Documentation : Utilisation de Git :

Sommaire :

- 1) Qu'est ce que Git ?
  - 1.1 : Utilité de Git
  - 1.2 : Un peu d'histoire
- 2) Utilisation de Git
  - 2.1 : Utilisation de Git avec Github
    - 2.2.1 : Initialisation avec Github
    - 2.2.2 : Envoyer les modifications vers le serveur distant
  - 2.2 : Utilisation (un peu plus avancée) de Git
    - 2.2.1 : Description des branches
      - 2.2.1.1 : Description des branches
      - 2.2.1.2 : Créez des branches
      - 2.2.1.3 : Fusionnez des branches
      - 2.2.1.4 : Supprimez des branches
    - 2.2.2 : Évitez les commits inutiles
  - 2.3 : Configurations de Git (avec ou sans Github)
  - 2.4 : Gestion d'erreurs
    - 2.4.1 : Problème de fusion

---

1 : Qu'est ce que Git ?

1.1 : Utilité de Git

Git est un logiciel de versionning (gestion de versions). Ce type de logiciel permet de stocker un ensemble de fichiers en conservant la chronologie de toutes les modifications qui ont été effectuées dessus, permettant de retrouver toutes les versions d'un projet.

Chaque projet est hébergé sur l'ordinateur du propriétaire du projet, sur ceux de ses collaborateurs (ainsi que certaines branches du projet), puis sur un serveur distant permettant la sauvegarde du code en cas de problème et pour faciliter le travail en collaboration.

Ces serveurs distants peuvent être ceux de sites proposant des interfaces web (Github, Gitlab, par exemple), voire son propre serveur (il est possible de stocker et de gérer un ou plusieurs dépôts sur un serveur personnel).

1.2 : Un peu d'histoire

Initialement, le projet du noyau Linux était hébergé sur des serveurs utilisant Bitkeeper. La communauté libriste, ne voyant pas cela d'un bon œil (Bitkeeper est un logiciel **propriétaire**), a décidé de créer un système de gestion de version libre le jour où la version gratuite de Bitkeeper a cessé d'être distribuée.

Le 7 avril 2005 sort la première version de Git, avant de se voir doté d'interfaces graphiques, d'interfaces web (Github, Gitlab, BitBucket, etc…), de scripts évolués, etc…

Fin 2005 sort la version stable 1.0, moins d'un an après le début du projet. La version 2.0 de Git, quant à elle, sort en 2014.

2 : Utilisation de Git

2.1 : Utilisation basique de Git

2.1.1 : Initialisation avec Github

Nous allons voir comment utiliser Git en interagissant avec l'utilitaire web Github (dans le cas où vous possédez un compte Github).

Pour commencer, connectez-vous à votre compte Github, puis en haut à gauche, cliquez sur le bouton vert **« New »**. Donnez un nom à votre projet et, si vous le souhaitez, ajoutez une description. Vous pouvez choisir qui a la permission de voir votre dépôt Github, ainsi qu'initialiser le dépôt en lui ajoutant un fichier README.

Une fois le dépôt créé, vous pouvez le cloner sur votre ordinateur. Pour cela, ouvrez un terminal, naviguez jusque dans le dossier où vous souhaitez stocker le dossier du projet, puis tapez la commande :

- **git clone** *https://github.com/votre_nom_de_profil/le_lien_de_la_page_principale_du_projet*

Naviguez avec votre terminal vers le dossier du repo fraîchement créé, puis commencez à travailler.

Un dossier caché nommé **.git** a été créé dans le dossier cloné, il contient de nombreux fichiers utiles, dont un fichier de configuration nommé **« config »**.

Si vous souhaitez voir quels fichiers ont été modifié en cours de travail ou avant de commit, pensez à la commande **git status**, elle vous listera les fichiers nouvellement créés, modifiés et supprimés.

Une fois le dépôt cloné (s'il n'a jamais été cloné auparavant sur votre système ou si le fichier de configuration a été modifié ou réinitialisé), tapez les commandes suivantes :

- **git config user.email "votre adresse mail utilisée sur Github ou n'importe quel utilitaire web"** pour vous identifier avec votre adresse mail.
- **Git config user.name "Prénom Nom"** pour vous identifier avec votre nom/prénom.
- Ces deux commandes sont **obligatoires**.

2.1.2 : Envoyer les modifications vers le serveur distant

Pour ajouter TOUTES vos modifications, dans la racine du projet, commencez par taper la commande **git add *.

Note 1 : Vous pouvez également remplacer l'étoile par un point, ou par l'option **« --all »**.

Note 2 : Si vous tapez les deux premières commandes dans un sous-dossier du projet, les fichiers des projets parents ou voisins ne seront pas ajoutés au commit.

Puis tapez la commande **git commit -m "*Tapez un message pour retrouver la version du commit*"** pour taper un message vous permettant de retrouver cette version du projet en cas de besoin.

Une fois ceci fait, vous n'avez plus qu'à *« push »* votre travail, c'est à dire l'envoyer sur le serveur distant en utilisant simplement la commande **git push**.

Si vous souhaitez récupérer les nouvelles modifications sur un autre ordinateur (machine physique ou machine virtuelle), tapez la commande **git pull**.

2.2 : Utilisation (un peu plus) avancée de Git

Comme dit précédemment, vous n'avez vu que le sommet de la partie visible de l'iceberg de Git

2.2.1 : Gestion des branches

2.2.1.1 : Description des branches

Git, comme tout logiciel de versionning, permet de travailler à plusieurs sur un même projet, cela peut aller d'une petite équipe de développeurs travaillant sur un petit programme jusqu'aux milliers de développeurs apportant leur contribution au développement du noyau Linux.

Plus les collaborateurs d'un projet sont nombreux, plus les contributions simultanées à une ou plusieurs parties d'un projet peuvent être différentes, jusqu'à causer des bugs ou d'autres situations indésirables sur d'autres parties du code dépendant de ces parties à modifier.

Pour éviter de modifier le dépôt, Git permet de créer des **branches**, permettant ainsi de séparer le code à modifier du code du projet.

Cela vaut également pour les nouvelles fonctionnalités d'un projet quand elles en sont à un stade expérimental, cela permet de les tester avant de modifier profondément le code du projet principal, dont il peut être fastidieux de retirer de nombreuses lignes de code d'une fonctionnalité finalement non-désirée tout en gardant celles d'autres fonctionnalités ajoutées entre temps.

Quand un projet est créé (via un gestionnaire web ou en ligne de commande), une branche est automatiquement créée et sert de branche de base au projet. Cette dernière est toujours appelée **master**.

Seul le propriétaire du dépôt Git peut accepter l'ajout du code d'une branche sur le code de la branche principale du projet (nommée **master**).

2.2.1.2 : Créez des branches

Pour créer une nouvelle branche, utilisez la commande **git branch nouvelle_branche**, où « nouvelle_branche » est le nom de votre branche à créer.

Une fois la branche créée, vous pouvez y basculer en tapant la commande **git checkout nouvelle_branche**, vous pouvez vous assurer à tout moment d'être bien passé sur la nouvelle branche en tapant la commande **git branch** (sans nom de branche), toutes les branches existantes sont listées, dont la branche sur laquelle vous vous trouvez actuellement, qui est marquée par un astérisque **« * »** et dont le nom est coloré différemment que celui des autres branches.

2.2.1.3 : Fusionnez des branches

Une fois une nouvelle fonctionnalité implémentée avec succès, le code de la branche **master** peut être fusionné avec celui de la branche implémentant la nouvelle fonctionnalité. Pour cela, la commande **git merge** fait tout le travail.

Pour que cette commande fasse son travail efficacement, il faut d'abord se placer dans la branche contenant l'ancien code (**git checkout ancienne_branche**), puis taper la commande **git merge nouvelle_branche**.

Il arrive souvent que des problèmes de fusion aient lieu, comme lorsqu'un fichier a été modifié. Ce problème est traité dans la partie 2.4.1.

2.2.1.4 : Supprimez des branches

Pour supprimer des branches, appelez la commande **git branch**, suivie de l'option **-d** (delete) en **minuscule**. Si elle n'a pas été fusionnée auparavant, une fonctionnalité de sécurité empêche sa suppression et affiche un message d'avertissement. Si vous êtes vraiment sûr de la supprimer, remplacez l'option **-d** par l'option **-D** (**EN MAJUSCULE**). ATTENTION, TOUT LE TRAVAIL EFFECTUÉ SUR UNE BRANCHE À SUPPRIMER SERA **DÉFINITIVEMENT** PERDU SI AUCUNE FUSION N'A ÉTÉ FAITE.

2.2.2 : Évitez les commits inutiles

Lorsque vous devez sauvegarder le code d'une branche avant de travailler sur une autre branche, vous serez sans doute tenté de faire un commit pour sauvegarder, au risque d'encombrer l'historique des modifications.

Une commande a été créée pour éviter que cet encombrement arrive, il s'agit de **git stash**. Cette commande **« met de côté »** les modifications que vous avez apporté sur la branche.

Vous pouvez lister les modifications mises de côté en tapant la commande **git stash show**.

Une fois que votre travail est terminé, vous pouvez récupérer votre travail mis de côté en tapant (tout en étant sur la branche dont le travail a été mis de côté) la commande **git stash pop**, qui va vider le stash de modifications, ce qui veut dire que vous devrez soit faire un commit, soit retaper **git stash**. Pour éviter cela, vous pouvez remplacer la commande **pop** par la commande **apply**.

Vous pouvez supprimer tout le contenu d'un stash sans le récupérer en utilisant la commande **clear**. Attention, il sera impossible de récupérer les modifications mises de côté.

2.3 : Configurations de Git (avec ou sans Github)

Vous savez déjà comment utiliser Git de manière (extrêmement) basique. Cependant, même pour ce genre d'utilisation, vous aurez parfois besoin de configurer votre dépôt.

La commande **« git config »** permet de configurer un dépôt Git avec de plusieurs options d'emplacement du fichier de configuration **« config »** :

git config -f, --file chemin_vers_le_fichier

- L'option –**file** permet de modifier/utiliser les options de Git pour un fichier de configuration dont son chemin est spécifié en argument.

git config –global

- L'option –**global** permet de modifier/utiliser les options de Git pour tous les dépôts sur un ordinateur.

git config --local

- L'option –**global** permet de modifier/utiliser les options de Git du dépôt local.

git config –system

- L'option –**global** permet de modifier/utiliser les options de Git de chaque compte utilisateur sur l'ordinateur, ainsi que chacun de leurs dépôts.

Ces options sont suivies d'autres options :

--add nom_variable valeur_variable

- Cette option permet de rajouter une variable dans le fichier de configuration.

- Elle attend en argument le nom de la variable, puis son type (**bool**, **int**, **path**, **bool-or-int**, **expiry-date**).

-e, --edit

- Cette option permet d'ouvrir un éditeur de texte pour éditer le fichier de configuration selon son emplacement.

-l, --list

- Cette option permet de lister les variables du fichier de configuration.

--unset expression_régulière

- Cette option permet de supprimer une variable selon la chaîne de caractères passée en argument

--unset-all expression_régulière

- Cette option permet de supprimer toutes les variables selon la chaîne de caractères passée en argument.

2.4 : Gestion d'erreurs

Comme tout logiciel, Git peut parfois rencontrer des problèmes résultant d'une mauvaise utilisation. Cette section est là pour vous aider à corriger ces problèmes.

2.4.1 : Problème de fusion

Ce type de problème arrive fréquemment quand plusieurs personnes travaillent sur un même fichier, même quand une seule personne travaille sur un projet, car le contenu de ce fichier est différent.

Pour y remédier, vous devrez modifier chaque ligne de code différente, avant de pouvoir finalement commit.

Maintenant que le conflit est résolu, il vous reste à le signaler à Git. Pour l'instant, si vous tapez la commande **git status**, Git vous dira que vous avez des branches non fusionnées (**unmerged paths**). Pour cela, faites un commit sans message en tapant simplement la commande **git commit**.