GIT

L'utilisation de Git dans un projet de programmation est une chose courante. Git est un logiciel de version décentralisé :

- Git sauvegarde toutes les modifications effectuées au cours du temps sur chaque document, et garde ainsi en mémoire les différentes versions de ces documents
- Git permet de cloner le dossier de travail sur différentes machines et de faire des modifications séparément sur ces différentes machines avant de fusionner toutes les modifications effectuées

L'utilisation de Git se fait en général avec celle d'un serveur sur lequel une copie du dossier de travail est présente, le dépôt distant, qui est synchronisée avec toutes les autres copies faites, les dépôts locaux.

Au début du stage, la version 0.1 de **udavis** était disponible dans un git sur un serveur du logiciel en ligne GitLab. Le projet a été à ce moment-là fourché : une copie exacte a été créée, mais indépendante et non synchronisée avec la copie originale. Le stage avait donc son propre dépôt dans lequel travailler, bien que les améliorations apportées ont par la suite été fusionnées au fur et à mesure dans le dépôt originel.

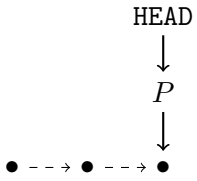
Git s'utilise en ligne de commande et demande une connexion Internet pour la synchronisation avec le dépôt distant.

Pour travailler localement sur un ordinateur, la commande `git clone git@gitlab.lif.univ-mrs.fr:aurele.dousselin/davis.git` a été lancée sur l'ordinateur dans un dossier qui comportera le projet, afin d'y créer une copie synchronisée du dépôt.

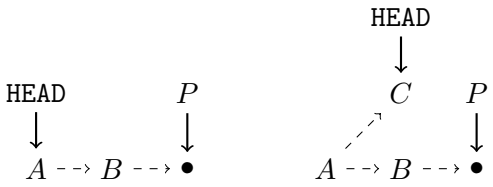
Après la modification d'un ou plusieurs fichiers, pour synchroniser cette modification avec le dépôt distant, il faut d'abord indiquer à Git les fichiers modifiés qui seront synchronisés, et cela via la commande `git add <file>`. Alors Git les ajoute à ce qu'on appelle l'index, d'où le *add*. Seules les modifications indexées seront par la suite synchronisées. Quand les différentes modifications à synchroniser ont été indexées, elles sont rassemblées en une seule "validation de modification", appelée *commit*, à laquelle on attache un message donnant une courte description des modifications effectuées. La commande pour faire cela est `git commit -m "<message>"`. Enfin, il suffit de pousser -*push*- les commits réalisés sur le dépôt distant, en faisant `git push`, pour que les modifications y soient enregistrées. La récupération des modifications poussées depuis d'autres clones du dépôt se fait grâce à la commande `git pull`.

Git garde en mémoire la suite de commits et utilise un pointeur P vers le commit le plus récent pour définir la version actuelle des fichiers.

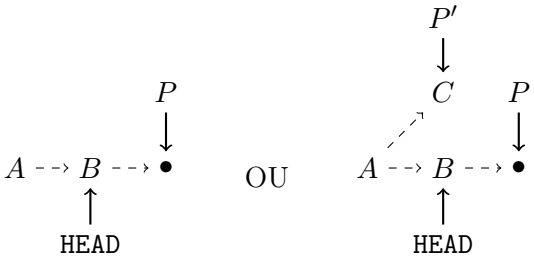
Lors de l'ajout d'un commit le pointeur se met à jour. Il existe un autre pointeur, appelé tête de lecture ou `HEAD` , et c'est en fait lui qui définit la version actuelle des fichiers, mais il pointe initialement sur P et le déplace avec lui.



Il est possible de faire pointer `HEAD` sur un commit précis A, en laissant P sur le commit le plus récent. Alors, si un nouveau commit C est créé, il se mettra à la suite de A, en parallèle avec le commit B qui était à la suite de A, et la tête de lecture avance sur C.



Si `HEAD` est de nouveau dirigé sur un commit de la "branche" principale, les commits créés en dehors de cette "branche" seront effacés, à moins d'avoir créé un nouveau pointeur vers le dernier d'entre eux.



Il est en effet possible de créer un autre pointeur P', qui pointera par défaut vers le même commit que `HEAD` . Il est aussi possible de faire pointer `HEAD` sur n'importe quel pointeur autre que lui-même, et dès que c'est le cas il déplacera avec lui ce pointeur lors de l'ajout de nouveaux commits. Les pointeurs autres que `HEAD` sont appelés des branches. Par souci de simplicité, les branches sont confondues avec la suite de commits menant au commit pointé.

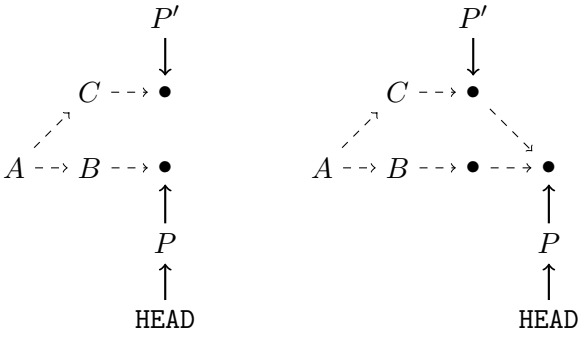
Pour bouger la tête de lecture sur un commit précis ou une branche, il suffit de lancer respectivement la commande `git checkout <commit>` ou `git checkout <branch>` où *<commit>* est le "nom" d'un commit et *<branch>* le nom d'une branche. Le nom d'un commit est son code attribué par Git (en fait la somme de contrôle SHA-1 de son contenu) ou alors les X premiers caractères de ce code, où X est tel qu'il n'y ait aucune ambiguïté avec d'autres commits : 6 ou 7 suffisent généralement. Une référence à un commit se fait toujours ainsi.

Pour créer une nouvelle branche, une commande est `git branch <branch_name>`, mais la commande `git checkout -b <branch_name>` est plus souvent utilisée car elle positionne HEAD sur la branch créée.

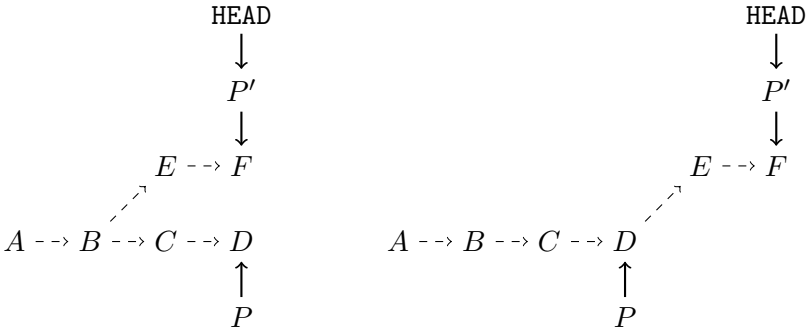
Pour supprimer une branche, il suffit de lancer `git branch -d <branch>`.

Git donne la possibilité de fusionner une branche P' dans une autre P : les modifications apportées dans les deux branches sont appliquées en même temps. Bien sûr, il peut y avoir

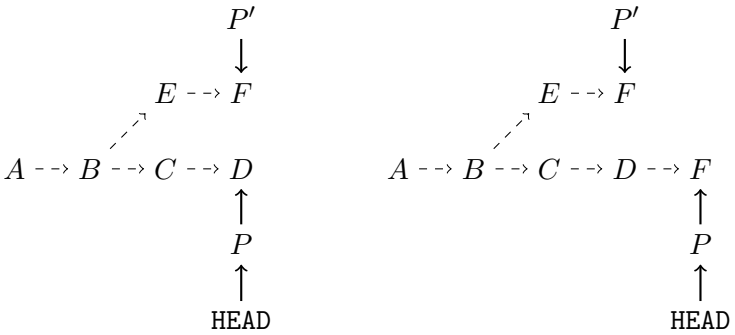
l'apparition de conflits, qui sont à résoudre à la main ou à l'aide d'outils de gestion de conflits. La fusion -*merge*- se fait par la commande `git merge <branch>`, alors la branche <branch> est fusionnée dans la branche actuelle. Le résultat d'une fusion est un nouveau commit dans la continuité de la branche actuelle P.



Une autre fonctionnalité de Git est la possibilité de rebaser une branche P' sur une autre P : P' est déplacée après P et tous ses commits sont ré-appliqués un par un après ceux de P. La commande pour faire cela est `git rebase <branch>`.



Git permet aussi d'appliquer un commit précis F à une branche P, grâce à la commande `git cherry-pick <commit>`



De plus, un commit B peut être désigné comme une correction d'un autre commit A. Pour cela il suffit que le message de B soit "fixup ! <message de A>". Une commande permet de faire cela automatiquement : `git commit --fixup <commit>`. Alors, lorsqu'un `git rebase --autosquash <branch>` sera effectué, B sera, lors de la ré-application des commits, fusionné avec A pour ne faire qu'un seul commit.