git checkout

La commande `git checkout`

La commande `git checkout` est l'une des plus utilisées dans `Git`.

Elle permet de se déplacer vers une autre branche ou vers un `tag` (comme nous le verrons), vers un autre `commit`, ou de restaurer la version indexée.

Il faut vraiment voir la commande `checkout` comme une commande de navigation de `HEAD`. Autrement dit elle permet de déplacer `HEAD` où vous souhaitez.

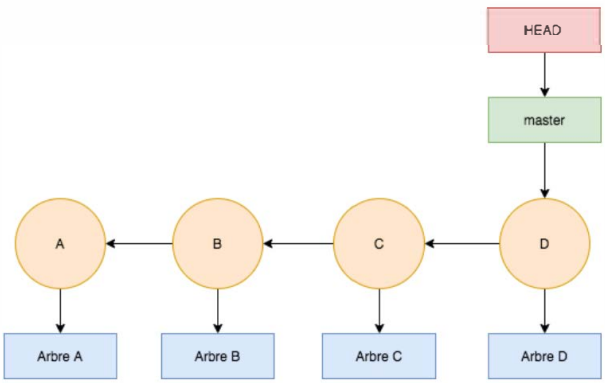
Nous verrons dans le chapitre suivant comment fonctionne le `checkout` sur les branches.

Dans cette leçon nous allons voir comment naviguer entre les `commits`.

Naviguer entre les `commits`

La commande `git checkout hashCommit` permet de déplacer `HEAD` vers le `commit` spécifié et de mettre à jour le répertoire de travail et l'`index` à la version correspondant à ce `commit`.

Prenons la configuration suivante :

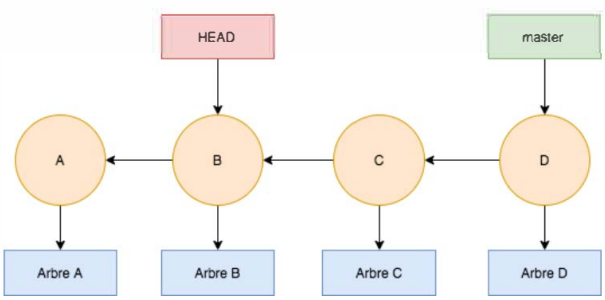


Par défaut, comme nous l'avons vu, `HEAD` pointe sur une branche, en l'occurrence la branche principale `master` créée par `Git` par défaut.

Mais si nous faisons :

```
git checkout hashCommitB
```

Que va t'il se passer ? Nous serons dans la situation suivante :



Nous avons déplacé `HEAD` vers le `commit B`.

Si nous faisons :

```
cat .git/HEAD
```

```
git status
```

Nous aurons en rouge :

```
HEAD detached at 9e633d5
nothing to commit, working tree clean
```

Nous allons voir ensemble ce que cela signifie.

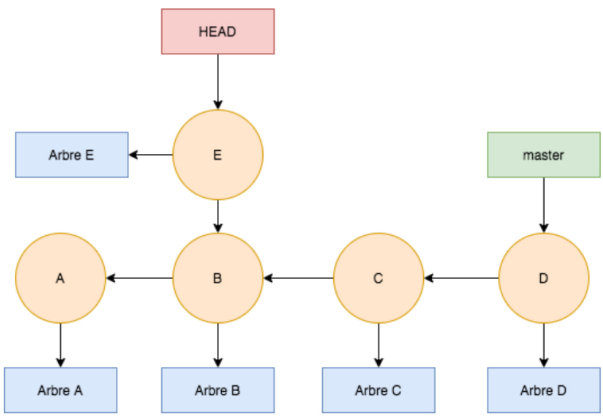
L'état `DETACHED HEAD`

Attention ! lorsque vous naviguez sur un `commit` en utilisant `git checkout commit` vous êtes dans un état très particulier appelé `DETACHED HEAD`.

L'état détaché signifie que `HEAD` ne fait référence à aucune branche et donc si vous effectuez des modifications vous ne pourrez pas les indexer et les sauvegarder.

Plus exactement vous pourrez le faire mais `Git` nettoie automatiquement les `commits` qui ne sont plus accessible par une branche.

Par exemple, si vous effectuez des modifications en étant sur le `commit B` puis que vous les indexez et les sauvegardez vous aurez l'état suivant :



Le problème est que le `commit E` n'est pas référencé par une branche.

Si vous vous replacez sur `master` en faisant `git checkout master`, plus aucune référence ne sera conservée pour le `commit E`.

Il sera très difficile d'y accéder : il faudra retrouver le `hash` du `commit E` et faire `git checkout hash`.

`Git` va détruire le `commit E` automatiquement si aucune branche n'y fait référence après une période de 90 jours.

Le cas d'utilisation de `git checkout commit` est donc simplement de lancer votre projet à cette version donnée pour regarder comment il était.

Il ne faut surtout pas faire de modifications, ou alors il faut créer une branche comme nous le verrons.

Annuler définitivement des modifications

Avec la commande `git checkout -- fichier` vous pouvez rétablir la version du fichier pointée par `HEAD`.

Notez que `git checkout fichier` et `git checkout -- fichier` font exactement la même chose. La différence est que si une branche a le même nom qu'un fichier (c'est improbable mais quand même), ce sera la branche qui sera `checkout`. Il vaut donc mieux dans un gros projet utiliser `-- fichier`.

Autrement dit, vous avez effectué des modifications sur un fichier dans votre répertoire de travail et vous ne voulez pas les conserver, vous pouvez toutes **les supprimer définitivement et rétablir la version indexée**.

Si vous n'avez pas indexé de modifications alors l'index correspondra à la dernière version sauvegardée, donc au dernier `commit`.

Attention avec cette commande ! C'est l'une des rares commandes qui supprime définitivement des

version spécifique, à savoir celle correspondant au `commit`.

Dans ce cas `HEAD` ne sera pas déplacé, la version du fichier indexée et de travail seront remplacées par la version du fichier spécifié.

Par exemple :

```
git checkout c5f567 -- fichier1.txt
```

Attention ! Si le `fichier1.txt` contenait des modifications non sauvegardées (`commit`) elles seront définitivement perdues, et ce même si elles étaient indexées.

Comme vous pouvez le voir la commande `git checkout` n'est pas à prendre à la légère ! Vous pouvez perdre des modifications qui n'ont pas été sauvegardées avec `git commit`.