



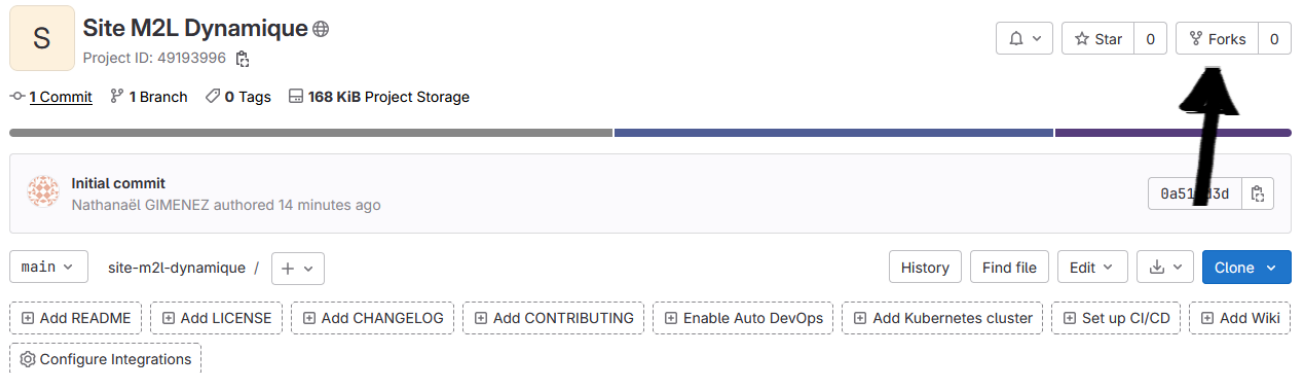
GIT AP3.1 M2L site dynamique.

1. Procédure à suivre par le chef de groupe.


Etape 1 :

 Se connecter à Gitlab puis aller sur le repository suivant : <https://gitlab.com/nath.gim.en/site-m2l-dynamique>


 Forkez le projet et renommez le en ajoutant le nom du groupe



Etape 2 :

 Attribuer les droits nécessaires aux autres membres du groupe et au professeur correcteur.

Etape 3 :


 Dans un dossier dédié (sur le disque E !), ouvrez **Git Bash** (clic droit -> Git Bash here) puis clonez le projet que vous avez forké avec la commande **git clone https://url_de_votre_projet**


Etape 4 :

 Créer la branche "develop".

git branch develop


Vous êtes automatique positionnés sur cette nouvelle branche. Vérifiez avec la commande **git status**

 Afficher les branches avec la commande **git branch**

 Envoyez cette nouvelle branche sur le serveur **git push origin develop**


La branche "develop" sera utilisée pour gérer le projet durant le processus de développement et la branche "main" contiendra le projet final (production)

Etape 5 :

 Créer la branche "comptesAautorisations".

git branch comptesAautorisations


Cette branche sera utilisée pour gérer la mission "gestion des comptes et des autorisations". L'ensemble des membres du groupe doivent participer à cette mission. Cette mission consistera essentiellement à modifier le contrôleur principal.

 Afficher les branches.

git branch

2. Procédure à suivre par les autres membres du groupe).

Etape 1 :

 Cloner le projet (sous GitBash) sur un disque local (poste ou réseau du lycée)
git clone

Etape 2 :

- ✚ Créer les branches locales pour les missions auxquelles le membre du groupe concerné sera affecté.

Exemples :

L'étudiant Jules Never devra disposer sur son dépôt local des branches :

- main
- develop
- comptesAautorisations
- clubsLigues

L'étudiant Albert Musca devra disposer sur son dépôt local des branches :

- main
- develop
- comptesAautorisations
- bulletins de salaire

- ✚ Créer une branche.
git branch "nom de la branche"
- ✚ Afficher les branches.
git branch

Les membres du groupe travailleront exclusivement sur les branches associées aux différentes missions (gestion des comptes et des autorisations, gestion des clubs et des ligues, gestion des bulletins de salaire, gestion des formations).

Ces branches seront fusionnées avec la branche "develop" durant le processus de développement. La branche "develop" sera finalement fusionnée avec la branche "main" lors de la mise en production de l'application.

3. Gestion des conflits.

Les conflits surviennent généralement lorsque deux personnes ont effectué des modifications en parallèle sur un même fichier.

Git ne peut pas choisir automatiquement la version correcte. Git se contente de signaler le conflit et arrêtera le processus de merge. Les développeurs concernés doivent alors se charger de résoudre le conflit.

Simulation d'un conflit.

Etape 1 :

Se positionner sur la branche "comptesAutorisations".

```
garri@DESKTOP-7CPTBCC MINGW64 /d/mesProjetsGit/m21 (develop)
$ git switch comptesAutorisations
Switched to branch 'comptesAutorisations'
```

Modifier le fichier "contrôleurPrincipal.php".

```
echo "Bonjour à tous";
```

Ajouter et valider les modifications.

```
garri@DESKTOP-7CPTBCC MINGW64 /d/mesProjetsGit/m21 (develop)
$ git add controleur/contrôleurPrincipal.php

garri@DESKTOP-7CPTBCC MINGW64 /d/mesProjetsGit/m21 (develop)
$ git commit -m "modification contrôleur principal develop"
[develop a51cb94] modification contrôleur principal develop
1 file changed, 1 insertion(+), 1 deletion(-)
```

Etape 2 :

Se positionner sur la branche "develop".

```
garri@DESKTOP-7CPTBCC MINGW64 /d/mesProjetsGit/m21 (comptesAutorisations)
$ git switch develop
Switched to branch 'develop'
```

Modifier le fichier "contrôleurPrincipal.php".

```
echo "Bonjour les membres du groupe"
```

Fusionner les branches "comptesAutorisations" et "develop".

```
garri@DESKTOP-7CPTBCC MINGW64 /d/mesProjetsGit/m21 (develop)
$ git merge comptesAutorisations
Auto-merging controleur/contrôleurPrincipal.php
CONFLICT (content): Merge conflict in controleur/contrôleurPrincipal.php
Automatic merge failed; fix conflicts and then commit the result.
```

Afficher le fichier "contrôleurPrincipal.php".

```
<<<<<< HEAD
echo "Bonjour à tous";
=====
echo "Bonjour les membres du groupe" ;
>>>>>> comptesAutorisations
```

Etape 3 :

- Visualiser les informations relatives au conflit.

```
garri@DESKTOP-7CPTBCC MINGW64 /d/mesProjetsGit/m21 (develop|MERGING)
$ git status
On branch develop
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   controleur/controleurPrincipal.php

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

- Modifier le fichier "controleurPrincipal.php".
Choisir entre les deux propositions.

```
echo "Bonjour les membres du groupe" ;
```

- Ajouter et valider la nouvelle version.
git add nom du fichier
git commit -m "gestion conflit ..."

4. Les commandes Git pour résoudre des conflits.

- Identifier les fichiers en conflit durant un merge.
git status
- Afficher la liste de commits en conflit entre les branches de merge
git log --merge
- Afficher les différences entre les états d'un dépôt/de fichiers.
git diff
- Annuler les changements apportés à des fichiers
git checkout
- Réinitialiser la branche à son état antérieur.
git merge --abort
- Réinitialiser les fichiers en conflit à un état fonctionnel connu.
git reset