

# Procédure de Tests Git LFS File Locking

## Prérequis :

- Le répertoire git contient un projet Unity.
- Le répertoire git est initialisé avec Git LFS.
- Le répertoire git contient deux branches ou plus divergentes en changements, dans le but de simuler un environnement de test similaire à un environnement réel.
- Le projet Unity contient au moins deux assets étant reconnu pour créer des conflits de merge (prefab, scène, image, fichier binaire).
- L'utilisateur 1 et 2 ont installé git et Git LFS et les commandes git et Git LFS sont accessibles par l'invite de commande de l'ordinateur.
- L'utilisateur 1 et 2 ont cloné le répertoire git localement.
- L'utilisateur 1 et 2 travaillent sur la même version de Unity.
- L'utilisateur 1 et 2 ont installé l'extension Unity suivante : <https://github.com/TomDuchene/unity-git-locks>
- La configuration du répertoire git contient : "lfs.setlockablereadonly false", selon la documentation de l'extension Unity.

## Workflow théorique :

1. Un utilisateur souhaite modifier un fichier commun
2. L'utilisateur verrouille le fichier
3. L'utilisateur modifie le fichier
4. L'utilisateur met à jour le serveur avec le fichier modifié
5. L'utilisateur déverrouille le fichier

## Test #1

Objectif : Déterminer si le mécanisme de verrouillage de fichiers fonctionne interbranche.

Étapes :

1. L'utilisateur 1 checkout la branche 1
2. L'utilisateur 2 checkout la branche 2
3. L'utilisateur 1 verrouille un fichier dans le projet
4. L'utilisateur 2 essaye de modifier ce fichier
5. L'utilisateur 2 sauvegarde ses changements avec un commit
6. L'utilisateur 2 essaye de push ses changements sur le serveur
7. Répéter en changeant l'utilisateur 1 et 2

Résultat attendu : Le mécanisme de verrouillage Git LFS permet le changement temporaire des fichiers verrouillés mais refuse d'apporter des modifications au répertoire serveur par un push d'un fichier verrouillé.

Résultat obtenu : Le verrouillage de fichier était visible interbranche par l'utilisateur 2, cependant celui-ci a réussi à modifier le fichier et le push sur le serveur tout de même.

Changement au workflow : **Réessayer le test en ajoutant le fichier à verrouiller dans “.gitattributes” pour qu’il soit traqué par Git LFS (voir test #2).**

## Test #2

Objectif : Déterminer si le mécanisme de verrouillage de fichiers fonctionne interbranche, **si ce fichier est tracké par Git LFS.**

Étapes :

1. L'utilisateur 1 exécute la commande : `git lfs track "*.prefab" --lockable`
2. L'utilisateur 1 commit et push ses changements sur main
3. L'utilisateur 1 exécute la commande : `git lfs migrate import --include="*.prefab" --everything`
4. L'utilisateur 1 push ses changements sur main
5. Les branches 1 et 2 sont rebased sur la branche main
6. L'utilisateur 1 checkout la branche 1
7. L'utilisateur 2 checkout la branche 2
8. L'utilisateur 1 verrouille un fichier dans le projet
9. L'utilisateur 2 essaye de modifier ce fichier
10. L'utilisateur 2 sauvegarde ses changements avec un commit
11. L'utilisateur 2 essaye de push ses changements sur le serveur
12. Répéter en changeant l'utilisateur 1 et 2

Résultat attendu : Le mécanisme de verrouillage Git LFS permet le changement temporaire des fichiers verrouillés mais refuse d'apporter des modifications au répertoire serveur par un push d'un fichier verrouillé.

Résultat obtenu : Résultat attendu 

Changement au workflow : Les fichiers devant être verrouillés par l'équipe devront être traqués et stockés par Git LFS. Cela ne cause pas de problème en particulier avec l'usage du stockage et l'usage de la bande passante puisque le prefab le plus gros obtenu pendant un projet antérieur était de 1,5 Mo et la taille moyenne d'un prefab était d'environ 10 Ko.

**CORRECTION :** Plutôt que d'exécuter la commande `git lfs track`, il est possible de simplement ajouter au fichier `.gitattributes` du projet l'extension d'un fichier suivi du mot "lockable" pour empêcher les changements au serveur sans qu'ils soient stockés dans le serveur LFS.

## Test #3


Objectif : Déterminer s'il est meilleure pratique de push un changement au serveur avant de déverrouiller un fichier.

Étapes :

1. L'utilisateur 1 checkout la branche 1
2. L'utilisateur 2 checkout la branche 2
3. L'utilisateur 1 verrouille un fichier dans le projet
4. L'utilisateur 1 apporte des modifications au fichier
5. L'utilisateur 1 sauvegarde ses changements avec un commit

6. L'utilisateur 1 déverrouille le fichier
7. L'utilisateur 2 verrouille le même fichier
8. L'utilisateur 2 apporte des modifications au fichier
9. L'utilisateur 2 sauvegarde ses changements avec un commit
10. L'utilisateur 2 push ses changements sur le serveur
11. L'utilisateur 1 essaye de push ses changements sur le serveur
12. L'utilisateur 2 déverrouille le fichier
13. L'utilisateur 1 push ses changements sur le serveur
14. Répéter en changeant l'utilisateur 1 et 2

Résultat attendu : L'utilisateur 2 sera en mesure de modifier le fichier précédemment verrouillé par l'utilisateur 1. L'utilisateur 1 ne sera pas en mesure de push ses changements au serveur jusqu'à ce que l'utilisateur 2 déverrouille le fichier.

Résultat obtenu : Résultat attendu , cela signifie par contre qu'il existe maintenant sur le serveur deux versions différentes du fichier (1 modifié par l'utilisateur 1 et l'autre par l'utilisateur 2). Cela pourrait éventuellement créer un merge conflict ou de la perte de travail.

Changement au workflow : Toujours s'assurer de push ses changements au serveur avant de déverrouiller un fichier.

## Test #4

Objectif : Déterminer si un utilisateur verrouillant un fichier travaille toujours sur la version la plus récente de celui-ci, peu importe la branche.

Étapes :

1. L'utilisateur 1 checkout la branche 1
2. L'utilisateur 2 checkout la branche 2
3. L'utilisateur 1 verrouille un fichier dans le projet
4. L'utilisateur 1 apporte des modifications au fichier
5. L'utilisateur 1 sauvegarde ses changements avec un commit
6. L'utilisateur 1 push ses changements sur le serveur
7. L'utilisateur 1 déverrouille le fichier
8. L'utilisateur 2 verrouille le même fichier
9. L'utilisateur 2 apporte des modifications au fichier
10. L'utilisateur 2 sauvegarde ses changements avec un commit
11. L'utilisateur 2 push ses changements sur le serveur
12. L'utilisateur 2 déverrouille le fichier
13. Répéter en changeant l'utilisateur 1 et 2

Résultat attendu : Lorsque l'utilisateur 2 verrouille le fichier modifié par l'utilisateur 1 dans une autre branche, il sera en mesure de modifier la version la plus à jour (celle de l'utilisateur 1).

Résultat obtenu : L'utilisateur 2 n'obtient pas la version la plus récente (modifiée par l'utilisateur 1) et donc est capable de verrouiller le fichier et de push une deuxième version de celui-ci sur le serveur.

Changement au workflow : Toujours s'assurer que le fichier est merged sur la branche de développement avant de le déverrouiller. Une fois qu'il est sur la branche de développement c'est maintenant la responsabilité au reste de l'équipe de s'assurer que leur branche est à jour avec la branche de développement.

## Workflow recommandé :

1. Un utilisateur souhaite modifier un fichier commun
2. L'utilisateur s'assure d'être à jour avec la branche de développement
3. L'utilisateur verrouille le fichier
4. L'utilisateur modifie le fichier
5. L'utilisateur met à jour le serveur avec le fichier modifié
6. L'utilisateur merge sa branche sur la branche de développement
7. L'utilisateur déverrouille le fichier