***RÉFÉRENCE OBLIGATOIRE :*** <https://medium.com/mindorks/understanding-git-merge-git-rebase-88e2afd42671>

1- Quand faut-il utiliser rebase et quand faut-il utiliser merge ?

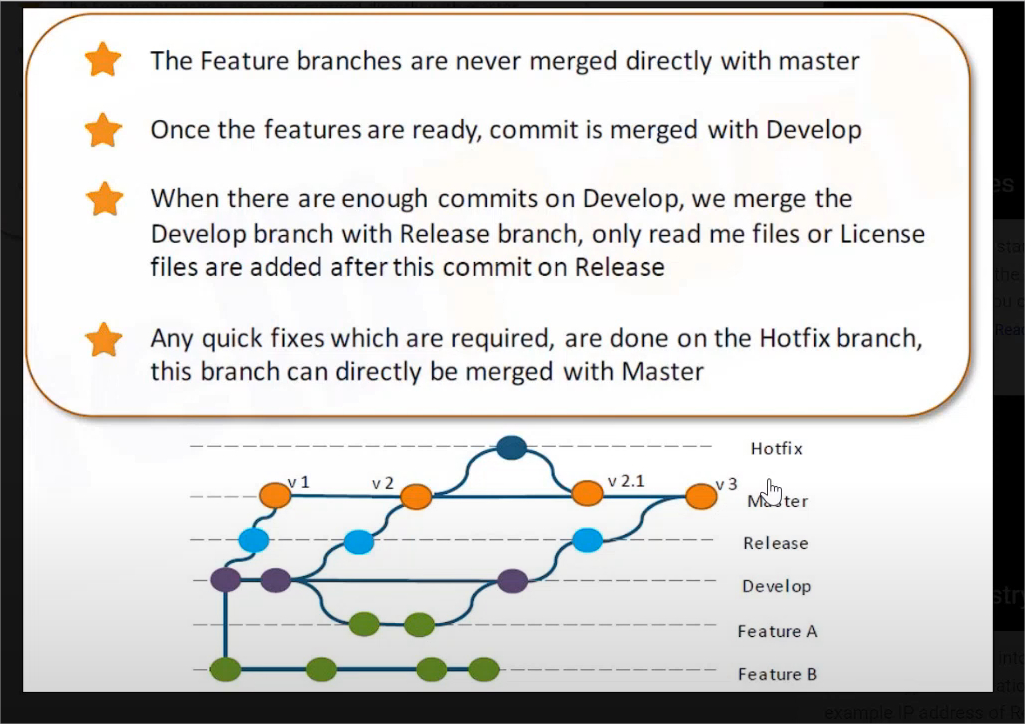
2 - Lorsque je suis sur la branche master, est-il préférable d'utiliser rebase ou merge ? Et si je suis sur la branche de fonctionnalité ?

3 - (Exemple 1) Quand je suis sur la branche principale (master), devrais-je opter pour rebase ou merge ? Et qu'en est-il lorsque je suis sur une branche de développement (feature branch) ?

4 - (Exemple 2) Quand je suis sur la branche principale (master), devrais-je opter pour rebase ou merge ? Et qu'en est-il lorsque je suis sur une branche de développement (feature branch) ?

5 - (Exemple 3) Quand je suis sur la branche principale (master), devrais-je opter pour rebase ou merge ? Et qu'en est-il lorsque je suis sur une branche de développement (feature branch) ?

6- Pour les meilleures pratiques, dois-je appliquer un rebase sur la branche master ou depuis la branche master ?



1- Quand faut-il utiliser rebase et quand faut-il utiliser merge ?

Voici une explication sur la différence entre rebase et merge dans Git, et quand utiliser l'un plutôt que l'autre:

### **Rebase (rebasage):**

Rebase (ou rebasage) est un moyen de **récupérer les changements d'une branche et de les appliquer sur une autre.** Lors de l'utilisation de rebase, ce qui se passe en réalité, c'est que les commits de la branche actuelle sont temporairement mis de côté, la branche actuelle est mise à jour avec les commits les plus récents de la branche spécifiée, puis les **commits temporairement mis de côté sont réappliqués** un par un sur la branche à jour.

**Quand l'utiliser:**

1. Historique linéaire: Si vous voulez un historique de commits propre et linéaire. Rebase évite les commits de fusion, ce qui rend l'historique plus lisible.
2. Avant de fusionner une branche de fonctionnalité dans la branche principale: Si vous avez travaillé sur une branche de fonctionnalité et que la branche principale a évolué entre-temps, vous pouvez utiliser rebase pour récupérer ces changements dans votre branche de fonctionnalité avant de la fusionner.

Inconvénients:

1. Modifie l'historique: Le rebasage modifie l'historique des commits, ce qui peut être problématique pour d'autres personnes travaillant sur la même branche que vous.
2. Peut être complexe: Si des conflits surgissent pendant le rebasage, ils doivent être résolus un par un pour chaque commit, ce qui peut être fastidieux.

### **Merge (fusion):**

Merge (ou fusion) prend les commits de la branche spécifiée et les fusionne avec la branche actuelle. Cela crée un nouveau commit de fusion.

Quand l'utiliser:

1. Simplicité: C'est souvent l'option la plus simple et la plus directe, surtout s'il n'y a pas de souci d'avoir des commits de fusion dans l'historique.
2. Garder l'historique intact: Si vous voulez garder tout l'historique des branches et savoir quand et où les fusions ont eu lieu.

Inconvénients:

1. Historique moins lisible: Si plusieurs fusions sont effectuées, l'historique des commits peut devenir difficile à lire et à suivre.

### **Résumé:**

* Utilisez rebase pour avoir un historique propre et linéaire ou pour **mettre à jour une branche de fonctionnalité avec les derniers commits d'une autre branche**.
* Utilisez merge pour une méthode simple et directe de combiner les commits de deux branches, surtout si vous ne voulez pas modifier l'historique des commits.

N'oubliez pas que rebase modifie l'historique des commits, alors soyez prudent lorsque vous l'utilisez sur des branches partagées.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2 - Lorsque je suis sur la branche master, est-il préférable d'utiliser rebase ou merge ? Et si je suis sur la branche de fonctionnalité ?

Si vous êtes sur la branche master (ou une branche principale comme main selon la nomenclature), vous allez souvent vouloir fusionner les branches de fonctionnalités (feature branches) dans master. Voici ce que vous pourriez envisager :

### **Sur master:**

* Utilisez merge pour intégrer les branches de fonctionnalités. Cela permet de garder un historique des points où des fonctionnalités ont été ajoutées ou des bugs corrigés. De plus, si plusieurs personnes travaillent sur des branches de fonctionnalités différentes, la fusion permet d'assurer que le travail de chacun est correctement intégré.

### **Sur une feature branch:**

* Utilisez rebase avant de fusionner votre branche de fonctionnalité dans master. Si vous avez été sur une feature branch pendant un certain temps, d'autres pourraient avoir apporté des changements à master. En rebasant d'abord votre branche de fonctionnalité sur master, vous appliquez vos changements sur le dernier état de master. Cela vous donne l'opportunité de résoudre les conflits sur votre branche de fonctionnalité plutôt que sur master. Une fois le rebase terminé et les éventuels conflits résolus, vous pouvez ensuite basculer vers master et utiliser merge pour intégrer votre feature branch.

Cela étant dit, ce n'est pas une règle stricte. Dans certains flux de travail, en particulier si l'historique linéaire est une priorité, vous pourriez vouloir toujours utiliser rebase, même sur master. Cependant, la méthode décrite ci-dessus est un bon point de départ général pour beaucoup d'équipes.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

3 - (Exemple 1) Quand je suis sur la branche principale (master), devrais-je opter pour rebase ou merge ? Et qu'en est-il lorsque je suis sur une branche de développement (feature branch) ?

Pour illustrer cela avec une analogie simple : Imaginez que vous travaillez sur la construction d'une route (votre code). La "branche master" est comme l'autoroute principale. Parfois, vous devez créer une petite voie d'accès ou une déviation (branche de fonctionnalité) pour des travaux spécifiques.

Si vous voulez intégrer cette voie d'accès dans l'autoroute principale sans perturber le flux, vous utiliseriez "rebase". C'est comme ajuster (rebase) la voie d'accès pour qu'elle s'aligne parfaitement avec l'autoroute avant de la joindre (merge).

Si vous voulez simplement lier la voie d'accès à l'autoroute telle qu'elle est, en conservant un enregistrement de l'endroit où elles se rencontrent, vous utiliseriez "merge". C'est comme construire un rond-point pour connecter la voie d'accès et l'autoroute.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

4 - (Exemple 2) Quand je suis sur la branche principale (master), devrais-je opter pour rebase ou merge ? Et qu'en est-il lorsque je suis sur une branche de développement (feature branch) ?

D'accord, utilisons une analogie pour simplifier la compréhension.

Imaginez que vous construisez une tour avec des blocs LEGO.

### **Merge (fusion) :**

Imaginons que vous construisez le tronc principal de votre tour (master). Pendant ce temps, votre ami construit une petite annexe séparée pour cette tour (c'est la feature branch).

Une fois que votre ami a terminé sa petite annexe, vous voulez l'attacher à votre tour principale. En utilisant la méthode merge, vous ajoutez simplement cette annexe à la tour principale en créant un nouveau niveau qui représente la jonction entre votre tour et l'annexe. Cette nouvelle couche représente le "commit de fusion".

### **Rebase (rebasage) :**

Maintenant, imaginons une situation légèrement différente. Vous continuez de construire votre tour principale pendant que votre ami construit son annexe. Mais avant d'attacher son annexe à votre tour, votre ami veut s'assurer que son annexe s'adapte bien à la hauteur actuelle de votre tour.

Alors, il démonte sa petite annexe (temporairement), attend que vous ajoutiez quelques niveaux supplémentaires à votre tour principale, puis il commence à reconstruire son annexe à partir de ce nouveau niveau. C'est comme s'il "transposait" son travail pour qu'il commence à partir du sommet actuel de la tour principale. C'est le processus de rebase.

### **Résumé avec l'analogie :**

* Merge : C'est comme ajouter une annexe construite séparément directement à la tour principale, avec un nouveau niveau pour montrer où elle a été attachée.
* Rebase : C'est comme ajuster l'annexe pour qu'elle démarre à partir du sommet actuel de la tour principale avant de l'attacher.

En utilisant ces méthodes dans le développement logiciel:

* Sur master, vous voulez généralement voir quand (et où) une feature branch a été ajoutée. C'est pourquoi vous faites un merge.
* Sur une feature branch, pour s'assurer que votre travail est compatible avec les derniers changements de master avant de le fusionner, vous faites rebase.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

5 - (Exemple 3) Quand je suis sur la branche principale (master), devrais-je opter pour rebase ou merge ? Et qu'en est-il lorsque je suis sur une branche de développement (feature branch) ?

**Contexte:**

Imaginez que vous et un ami écriviez un livre ensemble. Vous écrivez le corps principal du livre (c'est la branche master) et votre ami travaille sur un chapitre bonus (c'est la feature branch).

### **Merge (fusion):**

Pendant que vous travaillez sur le corps principal du livre, votre ami termine son chapitre bonus. En utilisant la méthode de merge, vous prenez simplement son chapitre et l'ajoutez à la fin du livre. Vous ajoutez également une note au début de ce chapitre pour indiquer qu'il s'agit d'un ajout spécial (cela représente le "commit de fusion").

### **Rebase (rebasage):**

Reprenons. Vous travaillez toujours sur le corps principal du livre et votre ami sur son chapitre bonus. Cependant, avant de vous donner son chapitre, il décide de relire tout le livre depuis le début, et de modifier son chapitre pour qu'il s'intègre parfaitement avec les dernières modifications que vous avez apportées. Il ajuste certaines de ses références, met à jour certains détails, etc. pour que tout soit cohérent. En essence, il "rebase" son chapitre sur les dernières versions du corps principal du livre. Une fois cela fait, le chapitre est ajouté au livre, tout comme dans le cas précédent.

### **Résumé avec l'analogie:**

* Merge: Vous ajoutez directement le chapitre bonus au livre et mettez une note pour indiquer qu'il s'agit d'un ajout.
* Rebase: Votre ami met à jour son chapitre bonus en fonction des dernières modifications du livre avant de vous le donner pour l'ajouter.

Dans le monde du développement:

* Avec merge sur master, vous pouvez voir quand (et où) une feature branch a été ajoutée grâce au "commit de fusion".
* Avec rebase sur une feature branch, vous assurez que les modifications sont à jour avec master avant de les fusionner.

Cette analogie souligne l'idée qu'avec rebase, nous adaptons notre travail en fonction des derniers changements, tandis qu'avec merge, nous ajoutons simplement notre travail tel que

​

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

​

6- Pour les meilleures pratiques, dois-je appliquer un rebase sur la branche master ou depuis la branche master ?

Les bonnes pratiques en matière de gestion des branches avec Git varient en fonction de la configuration de l'équipe et des préférences du projet. Cependant, une approche courante concernant l'utilisation de rebase et merge est la suivante :

1. Lorsque vous êtes sur votre branche de fonctionnalité (feature branch):
   * Avant de fusionner votre branche de fonctionnalité dans master, il est courant d'effectuer un rebase sur master. Cela permet de récupérer les dernières modifications de master et d'appliquer vos changements par-dessus, comme si vous veniez tout juste de créer votre branche de fonctionnalité. Cela a pour avantage de créer un historique linéaire, ce qui peut faciliter la lecture de l'historique.

git checkout feature-branch

git rebase master

1. Lorsque vous êtes sur la branche master :
   * Après avoir rebasé votre branche de fonctionnalité, vous pouvez fusionner cette branche rebasée dans master sans créer de commit de fusion (puisque l'historique a été linéarisé grâce au rebase).

git checkout master

git merge feature-branch

En résumé :

* Vous appliquez rebase sur la branche de fonctionnalité (par rapport à master).
* Vous effectuez merge à partir de la branche master pour intégrer les modifications de la branche de fonctionnalité.

Cependant, une note importante : rebaser une branche qui a déjà été poussée sur un dépôt distant (par exemple, GitHub ou GitLab) et partagée avec d'autres développeurs peut causer des problèmes, car le **rebase réécrit l'historique**. Dans ces situations, une communication claire avec l'équipe est essentielle pour éviter les confusions.

Dans le contexte d’un projet informatique, supposons que plusieurs personnes travaillent sur la même fonctionnalité. Le superviseur choisira le meilleur travail (gestion de conflits). Pour que les autres développeurs puissent continuer, il faut qu’ils mettent à jour leurs branches avec un rebase depuis la branche master.

Solution choisie (meilleur développeur choisi par le superviseur)

git checkout master

git merge feature-branch-fatemeh

Un autre développeur doit mettre à jour sa branche avant de continuer

git checkout feature-branch-haythem

git rebase master

Comme ça , feature-branch-haythem est au courant des dernières fonctionnalités dans master qui a été mis à jour depuis feature-branch-fatemeh.

Après haythem veut intégrer ses changements dans master

git checkout master

git merge feature-branch-haythem

*TP # 1 : La Chasse au Trésor Git !*

Contexte :

Dans ce projet, chaque étudiant contribuera à un dépôt Git partagé appelé "La Chasse au Trésor". Le but est de placer des indices (ou des énigmes) dans le dépôt, que les autres étudiants devront résoudre pour progresser dans la chasse. Chaque indice mène au suivant. Les étudiants devront collaborer, résoudre des conflits Git et interagir avec le dépôt tout en s'amusant !

Prérequis :

* Connaissance de base de Git
* Accès au dépôt Git du projet "La Chasse au Trésor"
* Token fourni par le professeur

Instructions :

1. Préparation :
   * Clonez le dépôt : git clone https://github.com/hrhouma/ChasseAuTresor.git
   * Placez-vous dans le répertoire du projet : cd ChasseAuTresor
2. Création de votre énigme :
   * Basculez vers une nouvelle branche avec votre nom : git checkout -b enigme\_[votreNom]
   * Créez un nouveau fichier avec votre nom : nano [votreNom].txt
   * Écrivez une énigme amusante et sauvegardez.
3. Publication de votre énigme :
   * Ajoutez et validez votre énigme :  
     git add [votreNom].txt

git commit -m "Ajout de l'énigme de [votreNom]"

* + Poussez votre branche vers le dépôt distant :

git push origin enigme\_[votreNom]

1. Résolution d'épreuves :
   * Basculez vers la branche main :

git checkout main

* + Mettez à jour votre branche :

git pull origin main

* + Tentez de résoudre au moins une énigme ajoutée par un camarade de classe. Lorsque vous trouvez la réponse, ajoutez-la à la fin du fichier correspondant.

1. Mise à jour de la branche principale :
   * Fusionnez votre branche avec main (après avoir résolu les éventuels conflits) et poussez vos modifications.
2. Revue :
   * Regardez le dépôt sur GitHub et observez les différentes branches et commits de vos camarades. Essayez de suivre l'évolution de la chasse au trésor !

*Objectifs d'apprentissage :*

* Comprendre les bases du travail collaboratif avec Git et GitHub
* Expérimenter le processus de résolution de conflits
* Encourager la communication et la collaboration entre étudiants

*Remarque :*

Le token fourni par le professeur doit être utilisé avec prudence. Le token est secret et ne doit pas être utilisé à des fins autres que l'exercice. Une fois l'exercice terminé, le professeur devrait révoquer ce token pour garantir la sécurité du dépôt.

***TOKEN pour le PUSH*** *:*

hrhouma

ghp\_fiVve8xmulxxwf9qQM2nQcRvkMxVtn29hooE

*SOLUTION - PARTIE 0 (SOLUTION TRÈS SIMPLE) :*

Conflits et Résolution lors de la Chasse au Trésor Git :

Supposons que tous les étudiants veulent mettre à jour le fichier indices.txt pour ajouter leurs énigmes, ce qui créera des conflits.

1. Léa commence :
   * Après avoir cloné et créé sa branche, elle met à jour indices.txt.

echo "Énigme de Léa : Je suis plein la journée, vide la nuit. Qui suis-je ?" >> indices.txt

git add indices.txt

git commit -m "Ajout de l'énigme de Léa"

git push origin enigme\_Lea

2.

Paul continue :

* Après avoir cloné et créé sa branche, il ajoute également son énigme à indices.txt.

echo "Énigme de Paul : Plus je sèche, plus je deviens humide. Qui suis-je ?" >> indices.txt

git add indices.txt

git commit -m "Ajout de l'énigme de Paul"

git push origin enigme\_Paul

3.

Marc se lance :

* Après avoir cloné et créé sa branche, Marc fait de même.

echo "Énigme de Marc : Je vole sans ailes. Je pleure sans yeux. Partout je vais. Qui suis-je ?" >> indices.txt

git add indices.txt

git commit -m "Ajout de l'énigme de Marc"

git push origin enigme\_Marc

Gestion des Conflits lors de la Fusion avec Master :

* L'enseignant décide de fusionner les énigmes dans master.

1. L'enseignant fusionne d'abord la branche de Léa sans problème, car c'est la première fusion.
2. En tentant de fusionner la branche de Paul, il y a un conflit puisque Paul et Léa ont tous les deux modifié indices.txt.  
   Pour le résoudre :

git checkout main

git merge enigme\_Paul

# Conflit détecté !

L'enseignant ouvre indices.txt, trouve le bloc de conflit, et le résout manuellement pour que les deux énigmes soient bien présentées. Il ajoute ensuite le fichier à l'index et termine la fusion.

​

git add indices.txt

git commit -m "Fusion de l'énigme de Paul après résolution de conflit"

3.

1. En fusionnant la branche de Marc, un autre conflit survient. L'enseignant le résout de la même manière que précédemment.

Après ces étapes, le fichier indices.txt dans la branche master contient toutes les énigmes, et les étudiants peuvent continuer à résoudre les énigmes des autres ou en ajouter de nouvelles. Ils doivent également gérer les conflits de la même manière lorsqu'ils fusionnent leurs solutions avec master.

**​**

*SOLUTION - VARIANTE I*

*(SANS REBASE ⇒ Beaucoup de conflits à gérer par le manager) :*

Nous avons utilisé git add indices.txt ? Nous devons maintenant suivre l'exercice indiqué ? N'oublions pas l'étape où l'on doit créer l'énigme : il faut changer de branche en utilisant git checkout -b enigme\_[votreNom], créer un fichier portant le nom de l'étudiant avec nano [votreNom].txt, y écrire l'énigme, puis sauvegarder.

Je vais développer un exemple détaillé avec trois étudiants :

Léa, Paul, et Marc.

La Chasse au Trésor Git : Aventure avec Léa, Paul et Marc

1. Léa commence :
   * Elle clone le dépôt.
   * Elle crée sa branche et son énigme.

git clone https://github.com/hrhouma/ChasseAuTresor.git

cd ChasseAuTresor

git checkout -b enigme\_Lea

echo "Je suis plein la journée, vide la nuit. Qui suis-je ?" > Lea.txt

git add Lea.txt

git commit -m "Ajout de l'énigme de Léa"

git push origin enigme\_Lea

L'enseignant décide de fusionner l'énigme de Léa avec master.

git checkout main

git merge enigme\_Lea

git push origin main

Paul poursuit :  
  
 git checkout -b enigme\_Paul  
 echo "Plus je sèche, plus je deviens humide. Qui suis-je ?" > Paul.txt  
 git add Paul.txt  
 git commit -m "Ajout de l'énigme de Paul"  
 git push origin enigme\_Paul

L'enseignant tente de fusionner l'énigme de Paul avec master. Comme il n'y a pas de conflit avec le fichier de Léa, la fusion se fait sans problème.  
 git checkout main  
 git merge enigme\_Paul  
 git push origin main

Marc se lance :

git checkout -b enigme\_Marc

echo "Je vole sans ailes. Je pleure sans yeux. Partout je vais. Qui suis-je ?" > Marc.txt

git add Marc.txt

git commit -m "Ajout de l'énigme de Marc"

git push origin enigme\_Marc

L'enseignant fusionne également l'énigme de Marc sans problème.

git checkout main

git merge enigme\_Marc

git push origin main

2. Résolutions d'épreuves par les étudiants :

Maintenant, imaginons que Paul veut ajouter sa réponse à l'énigme de Léa et que Marc veut faire de même au même moment. Les deux vont donc modifier le fichier Lea.txt.

* Paul trouve la réponse et la note :

git checkout enigme\_Paul  
 echo "Réponse de Paul : une chaussure" >> Lea.txt  
 git add Lea.txt  
 git commit -m "Paul résout l'énigme de Léa"  
 git push origin enigme\_Paul

Marc, presque simultanément, ajoute également sa réponse :

git checkout enigme\_Marc

echo "Réponse de Marc : un sac" >> Lea.txt

git add Lea.txt

git commit -m "Marc résout l'énigme de Léa"

git push origin enigme\_Marc

3. Fusion des réponses avec master et résolution des conflits :

L'enseignant fusionne d'abord la réponse de Paul, puis essaye de fusionner la réponse de Marc, mais un conflit survient.

Pour résoudre le conflit :

git checkout main

git merge enigme\_Marc

# Conflit détecté dans Lea.txt !

L'enseignant ouvre Lea.txt, voit les deux réponses (de Paul et Marc), et décide de les conserver toutes les deux. Après avoir arrangé le fichier, il le sauvegarde et finalise la fusion.

git add Lea.txt

git commit -m "Résolution du conflit entre les réponses de Paul et Marc"

git push origin main

​

​

*SOLUTION - VARIANTE II*

*(AVEC REBASE ⇒ Moins de conflits à gérer par le manager) :*

Où est intégré le rebase dans ce processus ? Car en réalité, les étudiants ne travailleront pas simultanément, et ils devraient effectuer un rebase avant de finaliser leurs commits, n'est-ce pas ?

**La Chasse au Trésor Git avec Rebase :**

1. Énigmes individuelles :

Chaque étudiant crée sa propre énigme :

***Léa :***

git checkout -b enigme\_Lea

echo "Je suis plein la journée, vide la nuit. Qui suis-je ?" > Lea.txt

git add Lea.txt

git commit -m "Ajout de l'énigme de Léa"

git push origin enigme\_Lea

* L'enseignant fusionne l'énigme de Léa avec main.
* Paul fait de même, puis Marc.

2. Résolutions d'épreuves avec rebase :

Paul veut répondre à l'énigme de Léa. Avant de le faire, il commence par s'assurer que sa branche est à jour avec main.

***Paul :***

git checkout enigme\_Paul

git fetch origin

git rebase origin/main

echo "Réponse de Paul : une chaussure" >> Lea.txt

git add Lea.txt

git commit -m "Paul résout l'énigme de Léa"

git push origin enigme\_Paul

Marc veut également répondre à l'énigme de Léa. Avant d'ajouter sa réponse, il met à jour sa branche :

git checkout enigme\_Marc

git fetch origin

git rebase origin/main

Pendant le rebase, **Marc découvre qu'il y a un conflit dan**s Lea.txt puisque Paul y a déjà répondu. Git indiquera le conflit.

**Marc résout le conflit manuellement dans le fichie**r Lea.txt, puis continue le processus de rebase.

git add Lea.txt

git rebase --continue

echo "Réponse de Marc : un sac" >> Lea.txt

git add Lea.txt

git commit -m "Marc résout l'énigme de Léa"

git push origin enigme\_Marc

L'utilisation de rebase de cette manière permet à chaque étudiant de travailler avec la version la plus à jour de la branche main. Cela signifie que lorsqu'ils trouvent et résolvent des conflits, ils le font dans le contexte de leur propre travail, plutôt que lors de la fusion avec main, ce qui facilite le processus.

​