Git : Travail en groupe

Maintenant que vous savez gérer un projet sur git, vous allez pouvoir travailler en groupe. Pour ce cours vous pouvez donc travailler à deux.

LE TRAVAIL À DISTANCE

Jusqu'à maintenant nous avons vu comment travailler avec `git` en local (sur notre machine) pour ajouter des points de sauvegarde (`commits`) et ainsi garder une trace de chaque changement apporté au fil du temps à notre code.

De cette manière, en cas de besoin, il est possible de revenir en arrière sur une version précédente plus stable très facilement.

Grâce à `git` nous pouvons communiquer directement avec des machines situées à distance.

Ces machines que l'on nomme des `remote`, peuvent être de deux types :

- Interne : un autre ordinateur que vous possédez par exemple.
- Externe : un service tiers qui hébergera sur internet votre code comme `GitHub` , `GitLab`, etc.

Ainsi, en plus d'avoir une version sur notre ordinateur, nous aurons une version identique de notre projet sur un serveur. Cette méthode à de nombreux avantages, notamment :

- Permet une sauvegarde supplémentaire en cas de perte de donnée.
- Favorise et facilite le travail collaboratif.
- Pouvoir récupérer un projet de n'importe où.

GitLab

Gitlab, est une plateforme permettant d'héberger et de gérer des projets web. C'est un remote externe. Il va donc permettre de gérer les versions git dans le cadre du travail distant.

Vous pouvez y accéder depuis l'interface.

Mettre un projet sur GitLab

Pour mettre un projet sur GitLab, il faut d'abord initialiser son projet comme vue dans le cours précédent. Avec le terminal, il faut donc se placer dans le répertoire du projet que vous souhaitez mettre sous git, puis l'initialiser avec la commande :

```
git init
```

Ensuite, vous renseignez l'emplacement du projet sur le remote avec la commande :

```
git remote add origin URL_DU_PROJET
```

Enfin, vous envoyez votre projet Git sur le remote avec la commande :

```
git push
```

Récupérer un projet

On peut récupérer un projet hébergé sur un remote, on parle alors de **cloner un repo git**.

Pour cela, avec le terminal il faut se placer à l'endroit où l'on souhaite cloner le projet. Ensuite, on va récupérer un projet sur un serveur et en faire une copie en local. Il suffit de récupérer son adresse URL et d'entrer la commande :

```
git clone URL_DU_PROJET
```

À noter : Le clonage peut prendre quelques minutes avant de se terminer. Tout dépend la taille des fichiers présents dans le repo sur le remote.

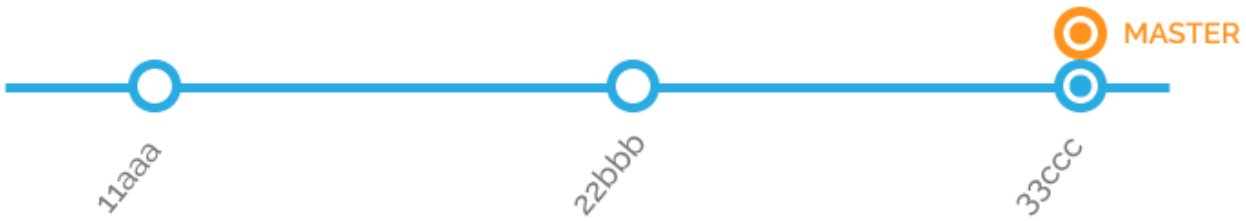
Une fois terminé vous avez maintenant une copie du projet sur votre ordinateur en local.

LE SYSTÈME DE BRANCHES

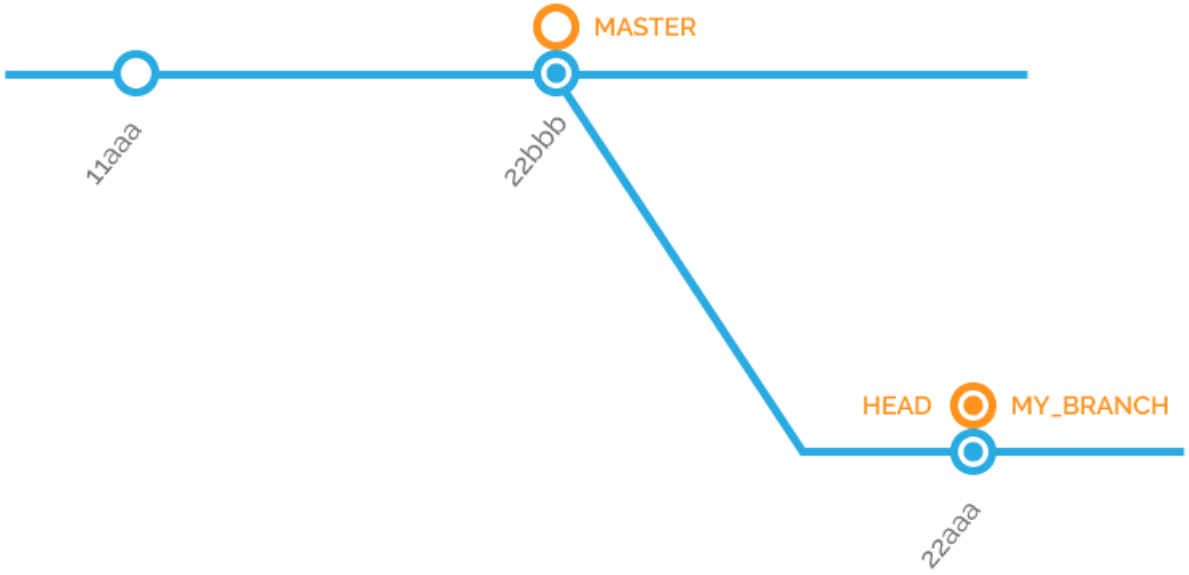
Les branches sont généralement au cœur des systèmes de versionning. Elles permettent de travailler à plusieurs en parallèle sur le même projet, sans se marcher sur les pieds.

Par défaut, la branche principale sur laquelle vous vous trouvez au début s'appelle master.

Actuellement, nous avons donc travaillé sur notre branche master, que l'on peut représenter comme ceci :



La branche master doit toujours être fonctionnelle. On va donc faire nos développements sur une autre branche. Créer une autre branche, c'est comme travailler sur une copie de la branche master, on aurait donc ceci :



Une fois seulement qu'on est sûrs que nos développements fonctionnent correctement, on passe alors nos modifications sur la branche master. Concrètement c'est cette version qui va être mise en « production », c'est à dire la version qui sera en ligne sur internet.

Créer une branche

Depuis votre terminal, on peut vérifier toutes les branches de notre projet avec la commande :

```
git branch
```

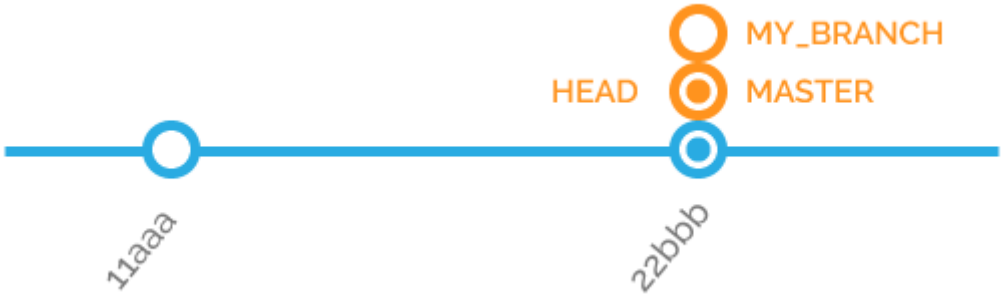
Pour créer une nouvelle branche on va utiliser cette même commande en ajoutant le nom de la branche que l'on veut créer. Par exemple :

```
git branch ma-branche-test
```

À ce stade si on affiche les branches de notre projet on aura :

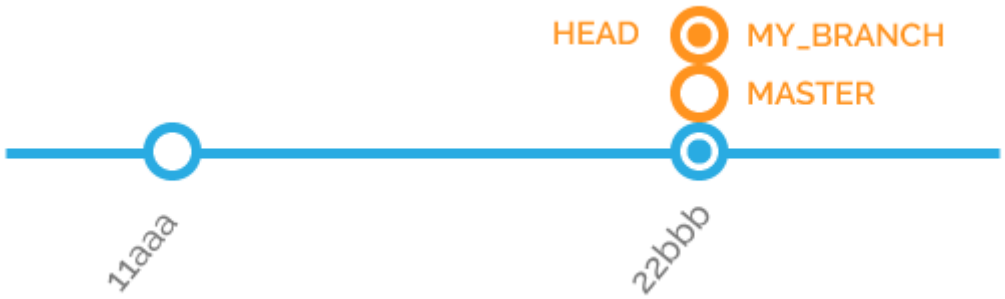
```
  ma-branche-test
* master
```

*L'astérisque, que l'on appelle le pointeur « **HEAD** » nous indique que nous sommes toujours sur notre branche master.*



Il faut alors passer sur la branche créée en déplaçant le pointeur HEAD. Pour cela on va utiliser la commande **`checkout`**. Par exemple :

```
git checkout ma-branche-test
```



Désormais le pointeur est bien sur notre nouvelle branche, les modifications n'impactent donc plus la branche master.

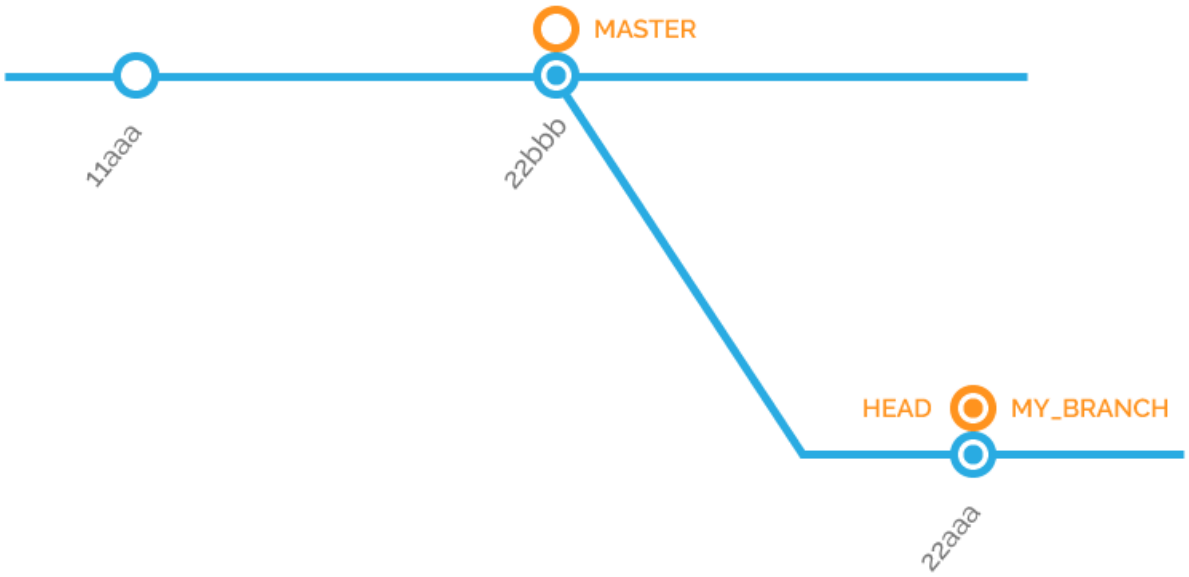
On peut aussi créer une nouvelle branche et déplacer le pointeur directement sur celle-ci afin d'utiliser une seule commande. Par exemple :

```
git checkout -b ma-branche-test
```

On obtient alors le même résultat.

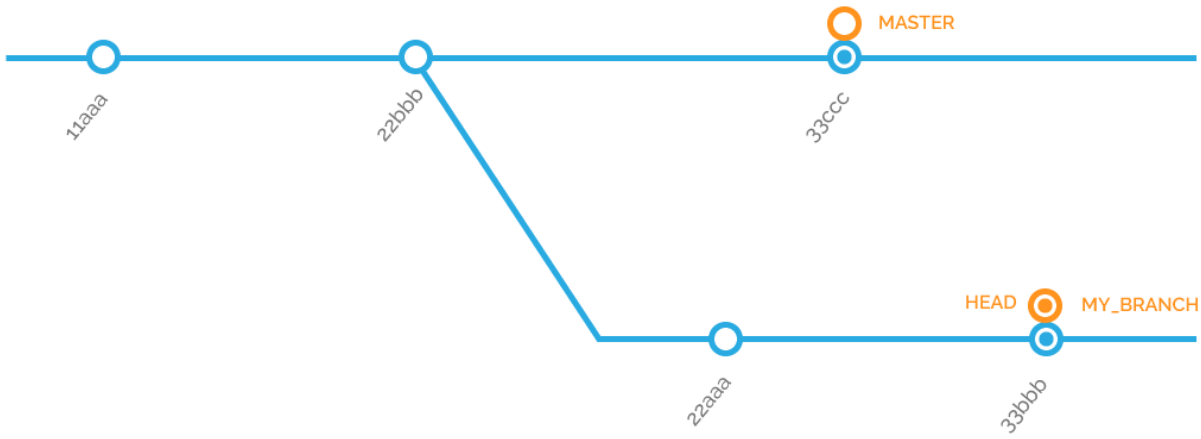
Faire des commits sur une branche

Lorsqu'on créé une nouvelle branche elle contient l'historique de la branche master. Les fichiers de cette nouvelle branche ont donc un statut **non modifié**. Si on fait des commits sur cette nouvelles branche, cela n'impactera donc pas l'historique et les fichiers de notre branche master. Les deux branches sont indépendantes.



Mettre à jour une branche

Si la branche master est mise à jour entre temps. Votre nouvelle branche, elle, ne sera alors plus à jour.



Pour mettre à jour notre branche avec les modifications apportées sur master, on utilise la commande :

```
git rebase master
```

Vous pouvez ensuite continuer à travailler sur votre branche.

Dans le cadre d'un travail distant, il faudra d'abord récupérer la version à jour sur le remote, avant de mettre à jour labranche. Pour cela on va utiliser la commande :

```
git pull
```

Fusionner une branche

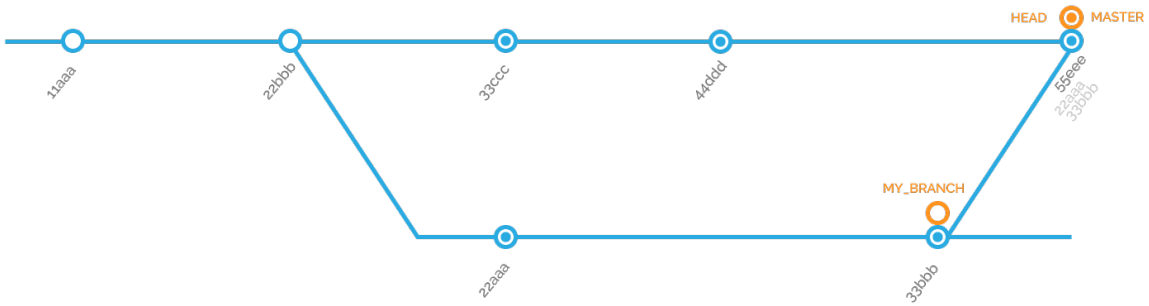
Une fois seulement que le travail sur notre branche est fini et qu'on est sûr que notre développement fonctionne. Il faut maintenant passer nos modifications sur la branche master pour la mettre à jour.

Pour cela il faut d'abord revenir sur notre branche master avec la commande :

```
git checkout master
```

Ensuite, on va faire ce que l'on appelle un « merge », c'est à dire que l'on va ajouter l'historique de notre branche à la branche master. On utilise donc la commande **`merge`**. Par exemple :

```
git merge ma-branche-test
```



Dans le cadre d'un travail distant, il faudra ensuite envoyer la version du projet à jour sur le remote. Pour cela on va utiliser la commande :

```
git push
```

Avoir des informations

Pour avoir des informations sur une branche. Par exemple, savoir si notre projet est à jour, ou si on a des commits en retard, on peut utiliser la commande :

```
git fetch
```

Cette commande va nous donner la marche à suivre.