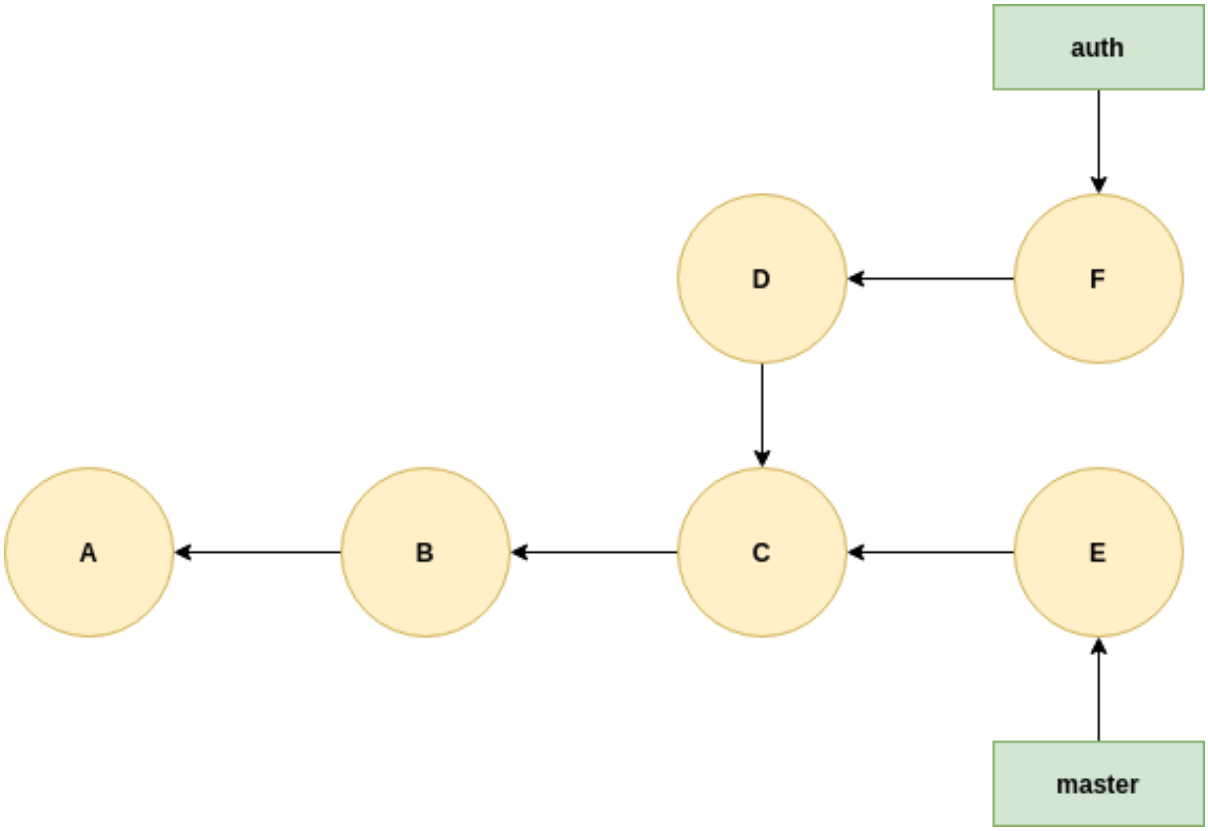
git rebase

Temps de lecture : 4 minutes

`Git` et le rebasage

Reprenons une situation que nous avons déjà vue.

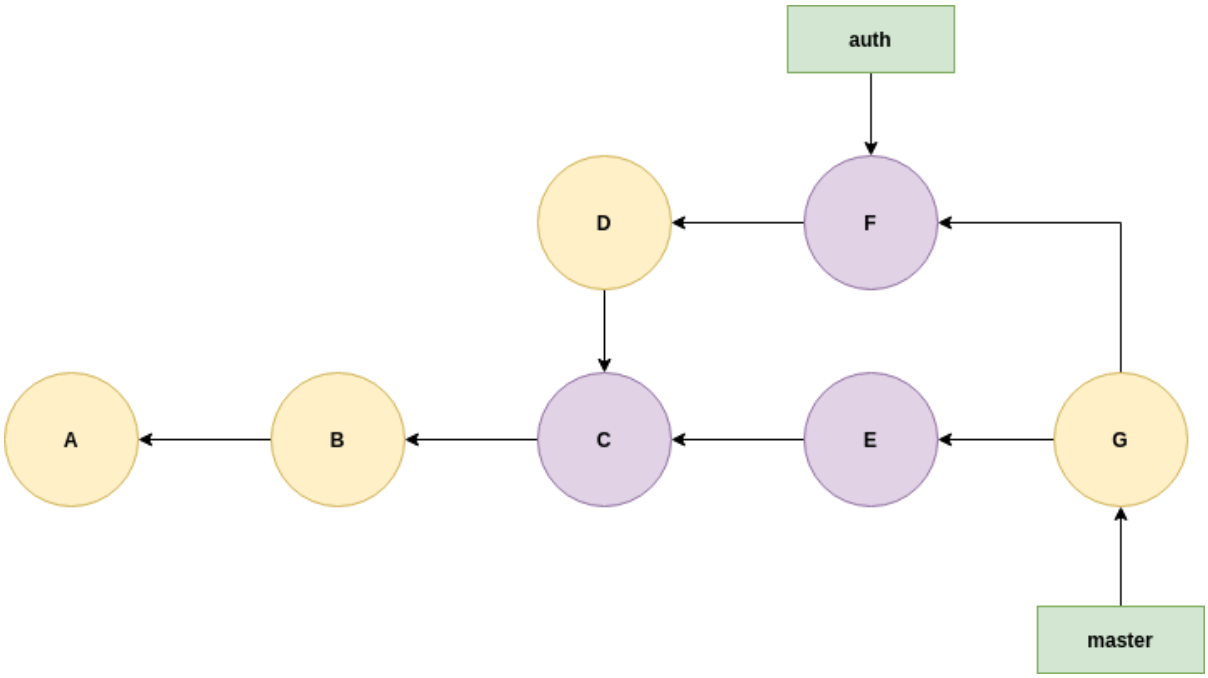


Jusqu'à maintenant nous avons vu que nous pouvions faire une **fusion à trois sources avec la stratégie récursive** dans cette situation.

```
git checkout main
git merge auth
```

Dans ce cas, les trois sources seront les versions correspondant aux `commits C` (premier ancêtre commun), `E` et `F` (`commits` en violet sur le schéma ci-dessous).

Dans ce cas, un nouveau `commit` de fusion serait créé, et si des conflits de fusion apparaissaient il faudrait les résoudre. Nous aurions ensuite :



Il existe également une autre méthode, qui n'est pas une fusion : **le rebasage**.

Cette méthode permet de prendre tous les changements de chaque `commit` sur une branche et de les appliquer les uns après les autres à la fin d'une autre branche.

Il suffit de faire, à partir de la situation originelle :

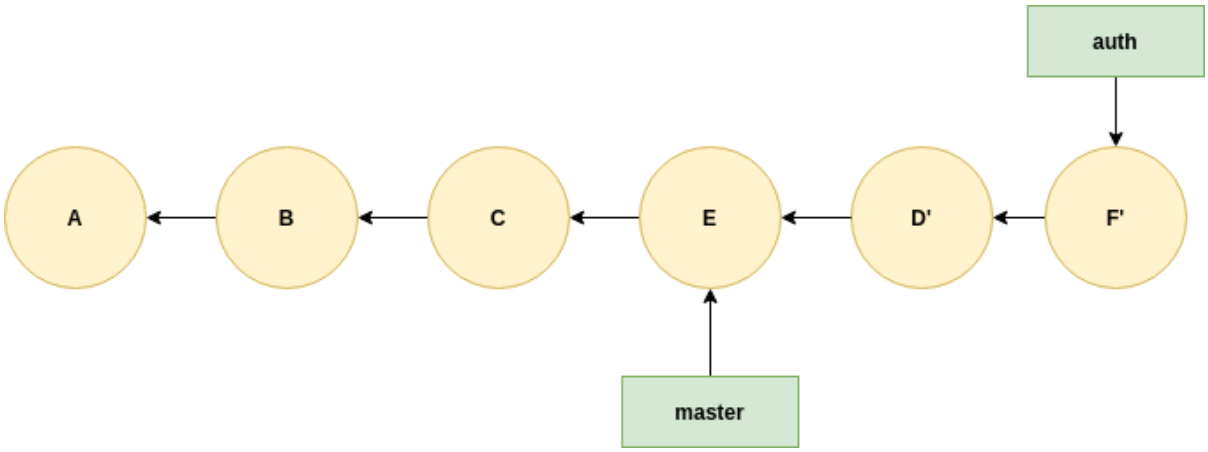
```
git checkout auth
git rebase main
```

Notez bien que nous faisons le `rebase` depuis la branche `auth` avec pour cible la branche `main`.

Vous aurez par exemple dans votre terminal :

```
Rembobinage préalable de head pour pouvoir rejouer votre travail par-
dessus...
Application de  Message de validation commit D
Application de  Message de validation commit F
```

Ce qui donne :



Notez bien l'ordre des `commits` ! Nous avons pris les `commits` de la branche `auth` et les avons **ajoutés à la suite de la branche** `main`.

Si des conflits apparaissent, il faudra bien sûr les résoudre.

Fonctionnement du rebasage

En fait, `Git` va rechercher le premier ancêtre commun aux deux branches (le `commit C` dans notre exemple).

Il va ensuite, prendre toutes les modifications de chaque `commits` sur la branche courante (ici `auth` et donc les `commits D` et `F`) et les enregistrer dans des fichiers temporaires.

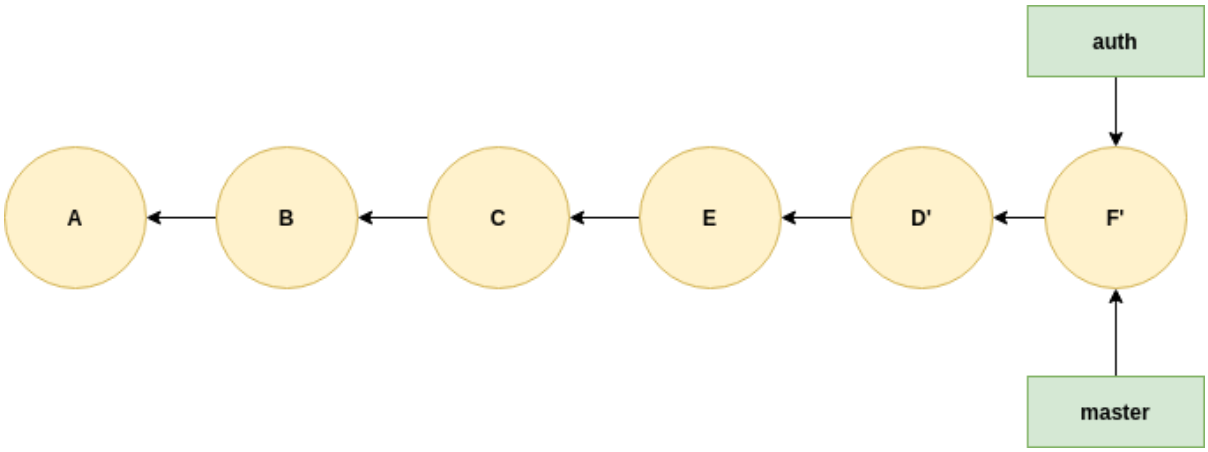
Puis, la branche courante (toujours `auth`) est réinitialisée au `commit` de la branche de destination (`commit E`).

Enfin, les modifications correspondants aux anciens `commits D` et `F` sont appliquées. Les nouveaux `commits` contiennent les mêmes modifications mais ont des `hashs` différents (c'est pour cela que nous les notons comme étant `D'` et `F'`).

Ensuite vous pouvez faire une **fusion en avance rapide** (`fast-forward`) ce qui revient à déplacer la branche `main` sur le même `commit` pointé par `auth` :

```
git checkout main
git merge auth
```

Vous aurez ensuite :



Le rebasage revient à supprimer la branche de l'historique et de faire comme si les changements avaient été appliqués à la suite de la branche ciblée.

Vous pouvez enfin supprimer la branche source (ici `auth`) :

```
git branch -d auth
```

Recommandations d'utilisation

Attention ! Il ne faut jamais rebaser des `commits` qui ont déjà été `push`.

Pourquoi ? Car comme nous l'avons vu en rebasant, les `commits` de la branche sont supprimés, et de nouveaux `commits` avec des `hash` différents seront créés.

De manière générale avec `Git`, et nous aurons l'occasion de le rappeler dans le chapitre suivant, **il ne faut jamais modifier l'historique si il a été envoyé sur le répertoire distant**, sinon vous casserez tout pour les autres.

Nous verrons en détails les recommandations de l'utilisation du rebasage avec les répertoires distants.

Mais pour le moment, **retenez que vous ne devez rebaser que vos branches qui n'ont pas été `push` ou alors dont vous êtes certain à 100% que personne d'autre n'a travaillé dessus.**