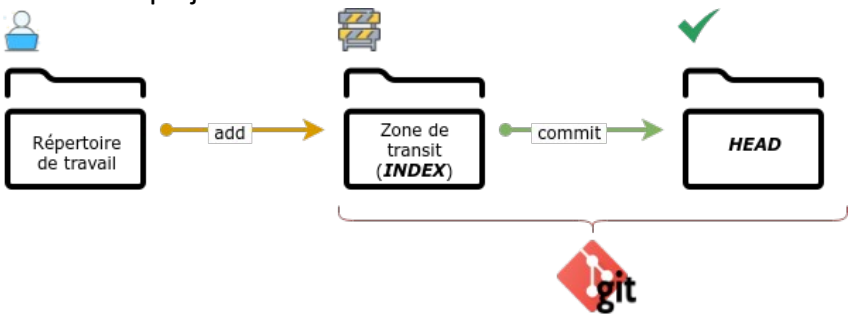
Antisèche Git

Cette antisèche ne peut pas se substituer à une bonne formation, mais elle peut t'aider au quotidien quand tu as oublié une commande.
C'est pour cela que nous n'abordons pas l'installation et la configuration de git, qui n'est pas effectuée tous les jours, dans cette antisèche.
Nous prendrons comme convention que les commandes entre crochets `[command]` sont des valeurs à remplacer
et celles entre les signes inférieur et supérieur `<command>`, des valeurs optionnelles.

- **Dépôt :** (🇬🇧 *repository* ou *remote*) c'est le lieu où sont stockés tous les fichiers sources de ton projet. Le dépôt peut être un répertoire local sur ta machine ou un répertoire distant sur un serveur git. Le dépôt distant peut être accessible en https ou en ssh. Utilise plutôt une connexion **ssh** (avec clé SSH personnelle) qui est plus sécurisée et cela t'évite aussi de créer un fichier avec tes informations d'authentifications.

- **Zone de transit :** (🇬🇧 *staging area* ou *Index*) cette zone mémoire contient les fichiers que tu mets de côté pour préparer le commit. On dit que les fichiers sont **indexés**.

- **Commit :** c'est une enveloppe qui contient une petite portion de codes modifiés d'un ou plusieurs fichiers. Fais des commits les plus petits possible. Tu peux comparer les commits entre eux et voir les évolutions du code de ton projet. On utilise le verbe **commiter**.



- **Tag :** c'est un pointeur vers un commit particulier. On s'en sert pour faire des *releases*.

- **Branches :** (🇬🇧 *branch*) une branche c'est une version du dépôt où tu peux travailler sur une fonctionnalité particulière sans impacter le code courant. Il y a toujours une branche par défaut, c'est souvent *master*, et tu peux avoir autant de branches en cours que tu veux.

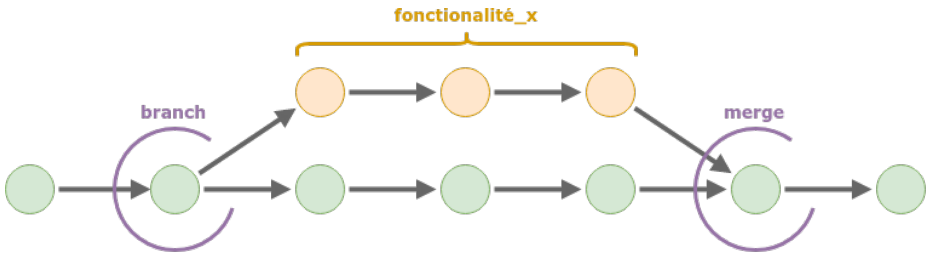
- **HEAD :** c'est un pointeur vers le dernier commit de la branche en cours.

- **Pousser :** (🇬🇧 *push*) cette action partage les commits du dépôt local avec le dépôt distant associé au projet en cours.

- **Rapporter :** (🇬🇧 *fetch*) cette action récupère les commits du dépôt distant en local sans les appliquer.

- **Tirer :** (🇬🇧 *pull*) cette action télécharge les commits manquant du dépôt distant sur notre dépôt local et les applique.

- **Fusion :** (🇬🇧 *merge*) cette action applique les changements d'une branche sur une autre. Sur un serveur git, tu peux faire une demande de fusion qui peut être revue par tes collègues avant la fusion réelle. Sur Github, c'est une *Pull Request* (PR) et sur GitLab une *Merge Request (MR)*.



Auteur : Chaudier Christophe

Au secours !

Toutes les commandes git possèdent une aide en ligne disponible en tapant :

`git [command] -h` ou `git command -help` pour la version courte.

`git [command] --help` pour la version longue.

C'est parti !

Tu peux commencer un projet soit par :

`git init` transforme le répertoire courant en nouveau dépôt local .

`git init [projet]` crée nouveau dépôt local dans le sous-répertoire `projet`.

`git clone [url` fait une copie locale d'un projet depuis un dépôt distant.

On sauvegarde, c'est important

`git status` affiche tous les fichiers qui peuvent être commités (ajoutés, supprimés, renommés, modifiés).

`git diff` affiche toutes les modifications des fichiers qui ne sont pas encore indexés.

`git diff [fichier]` affiche toutes les modifications de `fichiers` qui ne sont pas encore indexés.

`git diff --staged` affiche les différences entre la version indexée et la dernière version.

`git add [fichier]` indexe le fichier nommé `fichier` pour préparer le commit.

`git commit` crée un commit avec tous les fichiers qui sont dans l'index.

`git commit -m "[description]"` la même chose, mais en mettant le message du commit directement dans la ligne de commande.

`git tag [mon-tag] <[commit]>` étiquette le commit, ou par défaut le dernier commit, avec le tag. Ce tag est souvent un numéro de version et peut servir à faire des *releases*. Pour standardiser tes numéros de version, regarde du côté du versionnage sémantique.

Auteur : Chaudier Christophe

Travaux en cours

`git branch`	affiche toutes les branches du dépôt local.
`git branch [nom-de-branche]`	crée une nouvelle branche.
`git branch -m [nom-de-branche] [nouveau-nom]`	change le nom de la branche.
`git checkout [nom-de-branche]`	bascule sur la branche demandée, par défaut c'est master.
`git merge [nom-de-branche]`	fusion dans la branche courante des commits de la branche.
`git rebase [nom-de-branche]`	change le commit de base de la branche courante depuis la branche. Mais alors on fait quoi, merge ou rebase ?
`git branch -d [nom-de-branche]`	supprime la branche.
`git rm [fichier]`	supprime le fichier du répertoire de travail et met à jour l'index.
`git rm --cached [fichier]`	supprime le fichier du système de suivi de version, mais le préserve localement.
`git mv [fichier-nom] [fichier-nouveau-nom]`	renomme le fichier et met à jour l'index.

Et pour avoir des informations ?

`git log`	affiche la liste des commits de la branche courante.
`git log --author=[auteur]`	affiche tous les commits de l'auteur.
`git log --oneline --decorate --graph –all`	affiche tout l'historique sous forme de graph. Les interfaces telles que Gitlab ou Github proposent également une visualisation de ce graphe.
`git log --follow [fichier]`	affiche la liste des commits du fichier.
`git diff [br1] [br2]`	affiche les différences entre deux branches.
`git show [commit]`	affiche les métadonnées et le contenu du commit, ou du **tag**.

Et si je veux mettre de côté ?

`git stash`	enregistre de manière temporaire tous les fichiers qui ont été modifiés. On appelle cela « remiser son travail ».
`git stash list`	affiche la liste de toutes les remises.
`git stash pop`	applique une remise sur le répertoire courant et la supprime immédiatement.
`git stash drop`	supprime la remise la plus récente.

Auteur : Chaudier Christophe

Mince, je me suis planté !

`git reset [fichier]` désindexe `fichier` de l'index tout en conservant son contenu.

`git reset [commit]` annule tous les commits après `commit`, en conservant les modifications.

`git reset --hard [commit]` ⚠ supprime tous les commits **et** les modifications effectuées **après** `commit`.

Je ne veux pas tout suivre ?

Si tu veux exclure des fichiers et des chemins temporaires, il faut que tu crées un fichier `.gitignore`, le mieux c'est de le mettre à la racine de ton dépôt. Les fichiers et les chemins listés dans le fichier seront ignorés par git.

```
*.log
build/
temp-*
```

Et si j'ai déjà commité ?

Corriger des erreurs et gérer l'historique des corrections.

`git reset [commit]` annule tous les commits après `[commit]`, en conservant les modifications localement.

`git ls-files --other --ignored --exclude-standard $ git reset --hard [commit]` liste tous les fichiers exclus du suivi de version dans ce projet.

Il faut bien partager

Référencer un dépôt distant et synchroniser l'historique de versions.

`git fetch [depot]` récupère tout l'historique du dépôt, sans modifier votre branche locale.

`git merge [depot]/[branche]` fusionne la branche du dépôt dans la branche locale

`git pull` récupère tout l'historique du dépôt et incorpore les modifications en local. Cela revient en fait à faire un `fetch`, puis un `merge` en une seule commande.

`git push [alias] [branche]` envoie tous les commits de la branche locale vers le serveur git. Souvent l'alias est `origin`.

`git push --tags` ⚠ envoie **tous** les tags locaux vers le serveur git distant.

`git push [alias] [mon-tag]` envoie `mon-tag` de la branche locale vers le serveur git. Souvent l'alias est `origin`.

Auteur : Chaudier Christophe