




Git LFS

WIZARDS & DICE

 Créateur : Hugo CLAMOND – Responsable infrastructure et système

 Date de Création : 06/12/2024

 Dernier modificateur : Axel MOURILLON – Chef de projet

 Date de modification : 06/01/2025

 Version : 2.0



Table des matières

I. Introduction.....	2
1/ Problème avec Git "classique"	2
2/ Ce que fait Git LFS.....	2
3/ Quand utiliser Git LFS.....	2
II. Avantages.....	3
A) Gestion des gros fichiers.....	3
B) Performance améliorée.....	3
C) Collaboration simplifiée.....	3
D) Conformité aux meilleures pratiques.....	3
E) Prise en charge native sur GitHub.....	3
III. Inconvénients.....	4
A) Quota limité sur GitHub.....	4
B) Complexité supplémentaire.....	4
C) Dépendance à Git LFS.....	4
D) Stockage et récupération des fichiers.....	4
IV. Estimation d'atteinte du plan gratuit de GitHub.....	6
A) Données importantes à collecter.....	6
B) Calcul de la bande passante consommée.....	6
C) Exemple pratique.....	6
V. À retenir.....	8





I. Introduction

Git LFS (Large File Storage) est une extension pour Git qui facilite la gestion des fichiers volumineux ou binaires dans un projet.

1/ Problème avec Git "classique"

Git est conçu pour suivre efficacement les fichiers texte, comme du code ou des fichiers de configuration, mais pas les fichiers lourds ou binaires (PDF, images, vidéos). Ces fichiers alourdissent rapidement le repo, car chaque modification est stockée dans l'historique, ce qui rend le clonage et le travail avec le repo plus lents.

2/ Ce que fait Git LFS

Git LFS remplace le fichier lourd dans l'historique par un simple pointeur (une petite référence textuelle). Le fichier réel est stocké ailleurs, sur un serveur dédié (ici GitHub s'en occupe).

3/ Quand utiliser Git LFS

Git renvoie un message d'erreur explicite lorsque l'on tente de pousser un fichier de 100 Mo sur GitHub : le push est rejeté et il est nécessaire de mettre en place LFS pour tracker le fichier en question.

Git renvoie un message d'avertissement lorsque l'on dépasse les 50 Mo, disant alors qu'il est préférable de tracker le fichier en question avec LFS. Le push n'est pas empêché dans ce cas.





II. Avantages

A) Gestion des gros fichiers

Git LFS est conçu pour gérer efficacement les fichiers volumineux comme les PDF, ODT, DOCX, JPEG, et PNG. Au lieu d'ajouter ces fichiers directement au repo, Git LFS les stocke séparément sur un serveur distant et ne conserve qu'un pointeur vers ces fichiers dans le repo principal. Cela réduit significativement la taille du repo à long terme : au fur et à mesure que le PTUT avance, on ajoute de la documentation, et donc des fichiers binaires au repo.

B) Performance améliorée

Sans Git LFS, chaque clone ou fetch du repo inclut l'historique complet, y compris toutes les versions des fichiers volumineux. Cela peut rapidement devenir problématique pour un membre avec une connexion lente, car le téléchargement de l'intégralité des versions historiques de fichiers lourds est coûteux en bande passante et en temps. Avec Git LFS, seuls les pointeurs sont récupérés, ce qui accélère ces opérations.

C) Collaboration simplifiée

Avec Git LFS, chaque membre récupère uniquement la version nécessaire des fichiers volumineux. Ça permet à ceux qui n'ont pas besoin des versions anciennes ou complètes d'accéder rapidement à la version du fichier dont ils ont besoin, rendant le tout plus fluide.

D) Conformité aux meilleures pratiques

Git est conçu pour gérer efficacement des fichiers texte, pas des fichiers binaires volumineux. Ajouter ces fichiers directement dans le repo peut entraîner une fragmentation et des problèmes de performance. Git LFS contourne ces limitations en déchargeant Git de la gestion directe des fichiers lourds.

E) Prise en charge native sur GitHub

Git LFS est entièrement intégré à GitHub, ce qui facilite sa mise en application. Une fois configuré, il est transparent pour les contributeurs du repo et s'adapte bien à un workflow Git classique.





III. Inconvénients

A) Quota limité sur GitHub

GitHub Free propose 1 Go de stockage LFS et 1 Go de bande passante par mois. Ces quotas peuvent être rapidement atteints si vous manipulez de nombreux fichiers volumineux ou si les membres du projet effectuent souvent des téléchargements.

En cas de dépassement, il faudra souscrire un plan payant ou trouver des alternatives. Demander son avis à F. DUCHEMIN par rapport à l'utilisation de LFS et si on estime que l'on pourrait arriver à ce cas-là.

B) Complexité supplémentaire

Git LFS nécessite une configuration initiale :

- Tous les membres doivent installer et configurer Git LFS localement (git lfs install).
- Les fichiers à suivre doivent être spécifiés manuellement dans le fichier .gitattributes.

Si un membre oublie de configurer Git LFS ou pousse un fichier lourd sans le tracker avec Git LFS, cela peut alourdir le repo et contredire l'objectif de la solution.

C) Dépendance à Git LFS

Incompatibilité sans Git LFS :

- Si quelqu'un clone le repository sans Git LFS, il recevra uniquement des pointeurs (des fichiers texte) à la place des fichiers réels. Cela peut créer de la confusion ou des problèmes, surtout pour les membres moins expérimentés.
- Si on migre le repo sur une autre plateforme ne supportant pas Git LFS (par exemple sur Gitlab après la fin du PTUT), on devra transférer les fichiers lourds manuellement ou réintégrer leur historique dans le repo principal.

D) Stockage et récupération des fichiers

Les fichiers lourds sont stockés sur un serveur externe, ce qui peut ralentir :

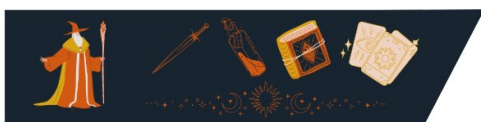
- Les opérations de pull ou checkout des fichiers, surtout avec une connexion lente.
- La collaboration en hors-ligne, car les fichiers ne sont pas stockés localement tant qu'ils ne sont pas explicitement téléchargés.

Ça peut paraître contre-productif par rapport au but initial de la manœuvre, alors je clarifie ce point en particulier :





- Téléchargement des fichiers réels à la demande :
 - Si un fichier volumineux est suivi avec Git LFS, il n'est pas téléchargé immédiatement lors du clone ou du pull. Si vous avez besoin de travailler avec ces fichiers (par exemple, pour ouvrir une image ou un document), Git LFS doit aller les chercher sur le serveur distant.
 - Ce téléchargement peut être plus lent, surtout avec une connexion réseau faible ou si le fichier est très gros.
- Collaborations en hors-ligne :
 - Sans Git LFS, tout est stocké dans le .git, donc on peut accéder à tout l'historique même hors ligne.
 - Avec Git LFS, si un fichier n'a pas encore été téléchargé et que l'on est hors ligne, on ne peut pas l'utiliser, car seul le pointeur est disponible localement.





IV. Estimation d'atteinte du plan gratuit de GitHub

A) Données importantes à collecter

Pour une estimation réaliste, il faut connaître :

- La taille moyenne des fichiers lourds suivis par LFS.
- Le nombre de fichiers ajoutés ou modifiés chaque semaine.
- Le nombre de fois où ces fichiers sont téléchargés via un pull, clone, ou checkout par les membres.

B) Calcul de la bande passante consommée

La bande passante est consommée à chaque fois qu'un fichier suivi par LFS est téléchargé :

- Lorsqu'un membre clone le repository :
 - Tous les fichiers LFS nécessaires à la branche principale sont téléchargés.
- Lors d'un pull ou checkout :
 - Les fichiers modifiés ou nouvellement nécessaires sont téléchargés.
- Lorsqu'un membre travaille sur une branche spécifique et accède à des fichiers qui n'ont pas encore été téléchargés localement.

Chaque téléchargement consomme la taille réelle des fichiers, pas celle des pointeurs stockés dans Git.

C) Exemple pratique

Les calculs suivants ont été réalisés à l'aide de ChatGPT

Supposons :

- Taille moyenne d'un fichier suivi par LFS : 10 Mo.
- Nombre de fichiers ajoutés/modifiés par semaine : 5 fichiers.
- Activité des membres :
 - Chaque membre fait un pull sur toutes les modifications une fois par jour.
 - Chaque membre clone le repository une fois tous les deux-trois mois.

Étape 1 : Calcul de la consommation hebdomadaire

- Taille totale des nouveaux fichiers ajoutés par semaine : $10 \text{ Mo} \times 5 = \mathbf{50 \text{ Mo}}$
- Bande passante consommée par les pulls, pour 4 membres et chaque jour) :
 $50 \text{ Mo} \times 4 \times 7 = \mathbf{1,4 \text{ Go}}$.





Étape 2 : Consommation liée aux clones

Si le repository contient 1 Go de fichiers suivis par LFS et que chaque membre clone le repository tous les 2 mois :

Bande passante mensuelle des clones : $1 \text{ Go} \times 4 = \mathbf{4 \text{ Go}}$

Un clone tous les 2-3 mois : $4 \text{ Go} \div 3 = \mathbf{1,33 \text{ Go par mois}}$

Étape 3 : Estimation globale

En combinant pulls (1,4 Go/semaine) et clones (4 Go/mois), on peut rapidement atteindre la limite gratuite en moins d'une semaine si l'équipe est très active.

Bien sûr, il est important de retenir que ce n'est qu'une estimation avec des chiffres initiaux revus volontairement à la hausse. Il est peu probable que l'on atteigne le 1 Go fatidique de fichiers stockés avec LFS.





V. À retenir

Git LFS accélère les opérations globales (surtout pour les repos avec de nombreux fichiers lourds et un historique riche), mais :

- Les opérations spécifiques sur les fichiers lourds (comme leur accès ou leur modification) peuvent être un peu plus lentes à cause du téléchargement à la demande.
- Il faut bien planifier l'utilisation de Git LFS, notamment si des membres de l'équipe ont une connexion instable ou doivent travailler hors ligne. Cela implique de télécharger les changements à partir desquels on souhaite travailler à l'avance.
- Faire attention à la consommation de la bande passante attribuée par le plan gratuit de GitHub chaque mois. (1 Go/mois, géré à la manière d'un forfait téléphonique)

Les alternatives possibles :

- Stocker les fichiers lourds sur un cloud dédié (ex : Google Drive)
- Autres suggestions apportées par l'équipe ;)

