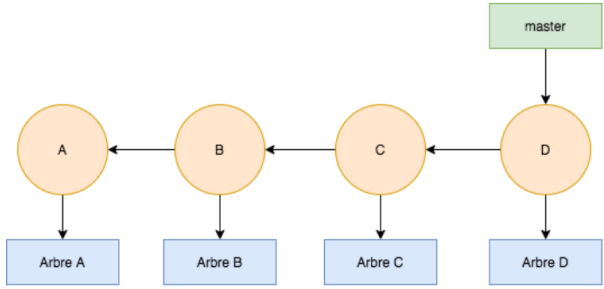
Introduction aux branches

Qu'est-ce qu'une branche ?

Nous avons vu que chaque `commit` (sauf le premier, appelé `commit` racine) a une référence à son `commit` parent, **formant ainsi une ligne ou une chaîne de** `commits` :



Parfois vous avez cependant besoin de travailler sans impacter cette ligne principale de développement, appelée **branche** `master`.

Les cas d'utilisation des branches

Des cas d'utilisation typique des branches sont principalement :

Premièrement, le développement d'une **nouvelle fonctionnalité**.

Deuxièmement, effectuer des **tests de fonctionnalité sans risquer d'interférer avec la version en production**.

Grâce aux branches, la ligne principale des `commits` n'est pas du tout impactée, ce qui permet d'avoir une branche propre généralement réservée à la mise en production : `master`.

Nous verrons en détails dans un chapitre dédié quel est le processus de développement classique avec une équipe qui utilise `Git` comme système de contrôle de versions.

Recommandations sur les branches

Les branches sont une des fonctionnalités principales de `Git`**, comme elles n'ont aucun coût d'espace mémoire (il s'agit simplement d'un pointeur sur un** `commit`**), elles sont très recommandées**.

En effet, elles ne prennent que 41 caractères (40 caractères du `hash` et un retour à la ligne), on dit donc que les branches en `Git` sont "gratuites", c'est-à-dire qu'elles ne coûtent quasiment aucune mémoire.

Ce qui prend de la mémoire en `Git`, comme vous le savez, c'est de sauvegarder un instantané du projet, et donc un `commit`. Il vaut donc mieux éviter de `commit` trop fréquemment : les `commits` sont conçus pour être des sauvegardes propres.

N'hésitez donc pas en revanche à vous servir de branches ! Sur un projet en équipe, il n'est pas rare d'avoir une dizaine de branches actives sur un projet.

Nous allons apprendre à bien gérer ses branches.