Gestionnaire de versions

Un gestionnaire de versions permet de gérer des versions lors du développement et de la vie d'un logiciel.

Ceux-ci ont évolué au fil du temps, et les services qu'ils fournissent permettent aisément de gérer tout type de contenu susceptible d'évoluer : documentation, fichiers graphiques…

Grâce à un gestionnaire de versions vous pourrez :

- Sauvegarder votre code à distance
- Gérer les différentes versions de votre logiciel
- Revenir à une version précédente
- Comparer à une version précédente
- Travailler en commun sur votre logiciel
- Gérer les conflits potentiels de code
- Mettre à jour des dépendances logicielles
- Déclencher des tests automatisés
- Et bien d'autres choses.

Si certaines de ces actions sont liées à l'activation de fonctionnalités supplémentaires, la majorité des gestionnaires existants vous les permettent très aisément.

Parmi les gestionnaires les plus connus, on peut en distinguer trois :

SVN

Autrefois le plus utilisé, ce gestionnaire est tombé en désuétude, car offrant peu de fonctionnalités quant au travail en commun.

Mercurial

Ce gestionnaire plutôt complet n'a pas acquis une grande popularité, et est du coup peu utilisé.

Git

Ce gestionnaire complet est très populaire, et fortement utilisé, même pour des besoins en réseau, en graphisme ou encore sur des rendus de projet. Il s'avère incontournable en entreprise en 2023.

Afin d'héberger votre code et ses versions, il existe plusieurs hébergeurs, par exemple :

Github

Le plus connu, possédé et hébergé par Microsoft.

Gitlab

Permet d'héberger soi-même son code sur son propre serveur, avec divers plugins possibles.

Vocabulaire

Repository ou dépôt

C'est l'endroit où le code se trouve, avec ses versions.

Clone

Un clone d'un dépôt est une copie locale permettant de travailler sur le code. C'est la première étape pour pouvoir travailler lorsque le dépôt est créé, afin de pouvoir travailler dessus.

Commit

C'est l'action de publier une version. Cette publication reste locale, sur le clone courant.

Push

C'est l'action d'envoyer sur le serveur distant la liste des versions publiées, afin de les enregistrer. Il est primordial de très régulièrement push son travail.

Pull

C'est l'action de demander la récupération et l'intégration des versions publiées sur le serveur vers notre clone local. C'est la récupération des données.

Merge

C'est l'action de fusionner deux versions.

Branch

Une branche de développement est un moyen de développer de manière parallèle aux autres développements. C'est souvent utilisé pour les gros développements, ceux qui prennent du temps, ou dans le cas de développement avec d'autres personnes.

Fork

C'est l'action de copier un dépôt « public » afin d'en obtenir une copie personnelle. Cela permet de contribuer à des projets open source.

Merge Request / Pull Request

C'est la demande d'intégration du code d'une branche dans une autre branche, par exemple pour ajouter la fonctionnalité que vous avez développé dans le programme principal.

Review

C'est l'action de vérifier le code produit sur une branche, afin de fournir soit des corrections soit des propositions de correction.

Conflit

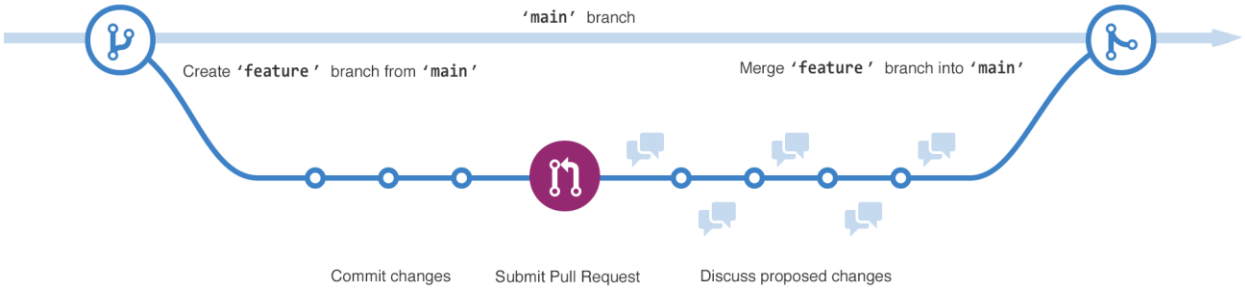
Lorsque du code différent est apporté sur un même fichier. Un conflit doit être résolu humainement.

Flux de développement usuel (branche)

Le processus de développement et de gestion de versions est propre à chaque équipe.

Mais un processus habituel serait :

- Le développement commence, avec la spécification des fonctionnalités voulues
- Le développeur crée une nouvelle **branche**
- Le développeur va régulièrement **ajouter** des modifications, créer des **commits** et **push** régulièrement ces commits
- Une fois que la fonctionnalité semble prête, le développeur crée une **Pull Request**
- Un autre développeur effectue une **review** du code
- Une fois la review terminée et validée, la pull request est validée (ou annulée) et le code est **mergé**
- La branche étant devenue inutile, on la supprime



Ce processus peut être différent ; c'est à votre équipe de le définir. En entreprise, il est souvent imposé, ou à définir par l'équipe de développement.

Les noms des branches sont des éléments importants pour se repérer. « feature-cart » pour indiquer l'ajout d'un panier, par exemple, est un bon nom de branche.

Une pull request liste tous les commits de la branche. Même si elle est déjà ouverte, tous les commits futurs de cette branche vont être visibles dans cette pull request ; il est donc possible de l'ouvrir dès le début du développement, à la création de la branche.

Il est primordial de push et pull régulièrement, afin d'éviter les conflits au maximum et de sauvegarder son code très fréquemment.

Main vs Master

A la suite d'une vague de prise de position en lien avec les connotations du mot « master », git d'abord puis github, ont choisi de renommer la branche par défaut « main ». Certains logiciels peu mis à jour n'ont pas réalisé la bascule. Pour déclarer à git le nom de la branche par défaut à la création (init) d'un dépôt :

```
git config --global init.defaultBranch <branch>
```

Qui êtes-vous

Github gère différemment l'authentification de l'identification.

Authentification

C'est ce qui vous accorde des autorisations, et vous permet de pull/push en fonction de vos droits, dans différents dépôts. Cette authentification est à charge de votre système d'exploitation. Sur les systèmes X, c'est généralement géré de la même manière que SSH. Sur Windows, c'est le « gestionnaire d'identification » (sic) qui retient le lien entre le compte courant et l'authentification Github.

Lorsque vous effectuez une opération sur Github qui nécessite des droits particuliers (pull d'un dépôt privé, ou push par exemple), le système vous demande vos informations d'authentification : votre accès Github.

Identification

C'est ce qui permet de dire à Git qui commit réellement. C'est une configuration Git, qui peut être au niveau du dépôt ou au niveau du système, et qui se compose d'au moins deux informations : votre nom et votre email.

Si vous effectuez un commit sans que votre poste n'ai une quelconque idée de qui vous êtes, vous aurez un message d'avertissement sur ce sujet.

Pour configurer globalement, sur votre poste, votre identité :

```
git config --global user.name <nom>
git config --global user.email <email>
```

Conflit

Un conflit peut apparaître dès que deux versions sont conflictuelles, c'est-à-dire qu'au moins un fichier présente des modifications différentes sur une même branche.

Cela peut arriver :

- Lors d'une pull request
- Lors d'un pull en local

La résolution du conflit est obligatoire pour continuer à travailler. Un conflit est toujours plus simple à résoudre en communiquant avec son équipe pour prendre une décision.

Il est donc plus aisé de gérer les conflits durant les Pull Request. Github propose notemment une interface pour cela, au lieu de l'interface par ligne de commande. Certains IDE vous propose aussi une interface pour cela.

Si un conflit n'est pas résolu correctement, il va se répercuter sur les copies locales de tout développeur travaillant sur le projet, et possiblement créer encore plus de conflits.

Commandes git usuelles

Cloner un dépôt

`git clone <uri>`
Clone le dépôt situé à l'adresse <uri>

Créer une branche

`git branch <name>`
Crée la branche <name>.

Attention : la branche est créée, mais vous n'êtes pas automatiquement basculé dessus.

Basculer sur une branche

`git checkout <branch>`
Bascule sur la branche <branch>

Astuce : l'argument « -b » vous permet de créer la branche si elle n'existe pas.

Donc `git checkout -b <branch>` vous permet de créer et de basculer sur une nouvelle branche.

Ajouter une modification au prochain commit

`git add <fichier>`
Ajoute le fichier (ou dossier) <fichier> pour le prochain commit

Créer un commit

`git commit -m "<message>"`
Crée un commit avec le message « <message> »

Envoyer tous les commits au serveur

`git push`
Envoie la liste des commits en attente vers le serveur

Récupérer tous les commits du serveur et les appliquer

`git pull`
Récupère tous les commits depuis le serveur pour mettre à jour la copie locale

Voir le statut de la copie locale

`git status`
Donne des informations de statut de votre copie locale : si des commits sont en attente, si une fusion est en cours, ou encore si des fichiers ne sont pas encore dans un commit.

Voir l'arborescence des branches

`git log --all --decorate --oneline --graph`
Affiche une arborescence visuelle des branches existantes, et des commits effectués.

Git Cheat Sheet

https://education.github.com/git-cheat-sheet-education.pdf