CHAPITRE 4C - MÉTHODOLGIE ET GESTION DE VERSIONS

Contrôle de versions - GIT

Table des matières

1 Système de contrôle de versions

1.1 Pourquoi un système de contrôle de versions ?

Lorsqu'on travaille en équipe sur un projet de longue haleine, qui comporte plusieurs fichiers de code, il est important de pouvoir :

- Accéder aux versions les plus récentes des différents fichiers.
- Synchroniser les changements lorsque plusieurs personnes travaillent en parallèle sur un même fichier.
- Conserver l'historique des changements.
- Être capable de facilement récupérer l'ensemble des fichiers, pour pouvoir les compiler, exécuter et tester.

Il est toujours possible de conserver à un endroit centralisé, les dernières versions, et de communiquer par courriel pour "bloquer" l'accès à un fichier pendant que quelqu'un le modifie. Cette façon de faire devient vite impraticable pour des équipes de plus de 2 ou 3 personnes.

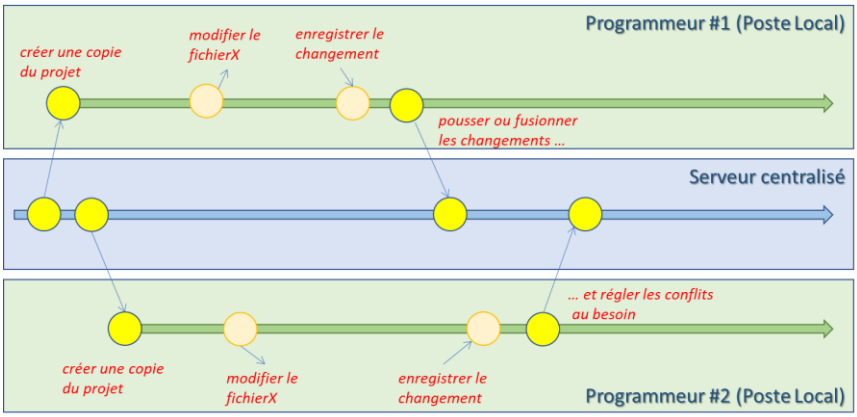
Il existe des outils informatiques qui permettent d'effectuer ces tâches de façon efficace et sécuritaire que l'on nomme système de contrôle de versions

1.2 Systèmes contrôle de versions (SCV)

1.2.1 Systèmes centralisés (comme CVS)

L'idée générale des **systèmes** contrôle de versions **centralisés** (***serveur-client***) est de :

- Conserver les **fichiers communs** officiels sur un **endroit centralisé** (serveur).
- Permettre aux programmeurs (client) de se **créer une copie locale de ces fichiers** pour y travailler.
- **Fusionner les copies** modifiées **de l'utilisateur avec les copies centralisées**, tout en gérant les modifications contradictoires ayant été insérées par d'autres programmeurs.

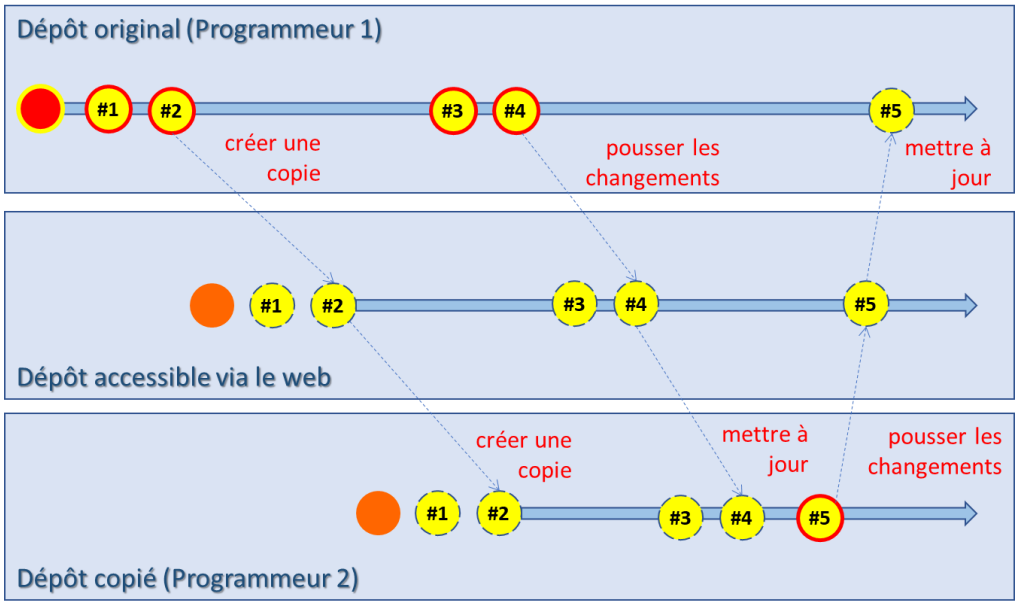


1.2.2 Systèmes distribués (comme GIT)

Contrairement aux systèmes centralisés (approche client-serveur), avec les **systèmes distribués** (approche **peer-to-peer**), chaque programmeur conserve l'intégralité du projet sur son poste. Seul l'historique des changements doit être transmise et synchronisée entre les différents postes des programmeurs.

Ainsi, si un poste de programmeur (***dépôt***) contient le code original avec quelques changements (***commit***), il est possible pour un autre programmeur de copier le tout sur son poste (code + changements) et de commencer à contribuer au projet en appliquant ses propres changements. Il s'agira par la suite de synchroniser les deux dépôts périodiquement, pour que tout le monde ait l'ensemble de l'historique des changements apportés au code original.

En pratique, puisque les communications directes entre les postes de plusieurs programmeurs n'est pas toujours possible, il est commun d'avoir recours à un endroit commun d'échange, souvent via le web, pour faciliter la synchronisation entre les différents dépôts. Mais contrairement, aux systèmes centralisés, cet espace commun n'est qu'un autre dépôt régulier, au même titre que celui d'un programmeur régulier. La seule différence est que ce dépôt est accessible via le web.



GIT est un système de contrôle de versions qui suit cette philosophie et GIT-Lab représente un dépôt accessible via le web.

1.3 Systèmes de contrôle de versions disponible sur le marché

On trouve plusieurs systèmes de contrôle de versions sur le marché. Parmi les plus populaires, on trouve :

- Git (GitLab, GitHub, Bitbucket, …)
- Visual Studio Team System (Microsoft Team Foundation Server)
- Perforce
- Tortoise-CVS (un des plus anciens …)
- …

Puisqu'une grande partie de sa fonctionnalité est gratuite, que nous avons un service d'hébergement web GIT-Lab complet avec licence académique, et qu'il est devenu le plus populaire dans les dernières années, c'est **GIT** avec le service d'hébergement web **GIT-Lab** que nous allons utiliser pour la majorité des cours de votre formation.

1.4 GIT vs GIT-Lab vs GIT-Hub vs …

1.4.1 GIT

GIT est un **logiciel de contrôle de versions** en source ouverte, donc gratuit à utiliser, qui s'installe facilement sur le poste local des programmeurs. C'est le logiciel qui permet d'effectuer toutes les commandes à partir du poste du travailleur.

1.4.2 GIT-Lab, GIT-Hub, Bitbucket

GIT-Lab , **GIT-Hub** et **Bitbucket** sont des **hôtes web** qui permettent de créer une zone de communication commune, à travers une interface web, pour que les programmeurs puissent utiliser GIT et synchroniser leur dépôt respectif. De plus, en utilisant ce service web, on peut configurer les paramètres d'un projet : ajouter/supprimer des membres de l'équipe, régler les paramètres d'accessibilités, ….

Les conditions d'utilisation de ces programmes varient :

Service d'hébergement web	Projets publics	Projets Privés
GIT-Hub	Gratuit	Payant
Bitbucket	Gratuit	Gratuit (≤5 collaborateurs)
GIT-LAB	Gratuit	Gratuit

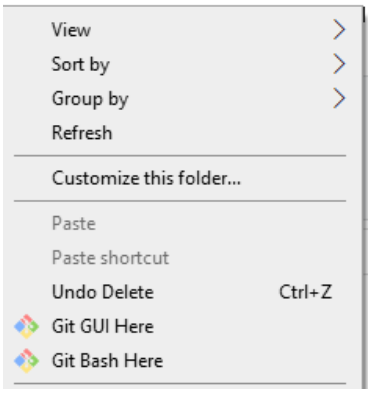
2 GIT

2.1 Installation de GIT

Pour installer GIT sur votre poste vous devez d'abord le télécharger à partir du site officiel (https://git-scm.com/download/win) et l'installer en suivant les instructions*.

* Attention, à l'étape 4 (après 3 Next), vous pouvez spécifier l'éditeur de texte à utiliser (ex : Notepad++) pour les différents outils de GIT …

2.2 Configuration de GIT



Pour accéder rapidement à l'invité de commande de GIT à un endroit précis sur votre poste, il suffit de se déplacer dans ce dossier et cliquez sur le bouton de droite de la souris et vous devriez obtenir le menu contextuel suivant ci-contre. Il suffit alors de cliquer sur l'option **Git Bash Here** et l'invité de commandes s'ouvrira.

Une fois l'invité ouvert, tapez les commandes suivantes pour permettre aux autres utilisateurs de pouvoir vous identifier sur les projets communs que vous aurez :

```
git config --global user.name Votre Nom Complet
git config --global user.email votreCourriel@dti.crosemont.quebec
```

Note :

Pour être capable de vous connecter avec Gitlab, il se peut que vous ayez à modifier la configuration de votre ordinateur et/ou de git.

Si les opérations de clonage (section 4.2) ou de création d'un nouveau projet sur GitLab à partir de l'invité de commandes (section 4.1) ne fonctionnent pas, essayez **une de ces 2 solutions** :

1. Dans l'invité de commande windows ou dans git bash, que vous ouvrez en tant qu'administrateur, tapez :
   ```
   git config --system --unset credential.helper
   ```
2. … ou si ça ne fonctionne pas, dans l'invité de commande Git Bash, tapez :
   ```
   export https_proxy=10.1.0.5:8080
   ```

2.3 Nomenclature et commandes principales

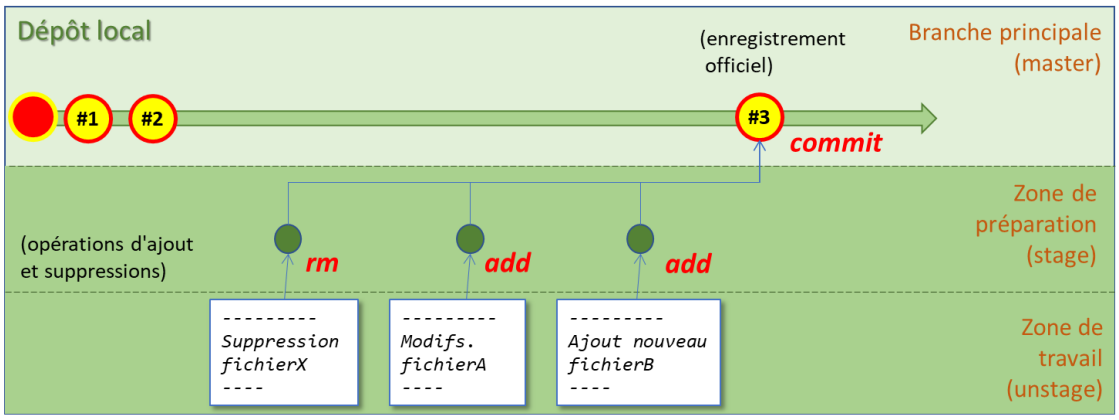
2.3.1 Nomenclature GIT

- **Projet** : le code, les changements, les informations des collaborateurs et les propriétés d'accès (privée ou publique), …

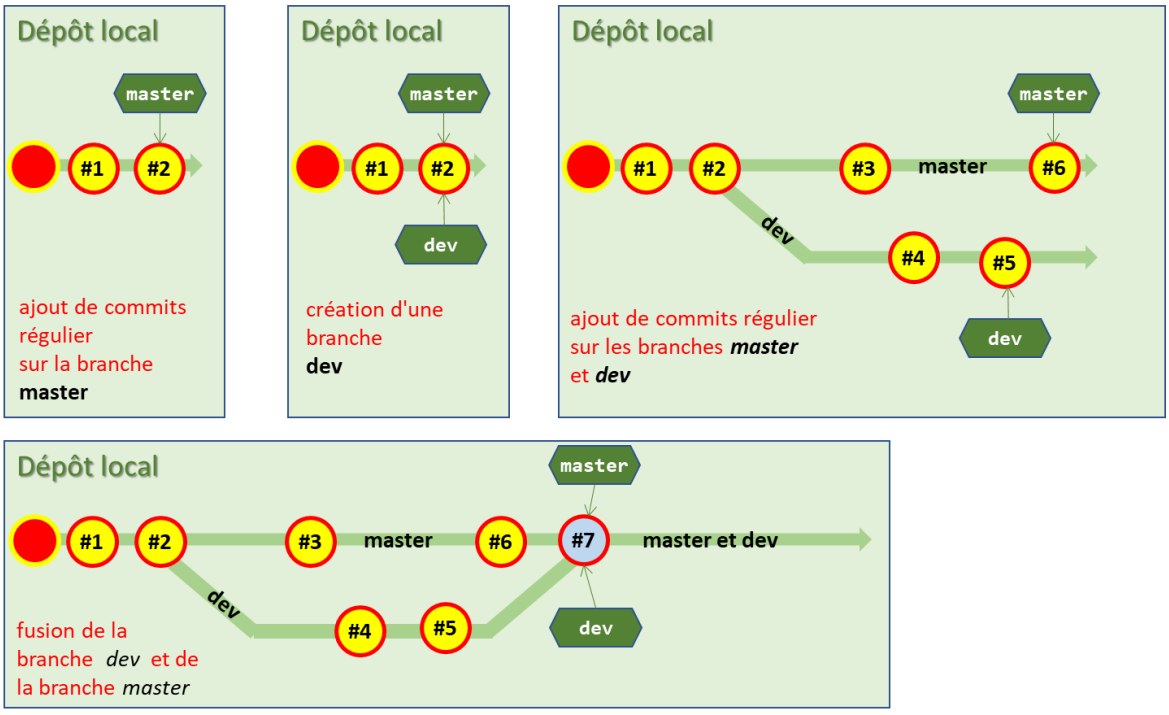
- **Dépôt** (***repository***) : endroit physique où sont placés les fichiers et l'historique des *changements*.
- **Dépôt local** : le dépôt sur lequel on travaille (typiquement votre poste).
- **Dépôt distant** (***remote***) : le dépôt externe avec lequel on veut communiquer (typique le dépôt web GIT-Lab)

- **Changement officiel** (***commit***) : changements enregistrés officiellement. Chaque **commit** est identifié par un numéro (long numéro en hexadécimal). Un commit peut être vu comme un nœud dans une structure d'arborescence de séquences de commits.

- **Zone de préparation** (***stage***) et **zone de travail** (***unstage***): lorsqu'on édite un fichier avec un éditeur quelconque dans un environnement GIT, la modification non-officielle apparait seulement dans la ***zone de travail***, donc directement dans les fichiers sur votre poste. Pour signifier à GIT qu'un changement fera partis du prochain commit, on doit l'envoyer dans la ***zone de préparation*** (commande ***add***).



- **Branche** (***branch***):
  - Les branches permettent de créer des séquences ou chemins indépendants de commits.
  - La branche est en réalité une étiquette qui pointe vers le dernier commit d'une séquence donnée.
  - Il est à noter que la branche initiale (ou principale) se nomme ***master*** et que seul les personnes ayant **un rôle de responsable** (« ***maintainer*** » dans GIT) peuvent y faire des modifications directement
  - Il est possible de fusionner 2 branches pour qu'elles se reconnectent.



Note :

Il y a deux types de commits (nœud) :

- Commit régulier (exemple page précédente : nœuds #1 à #6)
  - Il s'agit d'un commit contenant tous les changements depuis le dernier commit de la branche courante. Il y a donc une relation 1 parent à 1 enfant.
- Commit fusionné (exemple page précédente : nœud #7):
  - Il est possible de fusionner les commits de deux branches sur une de celle-ci, alors le commit résultant a 2 parents, soit le dernier commit des deux branches.

2.3.2 Commandes principales

Pour utiliser les commandes il suffit de taper ***git***, suivit du nom de la commande et des étapes. Voici un résumé des commandes principales :

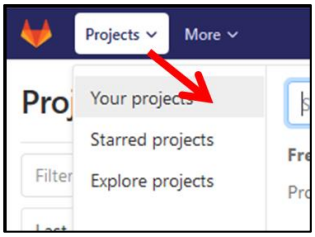
Commande	Description
Création de dépôts	
Init	Créer un dépôt local
clone *cheminDépôtDistant*	Obtenir une copie locale, d'un dépôt distant existant, pour créer un dépôt local
push --set-upstream *cheminDépôtDistant nomBrancheDistante*	Créer une copie de la branche actuelle sur une branche d'un nouveau dépôt distant
Mise à jour de branches	
pull *nomDuDépotDistant nomBrancheDistante*	Obtenir les dernières modifications de la branche distante sur notre dépôt local.
add *nomDesFichiers*	Ajouter les fichiers modifiés ou nouveaux à la zone de préparation de la branche locale courante.
rm *nomDesFichiers*	Effacer les fichiers de la branche locale courante. Ce changement doit être confirmé par un commit.
commit -m *"Commentaires"*	Enregistrer sur la branche courante locale les changements placés dans la zone de préparation. Les commentaires sont enregistrés pour décrire le commit.
reset *numeroDeCommit*	Efface tous les commits à partir du commit correspondant au numéro spécifié, impossible de revenir en arrière.
revert *numeroDeCommit*	Crée un nouveau commit qui renverse les changements du commit correspondant au numéro spécifié. Il est donc possible de conserver l'historique de ces changements.
Status	Vérifier s'il y a des fichiers non-ajoutés à la zone de préparation (unstaged) ou des changements non-commis (staged) à la branche courante locale.
push *nomDuDépotDistant nomBrancheDistante*	Pousser les commits de la branche locale courante sur la branche distante
Gestion des branches	
branch *nouvelleBrancheLocale*	Créer une nouvelle branche à partir de la branche courante locale
branch **-d** *ancienneBrancheLocale*	Supprimer une branche locale
merge *autreBrancheLocale*	Fusionner les changements d'une autre branche avec la branche courante.
checkout *autreBrancheLocale*	Choisir l'autre branche comme étant maintenant la branche active.

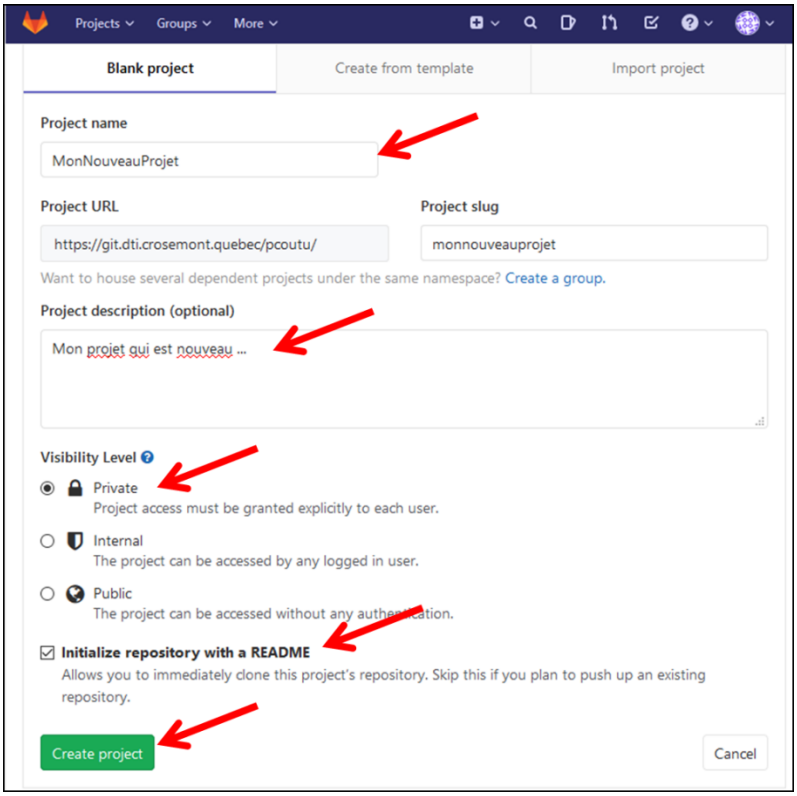
3 Créer et gérer un projet dans GIT-Lab

3.1 Créer un nouveau projet directement sur le site de GIT-Lab

Voici les étapes à suivre :

1. Assurez-vous que nom d'utilisateur et mot de passe vous permets d'accéder au courriel du département (vous devriez avoir reçu ces informations sur votre courriel personnel).
2. Allez sur le site https://git.dti.crosemont.quebec et connectez-vous avec le nom d'utilisateur et le mot de passe de votre compte du département.
3. Choisissez l'option ***Projects/Your Projects***
4. Cliquez sur ***New Project*** ou ***Create a Project*** (la 1ere fois)
5. Dans l'onglet ***Blank Project***
    - ajoutez un ***Project Name*** et une ***Description***
    - sélectionnez sur ***Private*** et ***Initisalize repository ...***
    - cliquez sur ***Create Project***



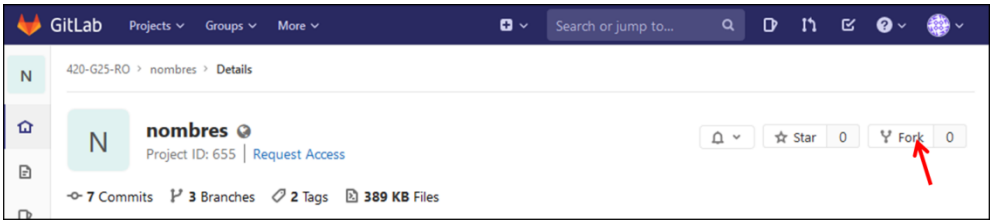


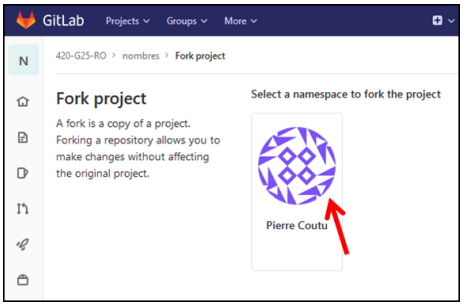
3.2 Créez un nouveau projet privé à partir d'un projet public

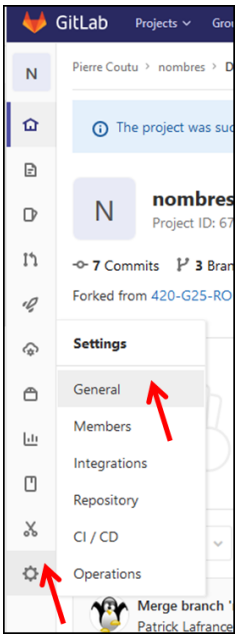
Il est aussi possible de créer un nouveau projet, à partir d'un projet public déjà existant, en créant une copie nommée bifurcation (**fork**). Il ne restera ensuite qu'à ajuster les paramètres de visibilités pour en faire un projet privé.

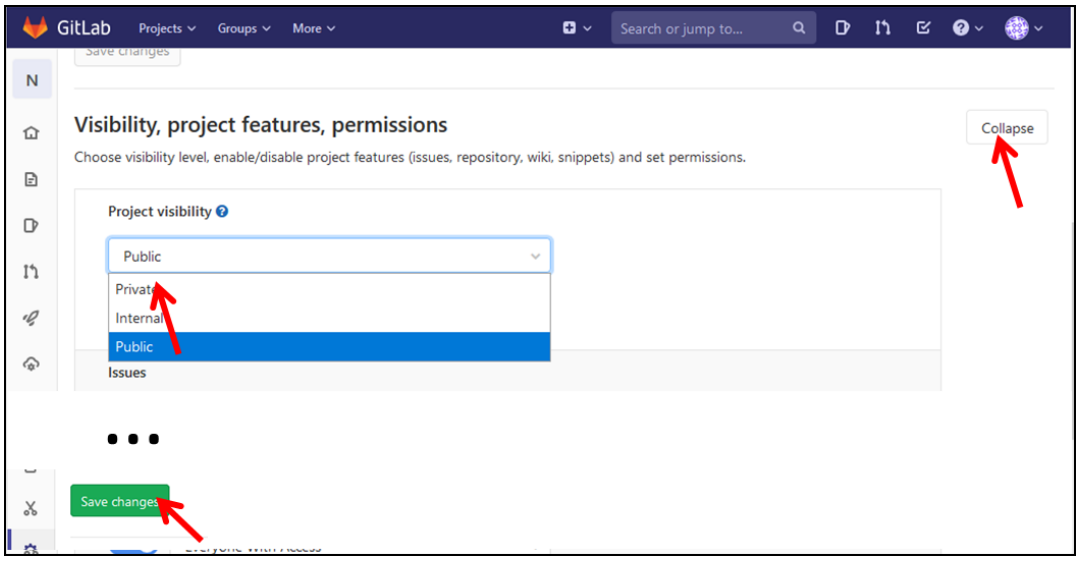
Voici les étapes à suivre :

- Sélectionnez et cliquez sur le projet public à copier.
- Cliquez sur le bouton **Fork** en haut à droite.
- Choisissez votre identité pour la destination.
- Choisissez le menu **Settings** (engrenage) avec la sous-option **General**.
- Dans la section Visibility, project …, cliquez à droite sur **Expand**/Collapse.
- Choisissez la visibilité ***Private*** et cliquez sur **Save changes** au bas.





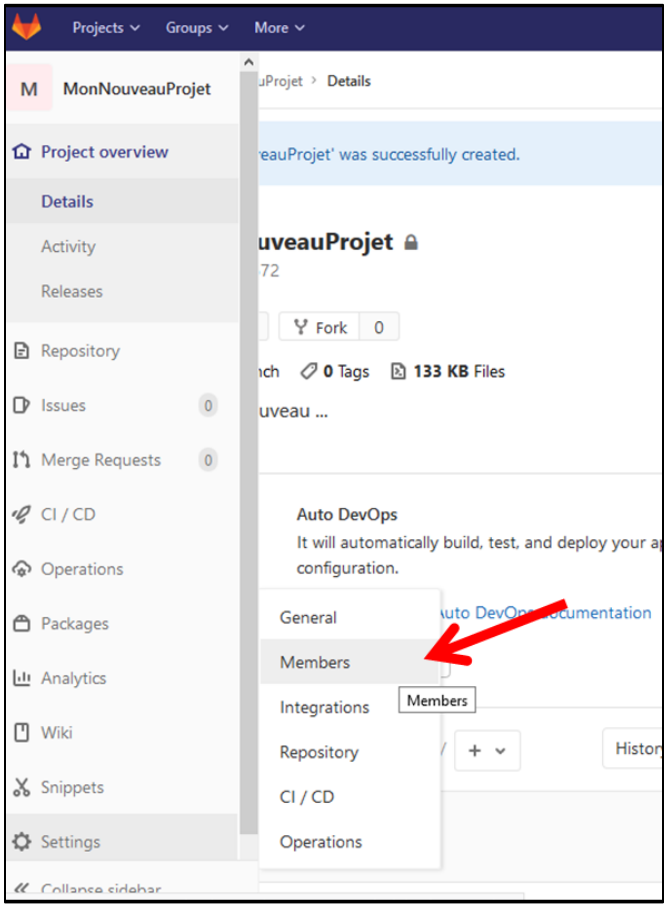


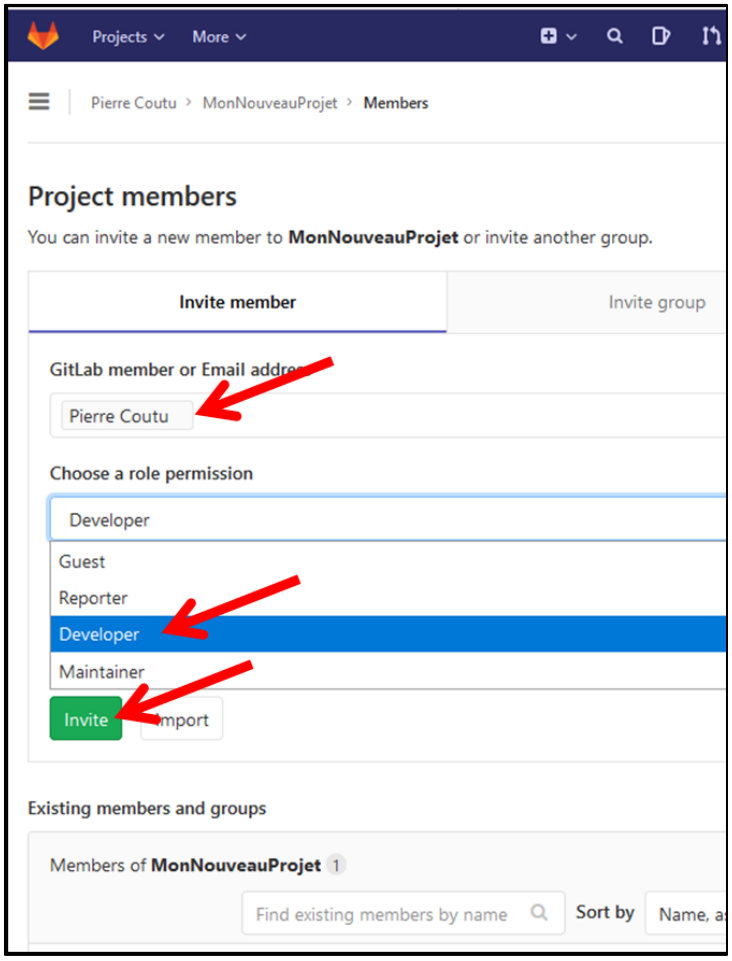


3.3 Ajoutez des membres au projet

Après avoir s'être connecté et listé tous vos projets (étapes 1 et 2 de la section 3.1)

1. Cliquez sur le projet pour y accéder
2. Choisissez l'option de menu **Settings/Member**
3. Dans l'onglet ***Invite Member***
    - ajoutez l'adresse courriel du membre à inviter
    - choisissez le **rôle** de ce nouveau membre (ex : ***Developper***)
    - facultatif : placer une date limite pour l'acccès
    - cliquez sur ***Invite***



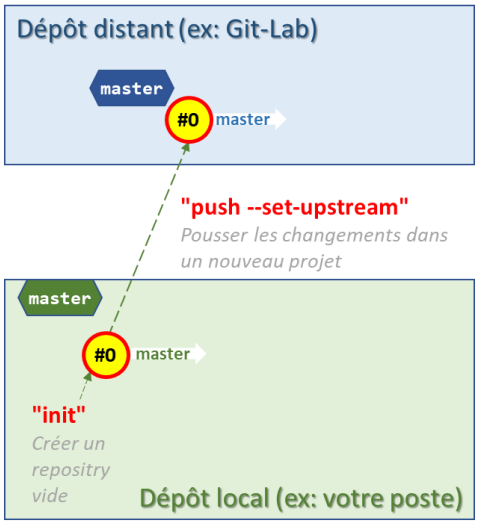


4 Importer/exporter un projet avec GIT

4.1 Créer un nouveau projet local et le pousser sur un dépôt distant

Dans l'invité de commandes GIT, déplacez-vous jusqu'à l'endroit où vous voulez placer votre dépôt local et tapez :

```
git init
```



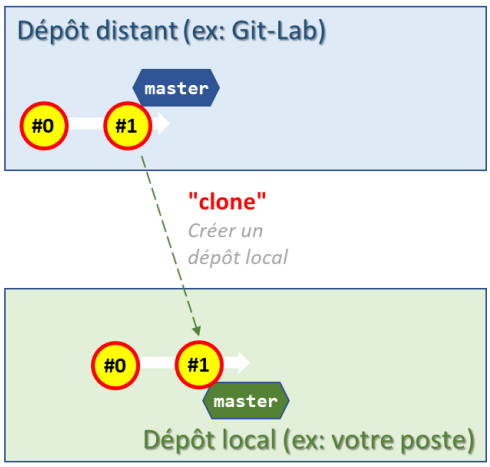
Et pour créer le dépôt distant correspondant, il suffit de placer la commande (dans l'exemple, le projet se nomme : *g16_demo4c01* , l'utilisateur : *pcoutu* et la branche : *master*)

```
git push --set-upstream https://git.dti.crosemont.quebec/pcoutu/
g16_demo4c01.git master
```

4.2 Cloner un projet GIT-Lab existant sur un dépôt local

Il est possible de prendre un projet public, ou un projet privé pour lequel vous êtes membres, et d'en créer un dépôt local (repository) sur votre poste. Voici les étapes à suivre :

- Déplacez-vous avec le navigateur Windows vers le dossier où vous voulez créer le dépôt. Le dépôt portera le nom du projet.
- Cliquez sur le bouton de droite de la souris et choisissez l'option ***GIT Bash Here***



Dans l'invité de commandes GIT :

- Clonez le projet (dans cet exemple : le projet *g16_demo4c03* et l'utilisateur *pcoutu*)

```
git clone https://git.dti.crosemont.quebec/pcoutu/g16_demo4c03.git
```

5 Exemple simplifié d'utilisation de GIT

5.1 Travailler directement sur la branche principale (non-recommandé)

Dans cette section nous allons illustrer les fonctionnalités de base de GIT en se concentrant sur une seule branche, soit la branche principale *master*. Cette **pratique n'est pas recommandée** pour les projets impliquant plusieurs développeurs.

Supposons que nous ayons déjà cloné un projet sur notre dépôt local (voir section 4.2) il y a quelques temps et que nous **voulions obtenir la dernière version** disponible à partir **du dépôt distant**. Puisque nous avons déjà une version sur notre dépôt local, on utiliser la commande ***pull*** au lieu de clone.

```
git pull origin master
```

Maintenant, supposons que nous voulions :

- **supprimer** un *demo4C_03.js*
- **ajouter** des images avec extension *.jpg* localisées dans un sous-dossier *images*
- **modifier** un *demo4C_03.html* déjà existant.

Après avoir fait ces changements directement sur les fichiers de notre dossier local (**zone de travail**), il faut ajouter ces changements à la **zone de préparation** de notre branche active locale (ici c'est la branche principale ***master***) avec les commandes **rm** et **add**.

```
git rm demo4C_03.js
git add images/*.jpg
git add demo4C_03.html
```

Puis, pour enregistrer ces changements à la **branche *master* locale**, nous allons utiliser la commande **commit** :

```
git commit -m "Ajouter images, effacer fichierA, ajout equals fichierB"
```

Le commentaire *–m* permet d'identifier les changements principaux apportés, pour être capable d'y référer dans le futur.

Pour vérifier que tous les dossiers et fichiers sur le dépôt local sont à jour, on peut appliquer la commande suivante :

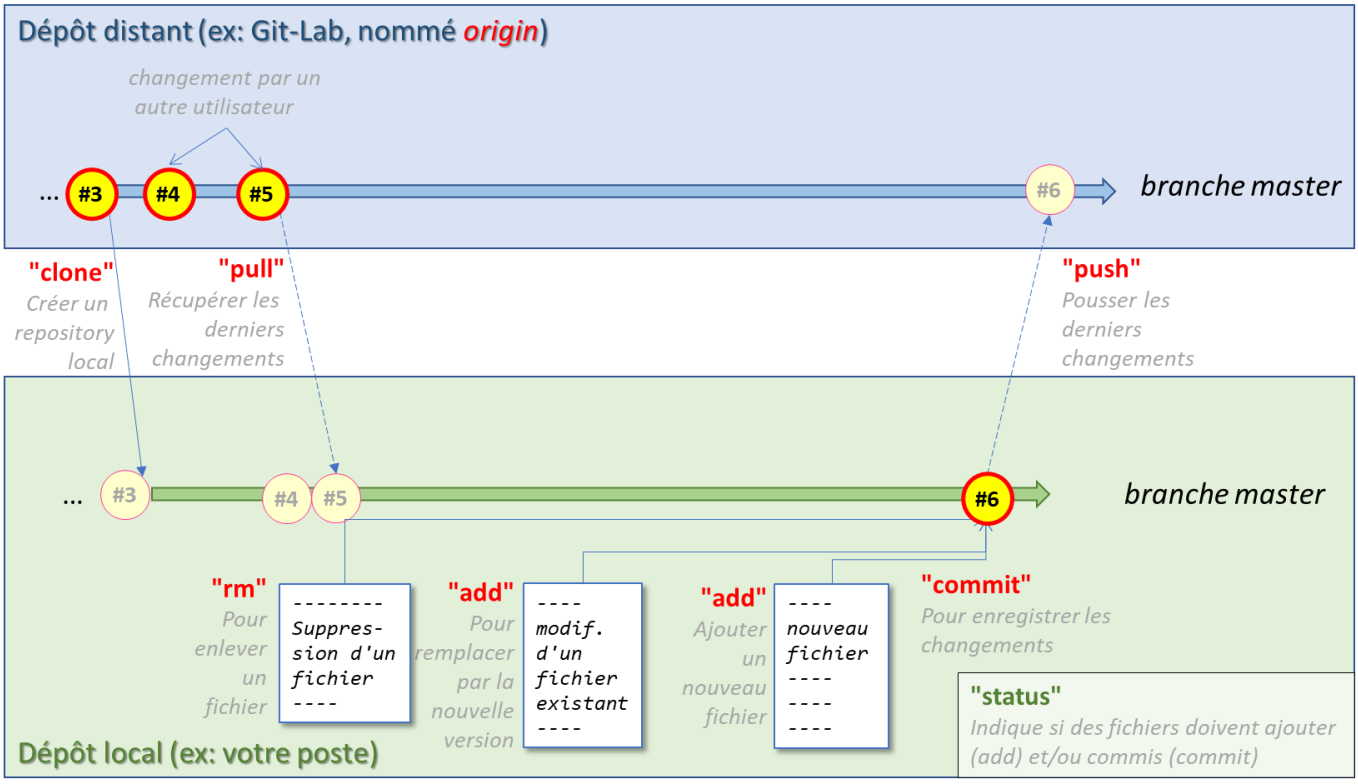
```
git status
```

Cette commande affichera tous les fichiers qui n'ont pas été ajoutés et supprimés de la zone de préparation, et toutes les ajouts et suppressions qui n'ont pas été enregistrées avec commit.

Pour terminer il suffit de pousser le dernier commit de la branche active sur la branche (*master*) du dépôt distant (*origin*) :

```
git push origin master
```

Le schéma suivant illustre le procédé …



5.2 Création de branches et fusion de branches

5.2.1 Créer une branche

En général, pour les systèmes de contrôle de versions, une branche est un chemin de développement alternatif au chemin principal. Bien que le résultat soit le même, avec GIT, un commit est une étiquette vers le dernier commit d'une séquence de commit qui l'ont précédé.

Pour créer une branche, il suffit de se placer sur la branche de laquelle on veut dériver, et d'utiliser la commande ***branch*** avec le nom de cette nouvelle branche. Par exemple, si on veut dériver une nouvelle branche nommée ***dev*** à partir de la branche courante, il suffit de taper :

```
git branch dev
```

5.2.2 Changer de branche

Lorsqu'on ajoute un commit, il s'ajoute à la fin de la branche active. Il est donc important de pouvoir changer la branche active. Ainsi pour travailler sur la branche ***dev*** on tapera :

```
git checkout dev
```

… donc, tous les commits qui suivront se feront sur cette branche.

Pour revenir à la branche principale nommée ***master***, il suffit de taper :

```
git checkout master
```

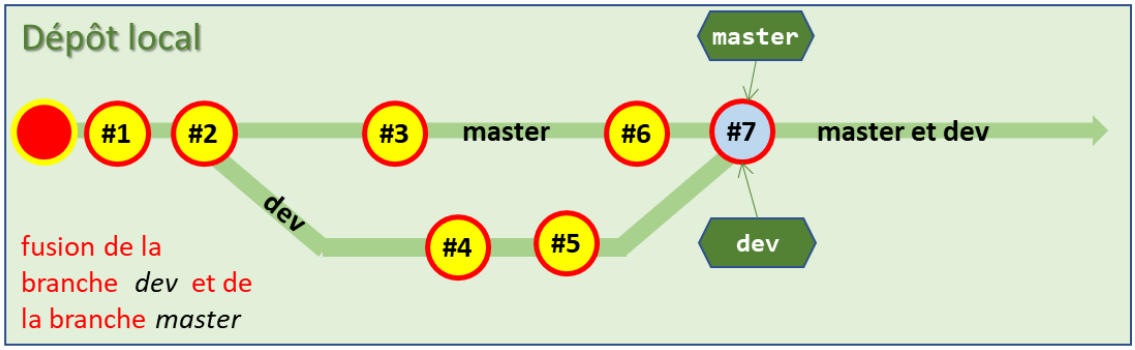
5.2.3 Fusionner deux branches (sans conflits)

Il faut se placer sur une des deux branches et utiliser la commande ***merge***. Si les changements apportés indépendamment sur les deux branches n'entrent pas en conflit, alors un nouveau commit à 2 parents sera créé et les deux étiquettes représentant les différentes branches seront identiques.

Par exemple, si on se trouve sur la branche ***master***, on peut y fusionner la branche ***dev*** avec l'instruction suivante.

```
git merge dev -m "commentaires …"
```

Comme pour un commit, l'option -m sert à décrire l'opération de fusion. Le schéma à la page suivante décrit l'opération.

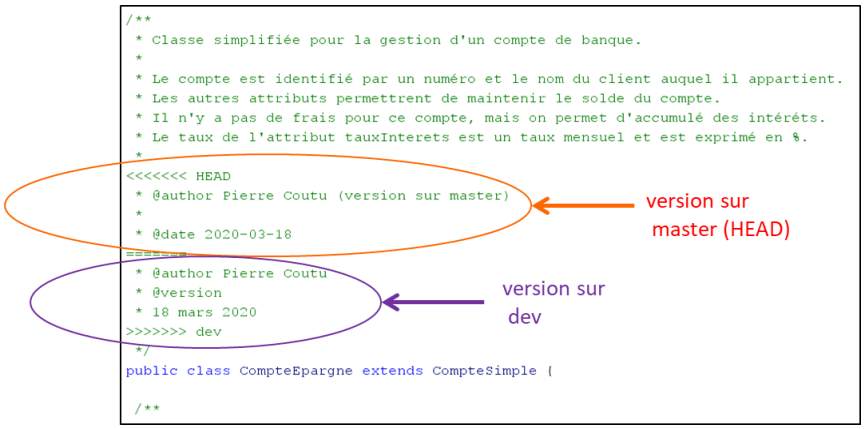


5.2.4 Fusionner deux branches (avec conflits)

Il se peut que des changements effectués sur les deux branches soit en contradiction. L'opération de fusion (***merge***) échouera, et les fichiers en conflit seront identifiés de cette façon, sur une branche temporaire *master|MERGING* :

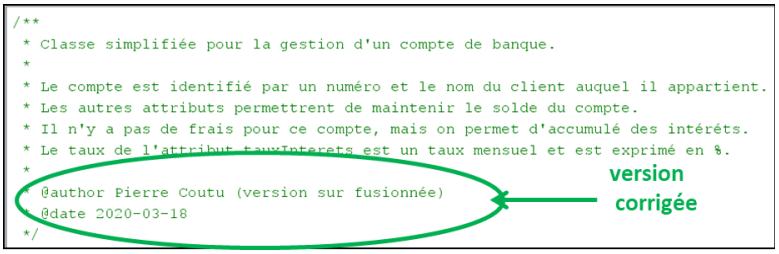
```
Pierre@Pierre-PC MINGW64 ~/Desktop/420-F16-RO/GIT/Test20200316/testbanquepc (master)
$ git merge dev -m "fusion dev master, commentaires et dates"
Auto-merging CompteEpargne.java
CONFLICT (content): Merge conflict in CompteEpargne.java
Automatic merge failed; fix conflicts and then commit the result.

Pierre@Pierre-PC MINGW64 ~/Desktop/420-F16-RO/GIT/Test20200316/testbanquepc (master|MERGING)
$
```

Il suffit d'ouvrir votre éditeur et de régler le conflit. Les sections en conflits seront indiquées par des `>>>>>>>>>>>` et `<<<<<<<<<<<`.



Il faut ensuite ajouter le fichier modifié dans GIT et faire un commit sur la branche temporaire *master|MERGING* et le conflit sera résolu et la fusion complétée.

```
git add fichier.java
git commit -m "…"
```



Note : Il existe des outils à 3 fenêtres qui permettent de comparer 2 fichiers dans 2 fenêtres supérieures placées côte-à-côte et de les fusionner dans une 3e fenêtre, placée en dessous.

6 Exemple d'utilisation de GIT (en équipe)

6.1 Travailler avec des branches et demande de fusions

En pratique, il est peu recommandé de travailler directement sur la branche principale « *master* » d'un projet, sauf pour les petits projets où l'on travaille seul. En fait, cette branche « *master* » est protégée par défaut et seules personnes ayant le rôle de gestionnaire principal (ou *maintainer* dans la nomenclature GIT) peuvent y apporter des modifications directement.

Lorsqu'on travaille en équipe, une stratégie est, pour **CHAQUE DEVELOPPEUR** (*developer* dans la nomenclature GIT), de :

1. **Récupérer la dernière version de la branche principale** « *master* » à partir du dépôt distant.

```
git clone https://git.dti.crosemont.quebec/pcoutu/g16_demo4c03.git
cd g16_demo4c03
…
git pull origin master      … étape optionnelle, si on vient de faire clone
```

2. Se **créer une branche de développement**, localement, à partir de la branche master (ex : la branche *devPierre*) (instruction : ***branch***).

```
git branch devPierre
git checkout devPierre
```

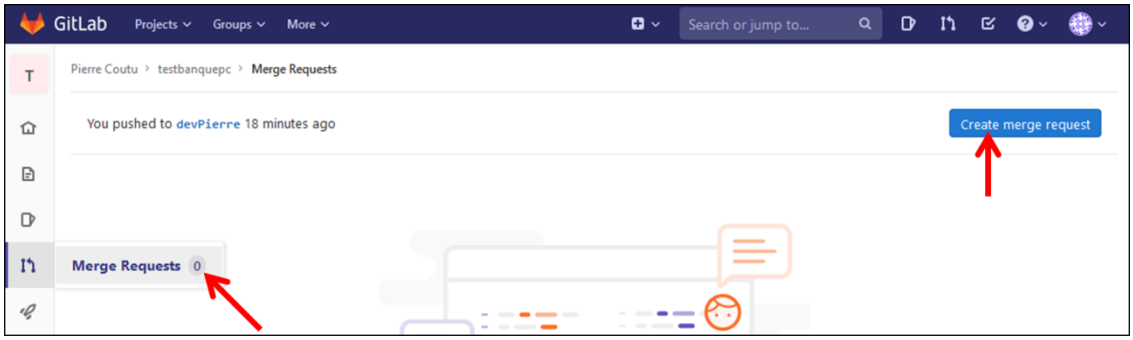
3. **Travailler sur cette branche** de développement, jusqu'à ce qu'on ait une portion significative et fonctionnelle de code à ajouter au projet (instructions : ***add***, ***rm***, ***commit***).

```
git add demo4C_03.html                      … étapes à répéter, jusqu'à ce
git commit -m "Détails des changements"     … qu'on fasse une demande
```

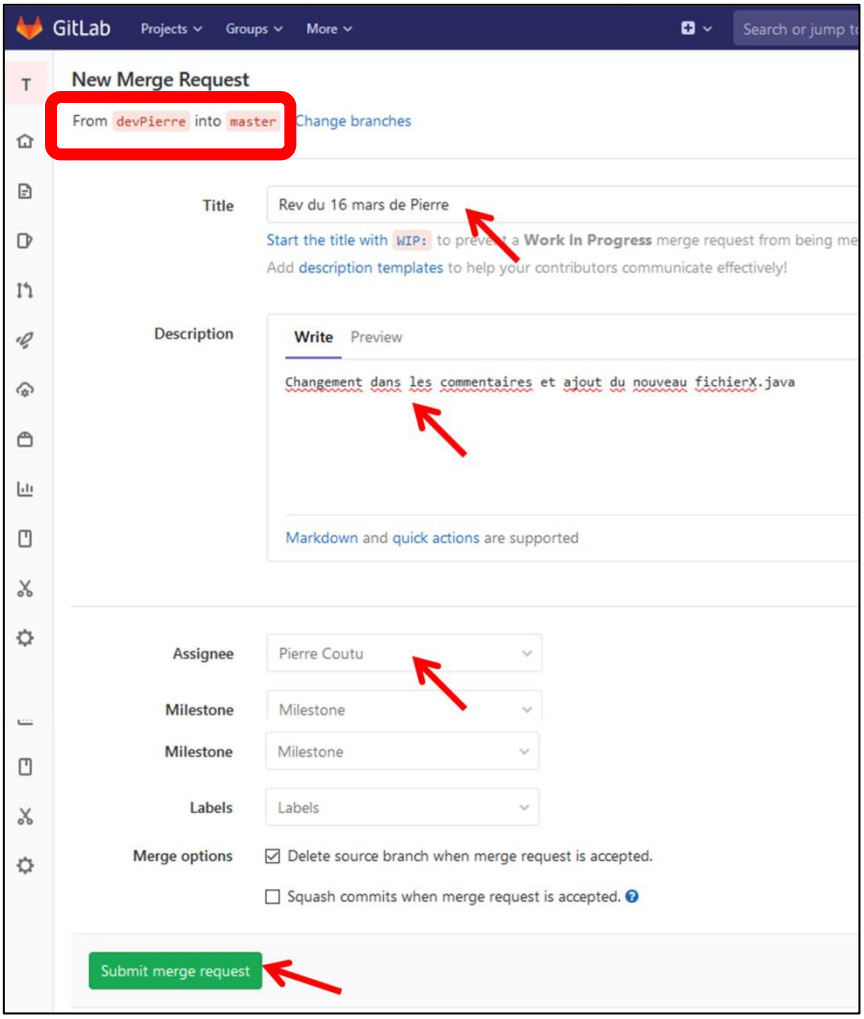
4. Ensuite, **pousser cette branche** sur le **dépôt distant** commun, i.e. sur Git-Lab (instruction : ***push***).

```
git push origin devPierre
```

5. Puis **dans l'interface web de GIT-Lab**, **effectuer une demande** pour que les changements que nous avons appliqués soient incorporés dans la branche principale « *master* ».
    - Sélectionnez le menu Merge Request

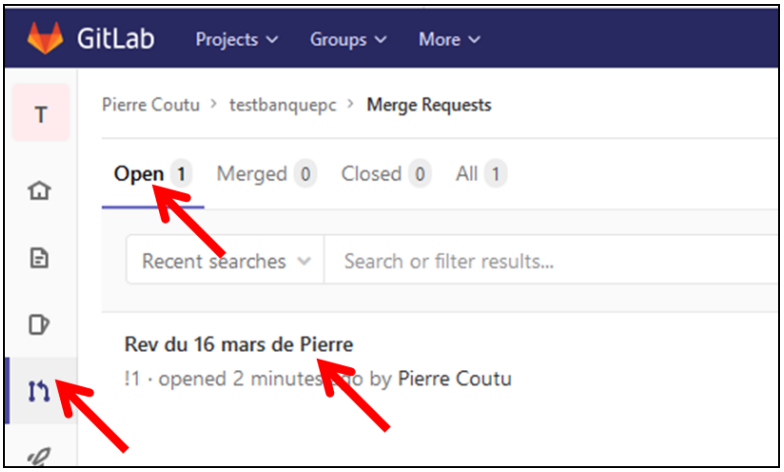


    - Remplissez les champs Title, Description, Assignee (personne qui fera la fusion avec la branche master) et le bouton vert:

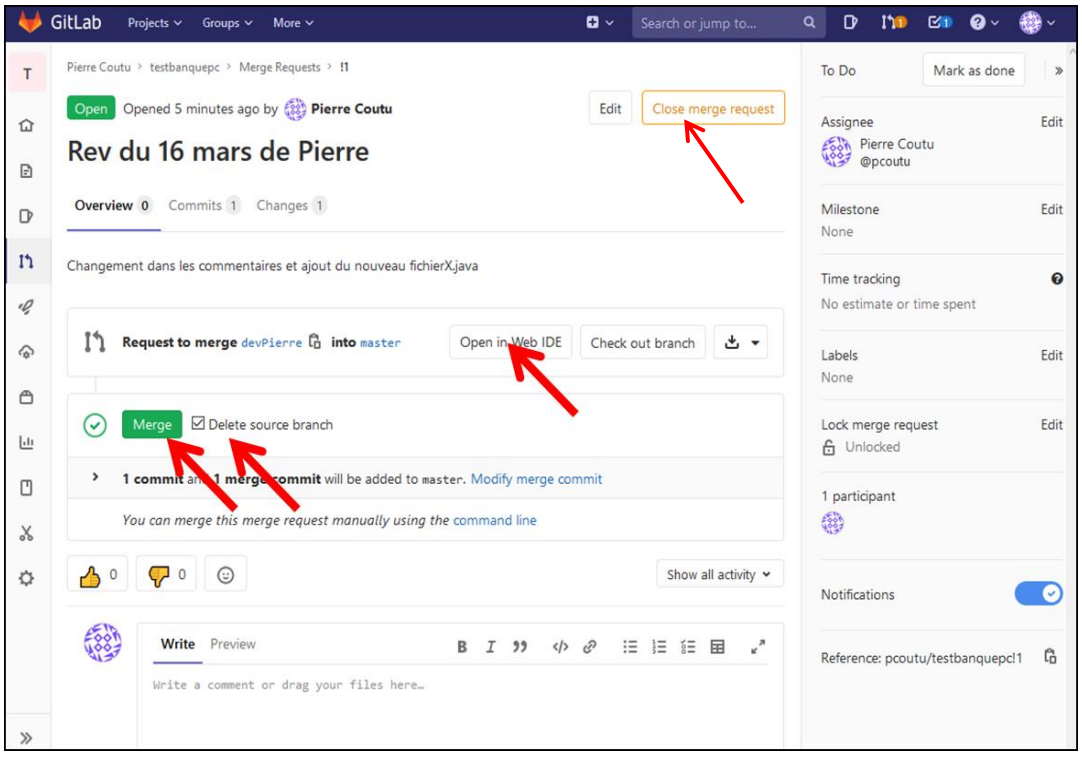


Ensuite, une des personnes ayant le **RÔLE DE RESPONSABLE** (*maintener*) pourra accepter cette requête et effectuer la fusion dans l'interface web Git-Lab :

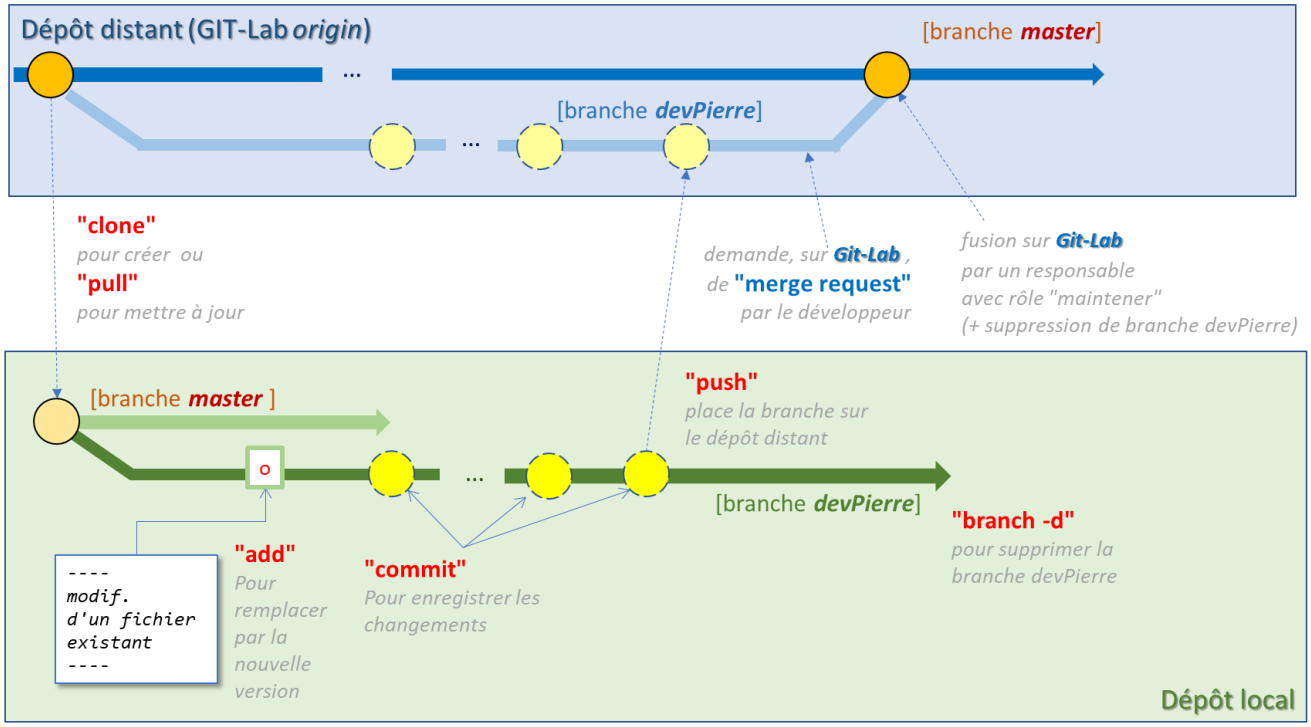
- Dans le menu merge request
  - Avec l'onglet « open », choisir la requête et l'ouvrir:



  - Révisez les changements avec le bouton « Open in Web IDE »
  - Acceptez les changements avec le bouton « merge », ou refusez avec le bouton « close merge request ». On peut sélectionner la case « Delete source branch » pour enlever la branche « DevPierre » une fois fusionnée.



Le résumé des opérations est illustré ci-dessous. Les cercles plus foncés, représentent l'endroit où s'est produit le commit ou l'opération original.



6.2 Créer des versions officielles

Il peut être nécessaire de sauvegarder un commit particulier à un certain moment charnière du projet pour créer des versions officielles (release). Il suffit alors de créer une branche et, possiblement modifier les propriétés de ce commit pour empêcher les changements.