Qu'est-ce qu'un commit ?

Derrière le mot « commit » se cache la commande fondamentale de Git. La traduction du mot « commit » (« commettre » en français) n'aide pas beaucoup à en comprendre la signification. En général, lorsqu'on commet quelque chose (crime, délit, péché), c'est de l'ordre de l'interdit. Est-ce que ça a rapport avec un logiciel de gestion de versions ? A priori, non. Du côté de l'étymologie, c'est un peu plus explicite puisqu'en latin « commettre » signifie littéralement « mettre ensemble ».
Bon, trêve de digression linguistique. Puisqu'il est difficile de trouver une traduction appropriée, voyons concrètement ce qui se passe lorsqu'on « fait un commit ». Imaginons que vous ayez travaillé sur votre projet c'est à dire que vous avez modifié ou créé au moins un fichier de code, de texte ou d'images. En clair, votre projet est mis à jour. Comme vous utilisez git, vous avez rassemblé ce ou ces fichiers dans un dépôt prêt à être « commité ». Ainsi lorsque vous lancez la commande git commit, c'est comme si vous preniez une photo de votre projet à l'instant t. C'est un instantané de votre projet. La syntaxe complète et habituelle de la commande est git commit -m « message ». Après le -m, on écrit entre les parenthèses à la place de « message », la description la plus précise possible de l'objet de ce nouveau dépôt. Ce point est important car plus les messages de commit seront précis, plus ils faciliteront la compréhension de l'évolution d'un projet. Et si par exemple, un projet fait fausse route, la lecture des messages permettront de retrouver plus facilement le commit à partir duquel le projet est parti dans la mauvaise direction. On pourra alors revenir à cet état du projet.

A quoi sert la commande git log ?

La commande git log permet d'obtenir l'historique de tous les commits d'un projet. Ceux-ci sont affichés dans l'ordre anti chronologique c'est à dire du plus récent au plus vieux. Chaque commit est indiqué avec son identifiant (Sha-1), le nom et l'e-mail de l'auteur, la date et le message du commit, la branche sur laquelle il se situe.
En fonction de la complexité du projet, de son nombre de branches et de contributeurs, de la fréquence des commits,… la consultation de l'historique peut devenir fastidieuse ou inadaptée pour comprendre l'évolution du projet. Heureusement, on peut adjoindre à git log de nombreuses options qui permettent de rendre la consultation plus facile et plus efficace. Par exemple, git log —all permet d'obtenir les commits de toutes les branches du projet et pas seulement ceux de la branche sur laquelle on travaille. L'option « —p » montre les différences introduites d'un commit à l'autre. Avec « —oneline », les informations de chaque commit sont condensées sur une seule ligne. On peut trier les commits par date, par auteur, limiter le nombre de commits étudiés, avec « —graph » on obtient une représention graphique de l'arborescence du projet,… Sachant qu'on peut combiner toutes ces options, il est possible de paramétrer très finement la commande git log pour obtenir exactement ce que l'on cherche à voir.

Qu'est-ce qu'une branche ?

Une branche permet de développer des fonctionnalités ou tester de nouvelles idées sans toucher le projet principal. Ça peut être très utile lorsqu'on veut développer une fonctionnalité qui risque d'introduire des régressions temporaires. Ainsi, on peut toujours faire des

changements dans la branche principale (corrections de bugs par exemple), tout en développant en parallèle cette nouvelle fonctionnalité.
On créé une branche avec la commande git branch <nom de la branche> et ensuite on travaille dessus comme on le ferait sur la branche master (la branche par défaut qui a été créée à l'initialisation du projet). Pour se positionner sur la nouvelle branche, on utilise la commande git checkout <nom de la branche>. Les commits sont envoyés dans cette branche mais ne touchent pas à la branche « master». Si on veut revenir sur la branche master, on saisi git checkout master. Plus tard, lorsque la fonctionnalité atteint un niveau de fiabilité suffisant, on fusionne la branche avec la branche « master ». On se positionne dans la branche master et on utilise alors la commande git merge <nom de la branche>. Il ne reste plus qu'à effacer la branche avec la commande git branch -d <nom de la branche>.

En résumé, dans la pratique, les branches sont utilisées pour permettre :

- la maintenance d'anciennes versions du projet (sur les branches) tout en continuant le développement des futures versions (sur le tronc) ;
- le développement parallèle de plusieurs fonctionnalités volumineuses sans bloquer le travail quotidien sur les autres fonctionnalités.