleted branch user-management (was 1c1a217).

sirin@LAPTOP-CH750RI6 MINGW64 /d/Nouveau dossier/tp3-Spring-data-jpa-crudRepo (main)
$ git branch
* main
```


Commandes GIT – Merge

- Pour fusionner le contenu de la branche **user-management** avec le **main**, il faut basculer vers la branche **main** et faire un merge:

```
sirin@LAPTOP-CH750RI6 MINGW64 /d/Nouveau dossier/tp3-Spring-data-jpa-crudRepo (user-management)
$ git checkout main
Switched to branch 'main'

sirin@LAPTOP-CH750RI6 MINGW64 /d/Nouveau dossier/tp3-Spring-data-jpa-crudRepo (main)
$ git branch
* main
  user-management

sirin@LAPTOP-CH750RI6 MINGW64 /d/Nouveau dossier/tp3-Spring-data-jpa-crudRepo (main)
$ git merge user-management
Updating 4934c49..1c1a217
Fast-forward
 src/main/java/tn/esprit/spring/entity/User.java | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Conflit de merge

- Essentiellement, les systèmes de contrôle de version sont conçus pour gérer les contributions entre plusieurs développeurs.
- Parfois, plusieurs développeurs peuvent essayer de modifier le même contenu. Si **le développeur A** essaie de modifier le même code que **le développeur B**, un conflit peut survenir.
- Pour limiter l'apparition de conflits, les développeurs doivent travailler dans des branches isolées et indépendantes.
- L'objectif principal de la commande **git merge** est de fusionner des branches distinctes et de résoudre les conflits d'édition.

Commande avancée

- Pour annuler les commits, il suffit de lancer la commande: `git reset HEAD <Option>`

```
sirin@LAPTOP-CH750RI6 MINGW64 /d/Nouveau dossier/tp3-Spring-data-jpa-crudRepo (main)
$ git log
commit b3fbac14d7ab9f5c38628962d47eeec17466e64c (HEAD -> main)
Author: Sirine.NAIFAR <naifar.sirine@gmail.com>
Date: Mon Oct 4 20:06:01 2021 +0100

    Modif du role

commit 1c1a2178b376aaca7560e7d922b0dfe62c574103 (origin/main)
Author: Sirine.NAIFAR <naifar.sirine@gmail.com>
Date: Mon Oct 4 18:51:15 2021 +0100

    modification user par la deuxième branche

commit 4934c493708fe182b44e75b052f370737db71fc5
Author: Sirine.NAIFAR <naifar.sirine@gmail.com>
Date: Mon Oct 4 18:23:34 2021 +0100

    modification user

commit 13966ef4dfdad9da09f753e4a08afe26daecc8cb
Author: Sirine.NAIFAR <naifar.sirine@gmail.com>
Date: Mon Oct 4 18:07:02 2021 +0100

    First commit
```

```
sirin@LAPTOP-CH750RI6 MINGW64 /d/Nouveau dossier/tp3-Spring-data-jpa-crudRepo (main)
$ git reset HEAD^
Unstaged changes after reset:
M   src/main/java/tn/esprit/spring/entity/Role.java

sirin@LAPTOP-CH750RI6 MINGW64 /d/Nouveau dossier/tp3-Spring-data-jpa-crudRepo (main)
$ git log
commit 1c1a2178b376aaca7560e7d922b0dfe62c574103 (HEAD -> main, origin/main)
Author: Sirine.NAIFAR <naifar.sirine@gmail.com>
Date: Mon Oct 4 18:51:15 2021 +0100

    modification user par la deuxième branche

commit 4934c493708fe182b44e75b052f370737db71fc5
Author: Sirine.NAIFAR <naifar.sirine@gmail.com>
Date: Mon Oct 4 18:23:34 2021 +0100

    modification user

commit 13966ef4dfdad9da09f753e4a08afe26daecc8cb
Author: Sirine.NAIFAR <naifar.sirine@gmail.com>
Date: Mon Oct 4 18:07:02 2021 +0100

    First commit
```

Commande avancée

- `git reset HEAD <Option>`
 - ✓ `HEAD` : dernier commit ;
 - ✓ `HEAD^` : avant-dernier commit ;
 - ✓ `HEAD^^` : avant-avant-dernier commit ;
 - ✓ `HEAD~ 6` : avant-avant-dernier commit (notation équivalente) ;
 - ✓ `d6d98923868578a7f38dea79833b56d0326fcba1` : indique un numéro de commit ;
 - ✓ `d6d9892` : indique un numéro de commit version courte.
- ➔ SHA (Secure Hash Algorithm): un ID donnée à chaque commit.

Commande avancée

- Afficher la différence entre le contenu du dernier commit et celui du répertoire de travail.

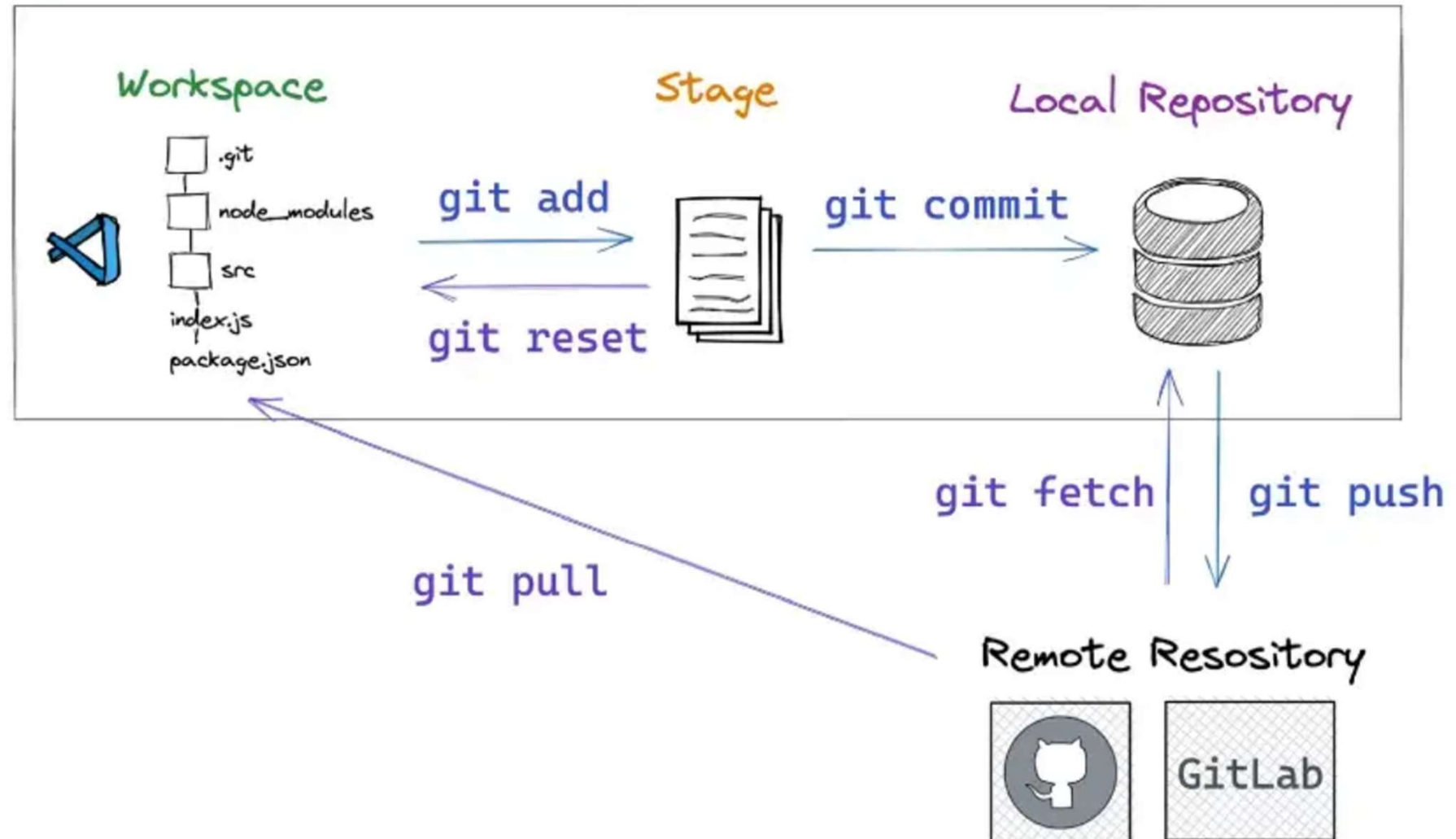
git diff HEAD

```
sirin@LAPTOP-CH750RI6 MINGW64 /d/Nouveau dossier/tp3-Spring-data-jpa-crudRepo (main)
$ git diff HEAD
diff --git a/src/main/java/tn/esprit/spring/entity/Role.java b/src/main/java/tn/esprit/spring/entity/Role.java
index b47c2c7..4a3ae8e 100644
--- a/src/main/java/tn/esprit/spring/entity/Role.java
+++ b/src/main/java/tn/esprit/spring/entity/Role.java
@@ -1,5 +1,5 @@
 package tn.esprit.spring.entity;

 public enum Role {
-    CHEF_DEPARTEMENT, ADMINISTRATEUR, INGENIEUR, TECHNICIEN, STUDENT
+    CHEF_DEPARTEMENT, ADMINISTRATEUR, INGENIEUR, TECHNICIEN, STUDENT, Simple
 }
```

Résumé

Local



Résumé

