Présentation de Git

I – Présentation

Git est un outil de gestion de version de documents distribué, gratuit et open source. Il est disponible sous Windows, UNIX/Linux et MacOS. Il utilise un système d'arborescence permettant de passer efficacement d'une version donnée de vos documents à une autre, et offre la possibilité de revenir à des versions antérieures de votre travail. Bien que cet outil dispose d'un très grand nombre d'options et de configurations possibles, seul un petit nombre d'entre elles suffisent pour l'utiliser efficacement.

Git peut être utilisé intégralement en local sur sa machine, sans jamais être connecté à internet. Cependant, il est souvent utilisé en lien avec des services web tels que Github ou Gitlab qui permettent d'héberger vos documents en ligne pour les partager, et apportent souvent davantage de fonctionnalités. Il est ainsi possible de partager son travail avec d'autres personnes et de travailler de concert sur un même projet grâce à sa connexion *peer to peer* (pair à pair).

Git est entièrement utilisable à partir d'une console de commande, il est même fourni avec une console BASH à l'installation. De plus, de nombreux IDE tels que Visual Studio ou Eclipse supportent l'outil en intégrant directement ses commandes aux actions possibles, ce qui permet de bénéficier d'une approche graphique plus intuitive.

II – Procédure d'installation

Pour installer Git, il suffit de se rendre sur la page https://git-scm.com/downloads et de télécharger la version correspondant à votre système d'exploitation. Cependant, git est aujourd'hui installé par défaut sur la plupart des systèmes d'exploitation Linux et MacOS. Il est également possible de directement installer git par ligne de commande, ou alors de télécharger les versions antérieures de l'outil disponibles sur le dépôt Github officiel https://github.com/git/git.

III – Fonctionnalités et guide d'utilisation

Il existe 2 façons de commencer à utiliser git pour un projet.

La première est d'utiliser la commande ***git init*** dans le dossier dans lequel vous souhaitez travailler. Cette commande permet de désigner le dossier courant comme étant un dossier git, et donc dans lequel vous souhaitez utiliser l'outil. Aucune modification n'est apportée aux fichiers déjà présents, vous pouvez donc utiliser la commande dans un dossier contenant déjà du travail.

La deuxième est d'utiliser la commande ***git clone*** pour dupliquer tous els éléments d'un dossier existant, en faisant suivre la commande de l'emplacement du répertoire (pour un fichier local) ou de l'url du dépôt (pour un dossier hébergé en ligne).

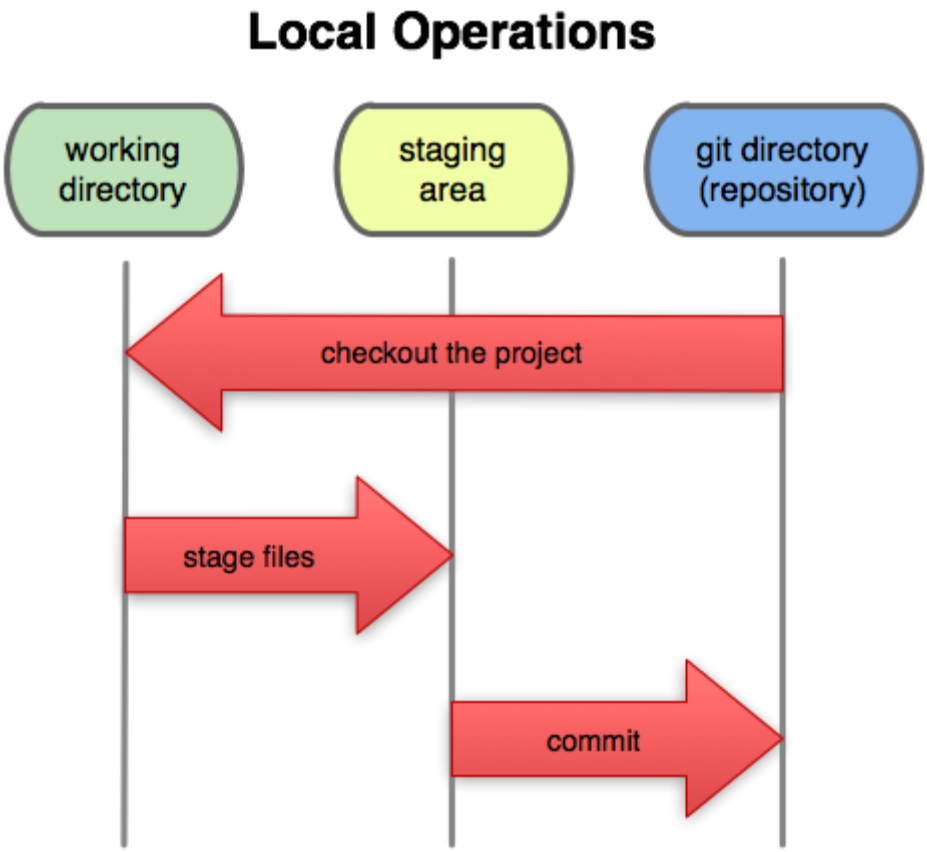
Pour ces deux commandes, il est également possible d'indiquer le dossier dans lequel vous souhaiter l'exécuter à la place à la fin de la commande.

```
nathan@LAPTOP-FL5QMEC6:~/exempleGit$ git init exempleInit/
Initialized empty Git repository in /home/nathan/exempleGit/exempleInit/.git/
nathan@LAPTOP-FL5QMEC6:~/exempleGit$ git clone ../GameJam_12-03-22/gamejam_12_03_2022/ exempleCloneLocal/
Cloning into 'exempleCloneLocal'...
done.
nathan@LAPTOP-FL5QMEC6:~/exempleGit$ git clone git@gitlab.isima.fr:nadesnos/gamejam_12_03_2022.git exempleCloneDistant/
Cloning into 'exempleCloneDistant'...
```

Exemple d'initialisation d'un répertoire git

Une fois l'une de ces commandes utilisées, vous trouverez dans le répertoire concerné un dossier ***.git*** contenant toutes les informations nécessaires à Git pour fonctionner. Il contient notamment un fichier ***.gitconfig*** qui comme son nom l'indique répertorie l'ensemble des configurations du répertoire. Ce fichier est généralement modifié par certaines options ou l'utilisation de certaines commandes, mais il peut être édité directement au besoin.

Une fois que vous travaillez dans un répertoire Git, il vous est possible d'enregistrer votre travail à l'aide de plusieurs commandes. Git fonctionne à l'aide de différentes « régions » stockant les informations sur vos fichiers, comme représenté sur le schéma page suivante, le tout étant invisible pour l'utilisateur.



Les différentes « régions » d'un répertoire Git

Le répertoire courant contient tous les fichiers présents dans le répertoire. Une fois que vous souhaitez enregistrer les modifications apportées à certains fichiers, il suffit d'utiliser la commande ***git add*** suivie des fichiers concernés pour que git ajoute les modifications effectuées à la zone de « staging ». Cette zone permet de contrôler quels fichiers sont concernés par la sauvegarde que vous souhaitez faire, modifier un fichier par inadvertance (ou pas) alors qu'il ne devrait pas être modifié dans la version de votre travail que vous souhaitez enregistrer n'est donc pas un problème. Si toutefois vous ajoutez un mauvais fichier dans la zone de staging, il est possible de l'en retirer avec la commande ***git reset***, suivie du nom du fichier. De plus, si vous apportez davantage de modifications à un fichier en staging, alors vous devrez à nouveau utiliser la commande ***git add*** pour ajouter ces modifications.

Une fois que vous êtes prêt à sauvegarder votre travail, il suffit d'utiliser ***git commit -m*** suivi d'un commentaire entre guillemets permettant de mieux identifier la version de votre travail. Les changements présents dans la zone staging seront alors enregistrées dans le dossier git.

Ces explications décrivent un cycle classique d'utilisation de git. Diverses commandes permettent ensuite de manipuler efficacement l'outil, dont entre autres :

- ***Git status*** permet d'obtenir l'ensemble des informations sur la présence de fichiers modifiés en zone de staging ou non, et donne d'autres informations sur la configuration du répertoire selon les cas.
- ***Git log*** permet d'accéder à une liste descendante avec brève description des différents commits.
- ***Git reset*** permet de récupérer la dernière version du travail enregistrée et efface toutes les modifications effectuées depuis. Il est possible de récupérer une version antérieure en rajoutant ***HEAD~(x)*** où (x) représente le nombre de version sur lesquelles vous souhaitez revenir (par exemple, HEAD~3 permet de retrouver la version du travail avant les 3 derniers commits).

Toutes ces commandes disposent de plus de multiples options, trop nombreuses pour être détaillées ici.

Il est également possible de développer plusieurs « lignes temporelles » de votre travail à l'aide du système d'arborescence de Git. Par défaut, vous travaillez sur une branche unique nommée « main ». il vous suffit d'utiliser la commande ***git branch*** suivi d'un nom évocateur pour créer une nouvelle branche partant de votre branche de travail. Vous pouvez ensuite utiliser ***git checkout*** suivi du nom d'une branche pour vous déplacer sur la branche concernée.

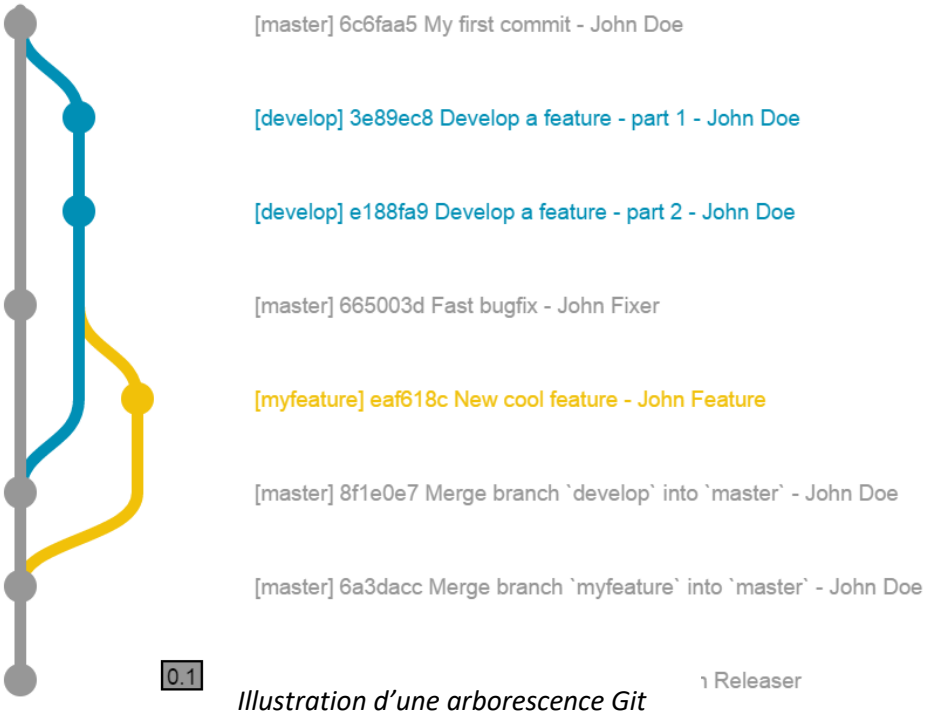


Illustration d'une arborescence Git

Cette fonctionnalité vous permet d'effectuer diverses expérimentations sans risque, et est très utile lors du travail de groupe. Une fois que vous êtes satisfait de votre expérimentation, la commande ***git merge*** suivi d'un nom de branche permet de fusionner les commits de la branche nommée sur votre branche actuelle. ***git branch*** sans nom associé vous permet de voir la liste des branches existantes ainsi que la branche sur laquelle vous vous trouvez en cas de doute. Enfin, ***git delete*** suivi d'un nom de branche permet de la supprimer lorsque vous n'en avez plus besoin.

Enfin, git permet de transférer et récupérer les informations de votre branche locale vers un dépôt distant. Les dépôts git en ligne fonctionnent exactement de la même manière qu'en local.

Pour utiliser cette fonctionnalité, il faut tout d'abord lier votre répertoire local à un dépôt distant, soit à l'aide de la commande ***git clone*** évoquée précédemment, soit à l'aide de ***git add*** accompagné du lien correspondant (http, ssh, etc…). Un ***git push*** vous permettra ensuite d'envoyer les commits de votre branche locale vers la branche du dépôt distant auquel elle est liée (généralement « main » par défaut). La commande ***git pull*** permet de faire l'opération inverse en récupérant les modifications enregistrées sur la branche distante liée. Il existe de nombreuses façons d'utiliser ces commandes, notamment pour choisir de quelle branche récupérer les changements ou vers quelle branche les envoyer. Toutes ces commandes permettent la mise en place d'un travail collaboratif et d'un partage de l'avancement.

IV – Synthèse des points clés

En résumé, Git permet de gérer efficacement les différentes versions de vos fichiers sans avoir à effectuer de copies de ces derniers pour en garder une trace. Il est possible d'expérimenter sur plusieurs branches parallèles, de revenir à des versions antérieures de votre travail et de partager votre avancée avec vos collaborateurs de façon très ordonnée.

L'ensemble des possibilités offertes par l'outil et les bonnes pratiques à adopter ne sauraient bien sûr se limiter à cette courte présentation. Si besoin, il est possible de trouver davantage de détail sur la page officielle (en anglais) https://git-scm.com/.