git reset

La commande `git reset` est l'une des plus importantes de `Git`. Il est très important de bien la comprendre et bien la maîtriser.

Elle permet de faire beaucoup de choses, et peut être dangereuse dans certains cas car elle **peut provoquer des pertes de données irréversibles**.

Désindexer des fichiers ou des dossiers indexés

La commande `git reset fichier` est l'inverse de `git add fichier` : elle permet de désindexer un fichier en conservant les changements dans le répertoire de travail.

En fait, elle prend la version du fichier du dernier `commit` et la met dans l'index. Cela revient effectivement à désindexer un fichier.

Si vous spécifiez un dossier, tous les fichiers contenus dans ce dossier, et le dossier lui-même seront désindexés.

Sur `VS Code` cela revient à appuyer sur le bouton `-` dans l'onglet `Source Control` sur un fichier dans le groupe `STAGED CHANGES`.

Désindexer un fichier est utile si vous voulez diviser vos modifications en plusieurs `commits`.

Par exemple, mettons que vous avez travaillez sur une fonctionnalité et une correction de bug, et que par inadvertance vous avez tout indexé avec `git add .`.

Pourtant vous souhaitez faire un `commit` pour la correction de `bug` et un autre pour la nouvelle fonctionnalité.

Vous allez donc pouvoir désindexer les fichiers modifiés pour la correction de bug avec `git reset`, `commit` les modifications restantes pour la fonctionnalité, puis réindexer les fichiers pour la résolution de bug et faire un second `commit`.

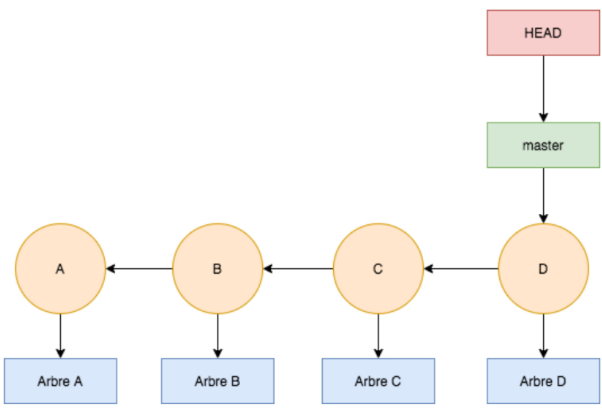
Déplacer la branche sur laquelle pointe `HEAD`

La commande `git reset commit` permet de déplacer la branche sur laquelle pointe `HEAD` sur le `commit` sélectionné.

Nous avons vu que `HEAD` contient la référence vers la branche sélectionnée, par défaut `master`.

Par défaut `git reset commit` va donc déplacer la branche `master` sur le `commit` spécifié.

Prenons par exemple la situation suivante :

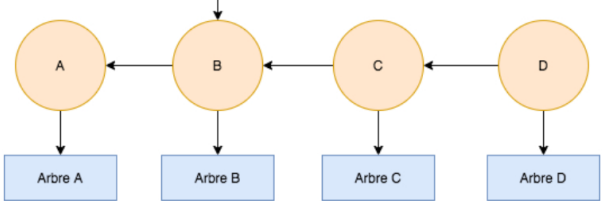


Que se passe t-il si nous faisons :

```
git reset B
```

Où `B` est le `hash` du `commit B`.

Voici le résultat :



`HEAD` pointe toujours sur `master` :

```
cat .git/HEAD
```

Donne toujours :

```
ref: refs/heads/master
```

Si vous faites `git status`, vous obtiendrez quelque chose comme cela :

```
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

  modified:   fichier1.txt
  modified:   fichier2.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

En fait, par défaut, la dernière version (celle du `commit D` dans notre exemple) est conservée dans le répertoire de travail.

Par contre, les versions correspondants aux `commits C` et `D` de notre exemples sont retirés du dépôt, elles ne sont plus sauvegardées.

Les trois options

Il existe trois options avec `git reset` qu'il est très important de connaitre.

`git reset --soft`

La première option est `--soft`, elle permet de déplacer la branche sur laquelle pointe `HEAD` sur le `commit` spécifié sans rien faire d'autre.

Autrement dit voici la version des fichiers dans les trois zones `Git` :

Dans le répertoire de travail la version est celle correspondant au dernier `commit` : rien ne change par rapport à avant de lancer la commande.

Dans l'index la version est celle correspondant au dernier `commit` : rien ne change non plus par rapport à avant de lancer la commande.

Dans le répertoire, `HEAD` pointe sur `master` et `master` pointe sur le `commit` spécifié. Ce qui signifie qu'il n'y a plus aucune référence vers les `commits` suivants (`C` et `D` dans notre exemple). **Ces versions ne sont donc plus sauvegardées dans le dépôt** !

`git reset --mixed`

La seconde option `--mixed`, est celle par défaut, c'est-à-dire celle utilisée en faisant `git reset commit` sans passer d'option.

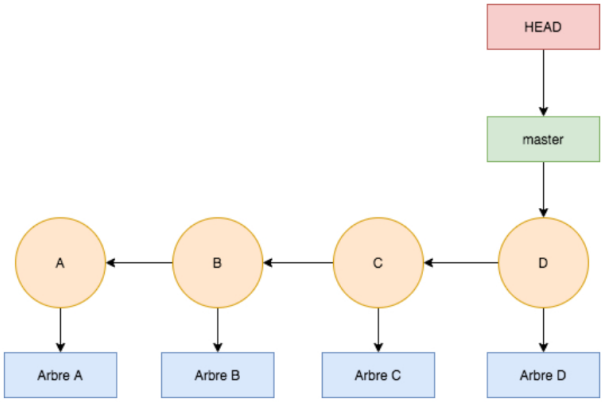
La différence avec `--soft` se situe au niveau de l'index : **dans l'index, la version des fichiers correspond au `commit` spécifié**.

Autrement dit, les modifications des `commits C` et `D` n'existent plus dans l'`index`. **Elles sont uniquement présentes dans le répertoire de travail**.

`git reset --hard`

La dernière option `--hard` est dangereuse : **les modifications sont définitivement perdues**.

Autrement dit elles ne sont plus dans le répertoire de travail, plus dans l'index et plus dans le dépôt !



Imaginons que le `commit C` soit la moitié d'une fonctionnalité et qu'il contient comme message `chantier en cours`. **Vous n'avez pas `push` ni le `commit C` ni le `D`.**

Vous aimeriez supprimer le `commit C` car la version `D` contient la fonctionnalité terminée avec un message de validation correct.

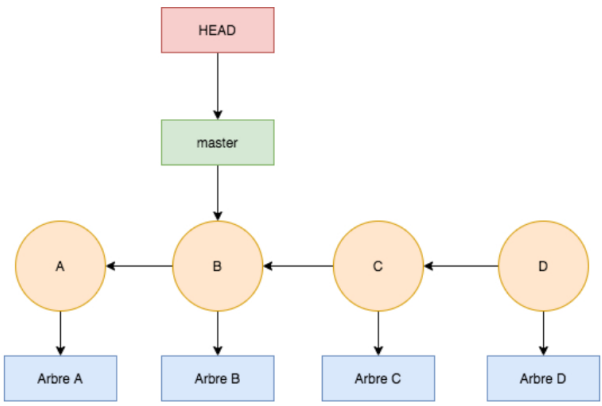
Commençons par faire :

```
git reset --soft B
```

Ou :

```
git reset --soft HEAD~2
```

`HEAD~2` ou `HEAD^^` signifient remonte de deux `commits` à partir du `commit` pointé par `HEAD`. Comme `HEAD` pointe sur `master` qui pointe sur `D`, cela revient à pointer sur `B` :



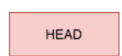
Comme nous avons fait `soft`, l'index et le répertoire de travail contiennent la version des fichiers correspondant au `commit D`.

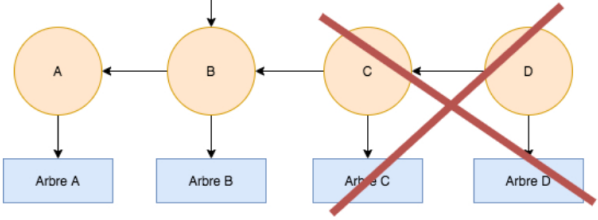
Nous n'avons plus qu'à `git commit` !

Cela créera un nouveau `commit` avec la version des fichiers correspondant au `commit D` qui n'existe plus !

En résumé, nous avons écarté complètement le `commit C` et créé un nouveau `commit` qui a la même version des fichiers que celle du `commit D`.

Nous obtenons :





Le `commit E` et l'arbre `E` ont la même version des fichiers que le `commit D` et l'arbre `D` mais sont des objets différents.