Quand on développe un projet il ne suffit pas de coder, mais il faut aussi penser aux évolutions possibles du code et des fichiers indexés dans le projet, aux erreurs que l'on peut faire, aux personnes qui vont aussi travailler sur le projet… C'est pourquoi il est indispensable de mettre en place dès le début un outil qui permet de gérer tout cela.
Le but de l'outil est de permettre à une équipe de développement de suivre les différentes versions d'un projet pour en gérer l'évolution, et de garder une copie de toutes les versions des documents, avec pour chaque version, l'identité de la personne qui a fait les modifications et les détails de cette modification.
Nous parlons ici des logiciels de gestion de versions qui sont nombreux, avec plus ou moins de fonctionnalités, et celui dont je vais vous parler aujourd'hui est au top depuis quelques temps déjà et est parti pour servir encore de longues années, il s'agit de Git.
Avec Git nous allons utiliser plusieurs mots et commandes que vous ne connaissez peut-être pas, et le but de cette exercice est d'expliquer :
-Qu'est-ce qu'un commit.
-À quoi sert la commande git log.
-Qu'est-ce qu'une branche.

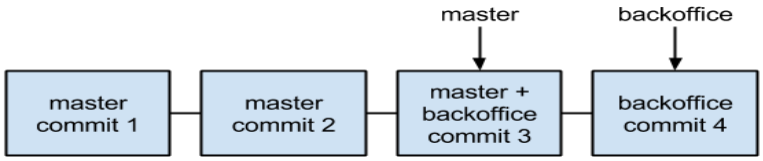
1/ Commit ou validation :
Un commit n'est ni plus ni moins qu'une version du projet, un instantané pris à un instant T. A chaque fois que vous faites un commit, Git enregistre un «arbre» qui permettra d'identifier vos fichiers modifiés et lui associe un objet commit qui, lui-même, pointera vers le (ou les) commit immédiatement précédent.
Le commit 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 " votre commentaire".`

2/ À quoi sert la commande git log ?
La commande "git log" est une commande informative, qui ne modifie rien, elle 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.

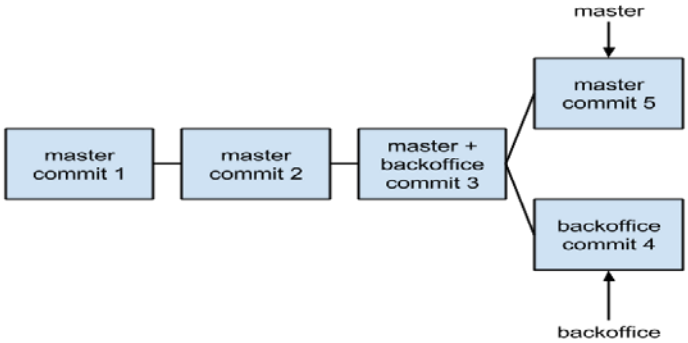
3/ Qu'est-ce qu'une branche ?
La notion de branche est certainement la plus difficile à comprendre dans Git. Nous venons de voir ce qu'est un commit, une branche est simplement un pointeur vers un commit. Puisque chaque commit connaît son précédent, il est possible de remonter de commit en commit pour retracer toute la branche. Pour simplifier, imaginons que nous avons créé un projet et fait 3 commits. Alors que nous ne nous sommes pas souciés de créer une branche, Git l'a fait pour nous. Dès le début le projet possède une branche par défaut, appelée master. Après nos 3 commits, master pointe vers le dernier commit en date.

Nous souhaitons maintenant créer une branche pour permettre à une autre équipe (le nom de l'équipe est : back-office) de travailler de son côté, Git va simplement créer un nouveau pointeur vers le dernier commit de la branche courante (donc master). Avec la commande suivante on va créer une nouvelle branche nommée «backoffice» : **`git checkout -b backoffice.`** Nos branches master et backoffice sont pour le moment identiques

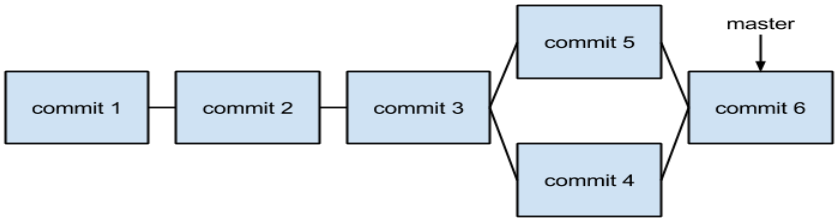
Si nous activons la branche backoffice (`git checkout backoffice`) et que nous faisons à nouveau un commit, le pointeur de la branche backoffice va se déplacer pour pointer sur le nouveau commit tandis que le pointeur de master ne bougera pas.



Si nous nous plaçons à nouveau sur master (`git checkout master`) et que nous faisons encore un commit, master va venir pointer vers ce dernier et nous aurons deux branches parallèles.



Je vous ai dit qu'un commit pouvait faire référence à plusieurs commit précédents, c'est le cas lorsque l'on fusionne deux branches, dans notre exemple la fusion de backoffice dans master entraînera la création d'un nouveau commit dont les «parents» sont à la fois c4 et c5.
On se repositionne sur la branche d'origine (`git checkout master`) et pour fusionner les branches on utilise la commande suivante : `git merge backoffice.`



Donc 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.