

# Gérer son code avec Git

Git



# Objectifs

- Comprendre l'intérêt des gestionnaires de code
- Effectuer des manipulations simples
- Découvrir une stratégie de gestion des projets



Git

## Qu'est-ce qu'un logiciel de gestion de versions ?

- Un système de gestion de version, appelé VCS en anglais : Version Control System
- Permet de suivre les modifications d'un ensemble de fichiers.
- Permet de
  - revenir à une version antérieure d'un fichier,
  - de pouvoir comparer des versions différentes d'un fichier,
  - ou de suivre l'historique des changements.
- Les évolutions de cet ensemble de fichiers sont matérialisées par des versions.



Un système de gestion de version, appelé VCS en anglais : Version Control System, permet de suivre les modifications d'un ensemble de fichiers.

Cela permet de revenir à une version antérieure d'un fichier, de pouvoir comparer des versions différentes d'un fichier, ou de suivre l'historique des changements.

Les évolutions de cet ensemble de fichiers sont matérialisées par des versions. Ces versions sont enregistrées dans un repository, que l'on peut voir comme une sorte de base de données de toutes les versions du projet.

Git

# Git : logiciel de gestion de versions

- Il existe plusieurs outils de versions :
  - Git
  - SVN
  - Mercurial
  - CVS, ...
- Git est un logiciel de gestion de versions distribué
  - Très puissant et récent
  - Créé par Linus Torvalds



Git est un système de gestion de version, comme SVN, Mercurial, ou CVS.

GIT est un logiciel de gestion de versions distribué

Très puissant et récent, il a été créé par Linus Torvalds, qui est entre autres l'homme à l'origine de Linux. Il se distingue par sa rapidité et sa gestion des branches qui permettent de développer en parallèle de nouvelles fonctionnalités.

Git

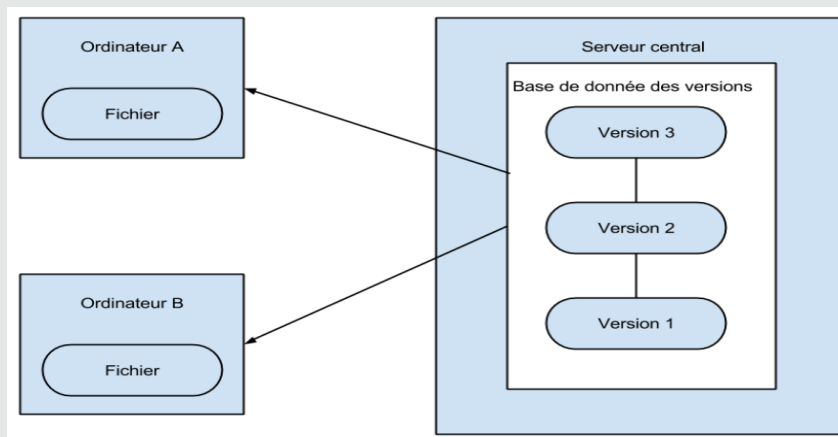
## Travail en équipe

- Un VCS permet de partager un projet entre différents contributeurs.
- Chaque contributeur peut
  - Ajouter, modifier ou supprimer des fichiers au projet
  - Créer des versions
  - Partager ces versions



Git

## Gestion de version centralisé

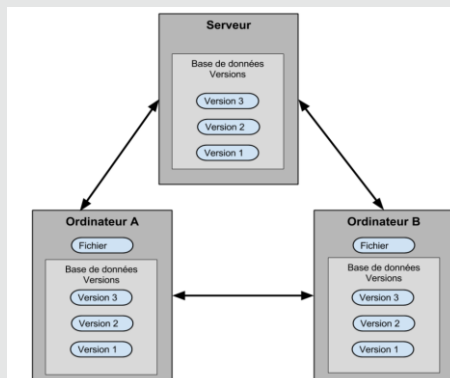


Dans une gestion de version centralisée, les versions sont stockées sur un seul repository disponible sur un serveur central. Les utilisateurs récupèrent leurs fichiers de travail, puis remettent le fichier sur le serveur après modification.

Le risque de cette architecture est la coupure d'accès au serveur central. Dans ce cas, plus aucun utilisateur ne peut travailler. Pire, si le serveur tombe définitivement, toutes les versions des projets sont perdues avec lui.

Git

## Gestion de version décentralisé



- Git fait partie des systèmes de gestion de version décentralisée.



Dans une gestion de version décentralisée, chaque utilisateur possède une copie du repository. Même en cas de coupure d'accès au serveur, chacun peut continuer à travailler, et gérer des versions en local.

Git fait partie des systèmes de gestion de version décentralisée.

### La gestion des versions par snapshots

Contrairement à d'autres systèmes qui enregistrent les différences, appelés patches, entre 2 versions d'un même fichier, Git enregistre pour chaque version le fichier modifié en entier. Pour les fichiers non modifiés, seul un lien vers la dernière version du fichier est stocké.

Git

## Les avantages de Git

- Git est flexible, il permet :
  - Le travail déconnecté
  - De travailler facilement sur plusieurs versions (branches)
  - Travailler à plusieurs
    - Avec un serveur central
    - Avec plusieurs serveurs
    - Directement entre plusieurs développeurs





Git

## Commandes de configuration de Git

- `git config --global user.name "name"`
- `git config --global user.email "email"`



configurer ses nom et prénom

```
git config --global user.name "<prénom> <nom>"
```

configurer son adresse mail

```
git config --global user.email "<adresse@campus-eni.fr>"
```

# Git

## Théorie de Git

- 4 espaces, 4 ambiances



Cette notion est primordiale pour la compréhension de la suite.

Git gère trois états dans lesquels les fichiers peuvent résider : validé, modifié et indexé.

- Validé signifie que les données sont stockées en sécurité dans votre base de données locale.
- Modifié signifie que vous avez modifié le fichier mais qu'il n'a pas encore été validé en base.
- Indexé signifie que vous avez marqué un fichier modifié dans sa version actuelle pour qu'il fasse partie du prochain instantané du projet.

Ceci nous mène aux trois sections principales d'un projet Git pour le dev :

- le répertoire de travail
- la zone d'index
- le dépôt Git

Git

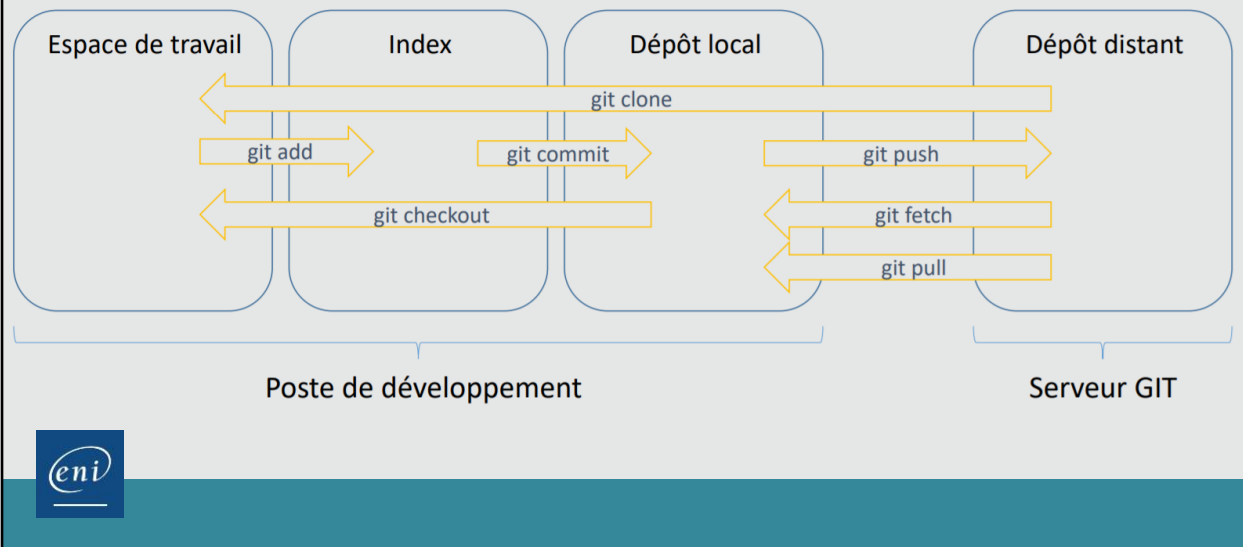
## Commandes d'initialisation de projet Git

- Créer un dépôt local
  - `cd path`
  - `git init`
- Cloner un dépôt existant
  - `git clone URL_DEPOT`



Git

## Commandes principales de Git



<https://training.github.com/downloads/fr/github-git-cheat-sheet.pdf>

`git status` → renseigne sur le status du dépôt actuel

Untracked = fichier non référencé

`git add .` → le point permet d'indiquer l'ajout de tous les fichiers non référencés

`git add toto.txt` → dans ce cas seulement toto.txt sera ajouté au référentiel Git

`git commit -m "initial commit"` → permet de sauvegarder la version du fichier en local. Le commentaire doit être court et explicite

### Annuler le dernier commit et modifs

`git reset` → tous les fichiers qui ont été indexés sont annulés

`git reset --hard` → permet de remettre même le contenu des fichiers à l'état précédent

Envoyer ses commits vers le dépôt distant

`git push` → Met à jour le serveur en appliquant les commits sur toutes les branches \*COMMUNES\* au DÉPÔT\_LOCAL et au serveur.

Les branches locales qui n'ont jamais été poussées sur le serveur ne

sont pas partagées.

Mettre à jour le dépôt local :

git fetch → Récupère tout l'historique du dépôt nommé

git pull → Récupère les modifications associées à la référence du DÉPÔT\_DISTANT et les fusionne dans la branche courante.

Les 2 commandes suivantes sont surtout pour gérer les branches :

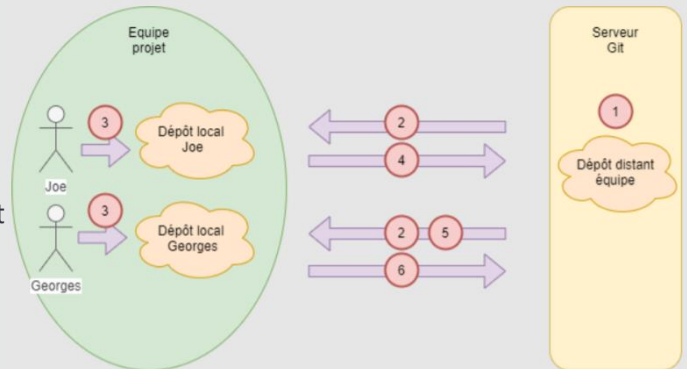
git checkout → Met à jour les FICHIER(S) ou DOSSIER(S) dans l'ESPACE\_DE\_TRAVAIL en écrasant toutes les modifications locales. Ne PAS changer de branches.

git merge → Fusionne les modifications du COMMIT ou de la BRANCHE dans la branche courante.

Utilisez --no-commit pour ignorer les modifications n'ayant pas encore fait l'objet d'un commit.

## Exemple d'utilisation de Git

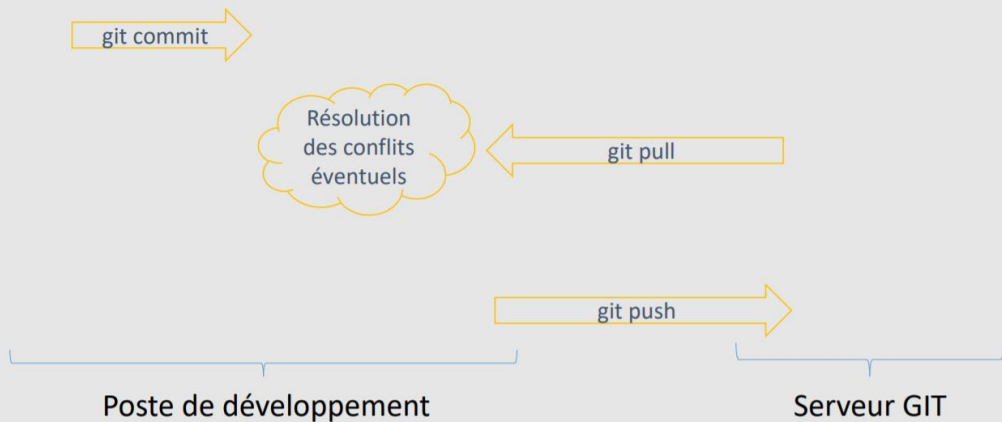
1. Création du dépôt Git sur le serveur
2. Clonage du dépôt en local
3. Gestion des fichiers en local pour chaque développeur
4. Envoi de Joe vers le serveur Git
5. Récupération par Georges des versions de Joe.
6. Georges envoie les fichiers fusionnés sur le serveur
7. ....



1. Création du dépôt Git sur le serveur
2. Initialisation des dépôts Git locaux par clonage
3. Ajout, modification et suppression de fichiers au projet par chaque membre de l'équipe. Versionnement des fichiers sur le dépôt local.
4. Envoi des versions locales de Joe sur le serveur
5. Récupération par Georges des versions de Joe.
  - Les fichiers distants et locaux sont fusionnés
  - Risque de conflit si les mêmes fichiers ont été modifiés par Joe et Georges
6. Envoi des fichiers fusionnés contenant les modifications faites par Georges sur le serveur ...

Git

## Git – commit – pull – push



Un bon commit est un commit :

- Qui ne concerne qu'une seule fonctionnalité du programme
- Qui est le plus élémentaire possible tout en restant cohérent
- Qui compile, seul idéalement
- Qui est documenté

Un commit c'est :

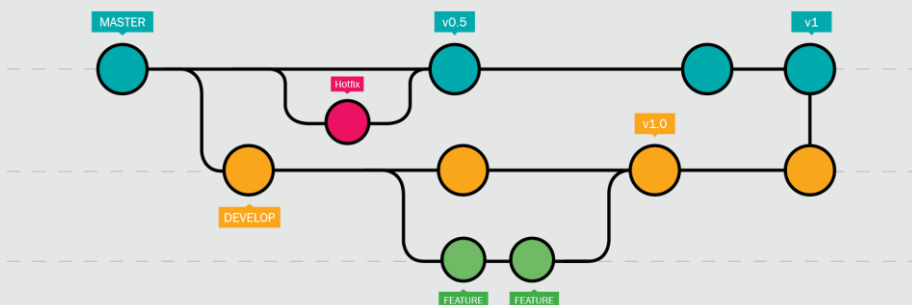
- Une validation d'ajouts, de modifications et de suppressions de lignes de code
- Des méta-données qui le documente (commentaire, auteur, date, ...)

Quelques règles :

- Commiter le plus souvent possible.
  - Au moins une fois par jour
  - Veuillez vérifier que le code compile correctement.
- Commiter uniquement le code source.
- Utiliser le fichier .gitignore pour vous aider
- Ecrire un commentaire clair et explicite

## Git Branches et Git

- Les branches sont des "pointeurs" vers des commits.



### Bien organiser ses branches = Workflow

- Master contient le code qui est actuellement déployé, c'est-à-dire, le code qui est en production et que les utilisateurs de l'application utilisent.
- Pour intégrer les changements (nouvelles fonctionnalités, corrections, etc.) = branche Develop.
- *Features*: ajout de nouvelles fonctionnalités

Une fois que Develop atteint un certain niveau d'avancement, et que celle-ci est testée et stable

- merge sur master.
- Ensuite une mise en production.



Git

## Commandes sur les branches de Git

- Créer une branche
  - `git branch [branch-name]`
- Changer de branche
  - `git checkout [branch-name]`
- Fusionner une branche secondaire dans la branche courante :
  - `git merge [branch]`
- Effacer la branche secondaire :
  - `git branch -d [branch]`



Il est possible de créer et de se placer sur une sous-branche en une passe :  
`git checkout v2.0 -b v2.1`

## Branches de Git – Conflit lors de fusion

- Il y a plusieurs stratégie de fusion automatique (fast-forward, 3 way, etc...)
  - `git merge v2.0`  
Auto-merging index.html  
CONFLICT (content): Merge conflict in index.html  
Automatic merge failed; fix conflicts and then commit the result.
- Si ça ne fonctionne pas, il faut le faire à la main.
  - Comparer et valider
  - Finaliser le commit
  - `git add index.html`
  - `git commit`



La notion de *fast-forward*.

- Lors d'un merge, si la branche est dans la suite logique de la branche Master, dans ce cas Git réalise un Fast-Forward.
- Il déplace tout simplement le pointeur vers la branche de merge

La notion de *three-way merge*

Le comportement semble légèrement différent de celui observé pour la fusion précédente de la branche correctif. Dans ce cas, à un certain moment, l'historique de développement a divergé. Comme le *commit* sur la branche sur laquelle vous trouvez n'est plus un ancêtre direct de la branche que vous cherchez à fusionner, Git doit effectuer quelques actions. Dans ce cas, Git réalise une simple fusion à trois sources (*three-way merge*), en utilisant les deux instantanés pointés par les sommets des branches ainsi que leur plus proche ancêtre commun.

## Git GitHub

- GitHub est un dépôt public Git, accessible sur Internet.
- Tout le monde peut créer son projet gratuitement, si le projet est open source et accessible aux autres.
  - Sur GitHub, on peut trouver du code utilisé par Facebook, Red Hat, et d'autres entreprises bien connues.
- Néanmoins, tous les projets GitHub ne sont pas nécessairement open source ni même publics.
  - GitHub propose alors une tarification en fonction du nombre de projets et d'utilisateurs.



**GitHub** est un dépôt public Git, accessible sur Internet. Tout le monde peut créer son projet **gratuitement**, pourvu qu'il soit open source et accessible aux autres. Sur GitHub, on peut trouver du code utilisé par Facebook, Red Hat, et d'autres entreprises bien connues. Néanmoins, tous les projets GitHub ne sont pas nécessairement open source ni même publics. Certains projets sont privés ou accessibles à certaines personnes seulement. GitHub propose alors une tarification en fonction du nombre de projets et d'utilisateurs.

Comprenez tout de suite la différence entre **Git** et **GitHub** :

**Git** est un logiciel pour créer des **versions de code**. Il peut être utilisé directement sur votre ordinateur pour stocker l'historique du code. Mais tout l'intérêt réside dans le partage du code entre développeurs. Il faut donc disposer d'une ou plusieurs machines qui vont héberger de l'espace compatible avec Git, un serveur compatible avec Git ;

**GitHub** est le plus grand **serveur compatible Git**, accessible de manière publique sur Internet. Les entreprises disposent aussi souvent d'un serveur privé, hébergé parfois dans l'entreprise, pour partager le code entre développeurs.

Git

## Créez un compte GitHub

### Chaque stagiaire doit :

- Accéder au site <https://github.com/>
- Suivez le processus de création de compte
  - En indiquant un nom d'utilisateur,
  - votre adresse e-mail
  - et le mot de passe que vous souhaitez utiliser.
  - Vous avez le choix entre un compte gratuit avec projets publics et privés mais limités en données, ou un compte payant sans limite.
- Dans ce cours, vous pouvez créer un compte gratuit



Git

## Créez un jeton d'accès

Sous l'IDE Eclipse, vous devez utiliser le jeton d'accès pour sécuriser le transfert de code entre IDE et GitHub.

### **Chaque stagiaire doit créer son jeton d'accès :**

- Suivez les recommandations officielles :  
<https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token>
  - Choisissez une durée d'au moins 1 mois pour le jeton.



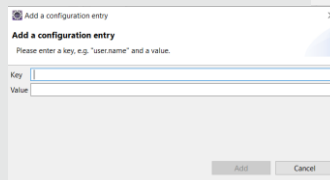
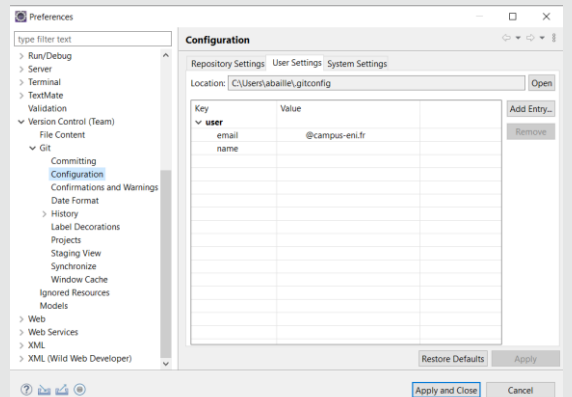
Pour accéder au compte GitHub en développement, il est possible d'utiliser soit son mot de passe soit un token.

Eclipse privilégie l'utilisation du Token.

# Git Configuration IDE/GitHub

**Sous l'IDE Eclipse, chaque stagiaire doit créer son jeton d'accès :**

- Dans le menu, cliquez sur Window > Preferences
- Sélectionnez Version Control > Git > Configuration
- Ajoutez 2 clefs :
  - user.email
  - user.name



Dans Eclipse vous configurer votre utilisateur GitHub  
Vous devez configurer votre email et name de GitHub

Toute l'équipe doit avoir la même version :

- de Java,
- d'Eclipse
- Tomcat

Git

# Responsable GitHub – Etape 1 – Repository

## Création « Repository » sur GitHub



- Il faut choisir une personne responsable du projet
- C'est sur son compte à lui qu'il faut créer un « New repository »



Create a new repository


A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)


Owner <sup>\*</sup> / Repository name <sup>\*</sup>

 / TestProjetEnchere 

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

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.](#)

[.gitignore template: java](#)

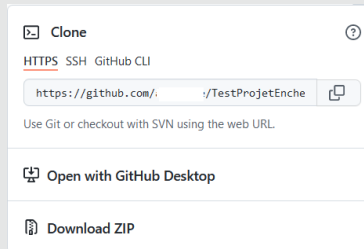
Mode privé

Et en cochant gitignore - Java

Git

## Responsable GitHub – Etape 2 – Clone

- Clone en local le « repository » sur sa machine en copiant l'adresse dans « Code »



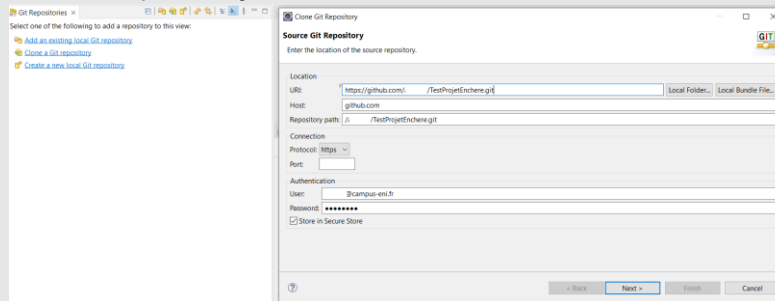


Git

## Responsable GitHub – Etape 3 – IDE

### Sous Eclipse :

- Ouvrir la perspective Git
- « Clone a Git repository »

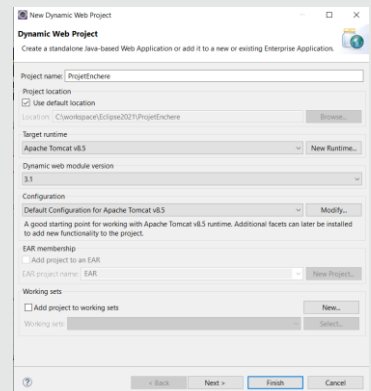
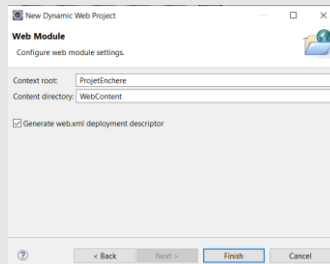
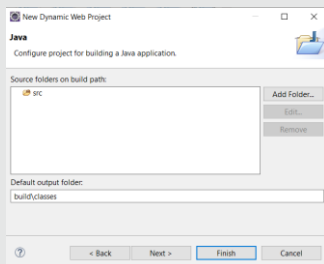


Par défaut, il devrait sélectionner le jeton de développement que vous avez créé.  
Si ce n'est pas le cas, il vous mettra une popup pour le saisir.  
Vous pouvez faire un copier/coller

Git

## Responsable GitHub – Etape 4 – Projet

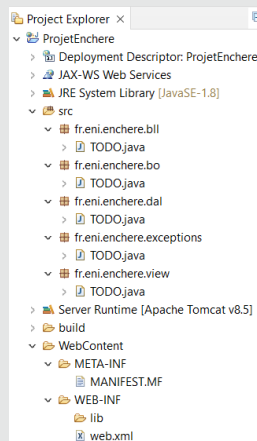
- Création d'un Dynamic Web Project en Java
  - indexé le nom du projet par vos initiales.
- Attention à la configuration du projet :
  - Vérifiez le folder de source et le répertoire pour le Web :



Git

## Responsable GitHub – Etape 5 – Packages

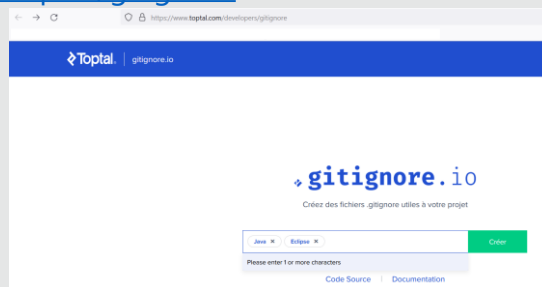
- Création des packages par défaut avec une classe par défaut :



Git

## Responsable GitHub – Etape 6 – .gitignore

- Pour éviter les contraintes de configuration dans l'équipe, il faut ajouter un fichier .gitignore à la racine du projet
- Pour le configurer, utilisez <https://www.toptal.com/developers/gitignore>
  - Et précisez Java et Eclipse
  - Décommenter .project
  - Ajouter dans le script .classpath



Attention, dans le script généré, il y a certaines lignes à décommenter :

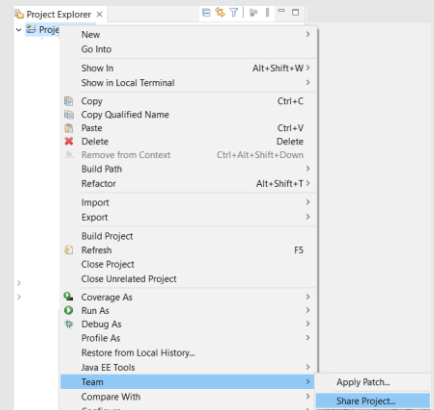
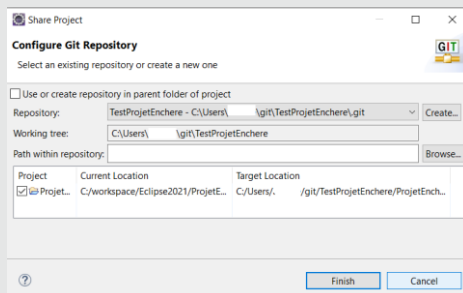
#.project

Et ajouter .classpath

Git

## Responsable GitHub – Etape 7 – Projet/Git

- Associer le projet au Git
- Clic droit sur le projet > Team > Share Project
- Sélectionner votre « repository »

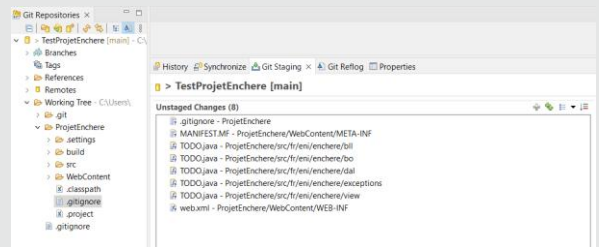


Git

## Responsable GitHub – Etape 8 – Commit/Push

Dans la perspective Git :

- Sélectionner tous les nouveaux fichiers et répertoires
- Cliquer sur +
- Mettre un message de commit par défaut
- Et cliquer « Commit and Push »
- Une popup de validation s'affichera

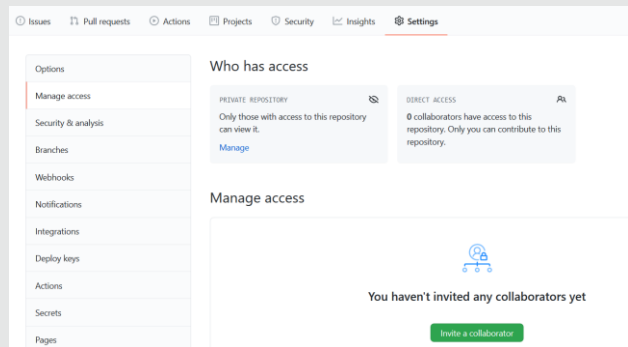


Sélectionner le projet sous Working Tree  
Vérifiez dans GitHub que tout c'est bien passé.

Git

## Responsable GitHub – Etape 8 – Invitation

1. Sous GitHub, Inviter les autres
2. Les autres doivent accepter l'invitation. Attention : il faut être connecté à GitHub avant d'accepter l'invitation.

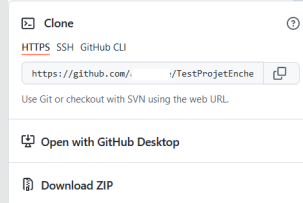


La suite est pour les coéquipiers

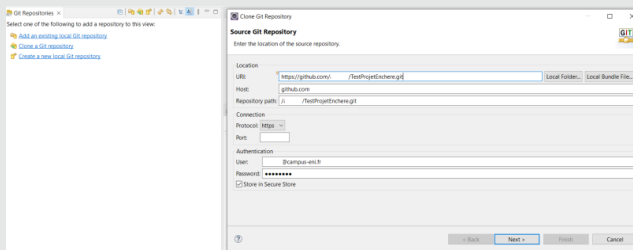
Git

# Coéquipier(s) – Etape 1 – Clone du projet

1. Copier le lien depuis GitHub



2. Dans Eclipse, faites le clone du « repository GitHub »



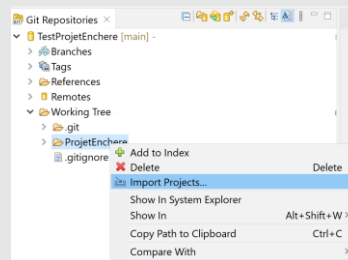
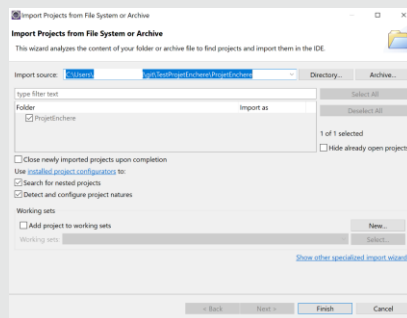
La suite est pour les coéquipiers



Git

## Coéquipier(s) – Etape 2 – Import du projet

- Dans la perspective Git :
  - Clic droit sur le projet > Import Projects...



Git

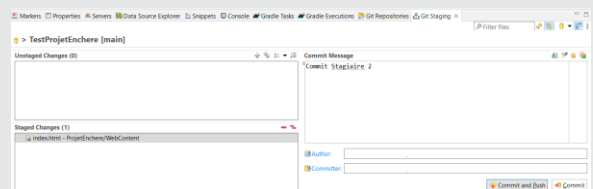
## Coéquipier(s) – Etape 3 – Commit/Push

- Ajouter une page HTML pour tester le « Commit and Push » de chaque coéquipier.
- L'indexer par le prénom

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8">
5 <title>Enchère</title>
6 </head>
7 <body>
8 <h1>Stagiaire 2</h1>
9 </body>
10 </html>
```

Dans la perspective Git :

- Sélectionner le nouveau fichier
- Cliquer sur +
- Mettre un message de commit par défaut
- Et cliquer « Commit and Push »



Git

## Commit / Push / Pull

- Chaque développeur doit faire « commit » en local régulièrement pour ne pas perdre son code et ses modifications.
- Il faut faire des « push » quand une nouvelle fonctionnalité est terminée et validée.
  - Il faut une communication active dans l'équipe, dès qu'un « push » est fait, il faut que les autres fassent un « pull » pour récupérer les modifications.
- Parfois, lors d'un « push » ou d'un « pull » des conflits peuvent apparaître.



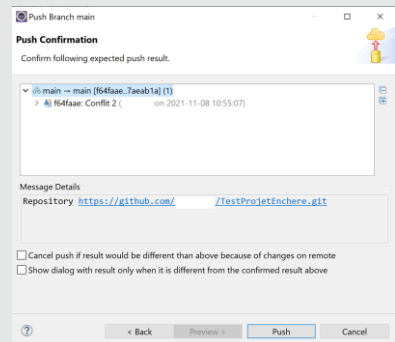
Pensez à faire le pull avant le push

# Git

## Détection des conflits

- Une popup vous indique qu'il y a un conflit
- Vous pouvez regarder le détail :

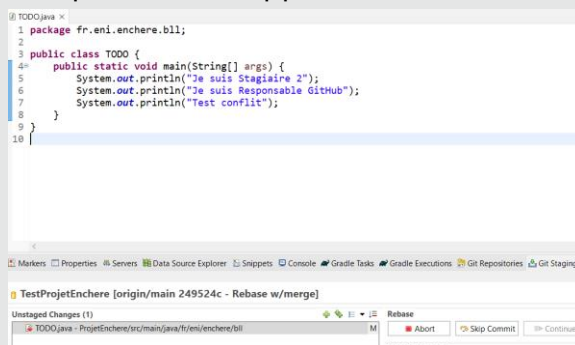
```
1 package fr.eni.enchere.bll;  
2  
3 public class TODO {  
4     public static void main(String[] args) {  
5         <<<<<< Upstream, based on branch 'main' of https://github.com/ /TestProjetEnchere.git  
6         System.out.println("Je suis Stagiaire 2");  
7         =====  
8         System.out.println("Je suis Responsable GitHub");  
9         >>>>>> 8ac8944 Add Main  
10        System.out.println("Test conflit");  
11    }  
12 }  
13 }
```



## Git

# Gestion des conflits

- Les conflits sont gérés manuellement par les développeurs :
  - Il faut déterminer le code à conserver
  - Faire les modifications
- Faire un « commit »



The screenshot shows an IDE window titled 'TODO.java' with the following code:

```
1 package fr.eni.enchere.bll;
2
3 public class TODO {
4     public static void main(String[] args) {
5         System.out.println("Je suis Stagiaire 2");
6         System.out.println("Je suis Responsable GitHub");
7         System.out.println("Test conflit");
8     }
9 }
10
```

Below the code editor, a 'Rebase' dialog box is open for 'TestProjetEnchere [origin/main 249524c - Rebase w/merge]'. It shows 'Unstaged Changes (1)' with the file 'TODO.java - ProjetEnchere/src/main/java/fr/eni/enchere/bll'. The dialog has buttons for 'Rebase', 'Abort', 'Skip Commit', and 'Continue'.



Les conflits sont gérés manuellement par les développeurs :

Il faut déterminer le code à conserver

Faire les modifications

Faire un « commit »

Si la fonctionnalité est terminée :

Push

Prévenir le reste de l'équipe

Sinon continuez votre travail