Qu'est-ce qu'un commit ?

Git est un outil de suivi de versions, qui permet de "suivre" les modifications apportées aux fichiers indexés dans un projet.

Pour expliquer ce qu'est un commit, reprenons la petite séquence suivante qui permet de créer un nouveau commit :

1. Lorsqu'on suit un projet avec "git", par exemple ici la réponse à un exercice, les modifications apportées aux fichiers sont listées dans le projet, et peuvent être affichées par la commande qui affiche l'état actuel du répertoire de travail du projet par rapport à l'index de "git" :
   `git status`
   Cela signifie à ce stade que ces modifications existent bien dans l'état actuel du projet (si on l'affiche sur un serveur local, les modifications faites sont bien visibles), mais qu'elles n'ont pas encore été "validées" pour le prochain "commit".
2. Ces modifications doivent alors être "validées", c'est-à-dire qu'il convient d'indiquer lesquelles doivent être inclues dans ce prochain "commit", ce que l'on fait avec la commande suivante qui ajoute tous les fichiers déjà suivis à l'index (mais n'ajoutera pas les fichiers que l'on vient de créer) :
   `git add .`
   Le ou les fichiers modifiés (et déjà indexés) sont alors "prêts à être commités", c'est-à-dire *prêts à être inclus dans le prochain commit*. Une variante de cette commande permet d'ajouter de nouveaux fichiers à l'index (ou de prendre en compte les fichiers supprimés) :
   `git add -A`.
3. Le "commit" lui-même est créé par la commande suivante, qui va créer un nouveau commit, et ajouter un commentaire pour décrire la raison et/ou le contenu des modifications :
   `git commit -m "Rédaction de l'exercice 1"`

   Un commit est un état précis des fichiers du projet indexés avec git. Cet état est identifié par une chaine de caractères réputée unique (un hash).

Du point de vue du développeur qui utilise "git", l'évolution d'un projet est ainsi composé d'une succession de "commits", commentés, qui correspondent à autant de modifications des fichiers du projet, commentées par leur auteur.

Par comparaison avec l'état précédent du projet, un commit peut être considéré comme la matérialisation d'une série de modifications que l'on souhaite enregistrer dans le projet : d'un point de vue très "pratique", un commit peut être vu comme une étape de plus dans l'avancement d'un projet.

Un commit peut aussi bien consister en la modification d'un seul caractère, qu'en l'ajout d'une librairie complète au projet., cela ne dépend que de la manière de les utiliser !

À quoi sert la commande git log ?

> **La commande "git log" permet d'afficher la liste des derniers commits**, c'est-à-dire l'historique des dernières modifications.

Cette liste précise l'auteur, la date et le commentaire associé à chacun des commits, et dispose de

diverses options qui permettent d'affiner les résultats sur une période ou un répertoire précis, afin de "voir ce qui s'est passé récemment" dans un répertoire précis.

C'est donc une commande informative, qui ne modifie rien... elle est tout à fait comparable à un `tail` sur un fichier de log apache.

Qu'est-ce qu'une branche ?

Une branche ou "fork" est une dérivation à partir du "tronc commun" d'un projet, avec lequel elle peut ensuite être fusionnée.

C'est une manière de cloisonner certains développements, comme par exemple le développement d'une nouvelle fonctionnalité ou d'un patch, mais aussi de maintenir et faire évoluer en parallèle plusieurs versions d'un même logiciel.

> **Une branche constitue un espace de travail isolé dans lequel il est possible d'apporter diverses évolutions sans perturber le développement de la partie principale du code.**

Les branches peuvent servir aussi bien seul qu'en collaboration, en permettant de structurer et d'organiser le travail sur différentes parties d'un projet, ou à plusieurs niveaux de maturité de ce projet. Une forme d'organisation simple peut ainsi consister à travailler dans une ou plusieurs branches de "dev" (par. ex. "dev_marc", "dev_utf8" ou "dev_v1.0.3"), puis à utiliser une branche "preprod" pour les tests, et enfin la "master" pour les versions prêtes à déployer.

Il est possible de passer d'une branche à l'autre afin de travailler successivement dans plusieurs parties d'un projet, ou d'avancer à la fois sur des tâches urgentes, tout en progresssant sur des projets à moyen terme.

Ce qu'il faut bien noter, c'est que le répertoire de travail sera toujours dans l'état de la branche sur laquelle on est positionnée : si on repasse sur la branche "master" après avoir travaillé dans une autre branche, le contenu des fichiers sera modifié pour réprésenter l'état de la branche "master".

Une branche peut être ensuite tout simplement abandonnée, ou fusionnée avec une autre branche. Dans ce cas, il peut être nécessaire de gérer des conflits de fusion, lorsque certains fichiers ont évolué différemment dans les deux branches.

Le processus de création, travail et fusion d'une branche peut être résumé avec les commande suivantes :

1. Créer une nouvelle branche et se positionner dessus
   ```
   git checkout -b ma_branche
   ```
   On travaille ensuite normalement dans "ma_branche", les commits étant dès lors enregistrés dans cette branche.
2. Pour changer de branche, on utilise
   ```
   git checkout autre_branche
   ```
3. Une fois le travail dans une branche terminé, on se repositionne sur la branche d'origine pour y intégrer son travail
   ```
   git checkout master
   git merge ma_branche
   ```
   Il est également possible de fusionner "l'avancée" de la branche master dans sa branche de travail, ou dans une autre branche.