POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

GIT : LES BRANCHES PAR UN EXEMPLE

25 SEPTEMBRE 2023

JÉRÔME COLLIN, CHARLES DE LAFONTAINE, MERIAM BEN RABIA
GIGL | Polytechnique Montréal

N.B. Le masculin est utilisé pour alléger le texte.

TABLE DES MATIÈRES

GIT : LES BRANCHES PAR UN EXEMPLE

1. NOTIONS À SAVOIR *A PRIORI*

La branche *master* existe toujours et est créée par *Git* directement, mais son comportement est le même que pour toute autre branche. Certains aiment bien définir une branche comme étant un pointeur sur un *commit* et duquel on part pour créer un développement en parallèle. Ça donne un bon aperçu de ce qui se passe !

Ce n'est pas le but de multiplier les branches pour le plaisir, car il faut par la suite les gérer ! Ultimement, on veut fusionner les améliorations stables dans la branche parent et détruire la branche qui devrait jouer un rôle limité dans le temps.

On peut se déplacer d'une branche à l'autre dans *Git*. L'indicatif **HEAD** (pointeur) nous permet de savoir dans quelle branche on se trouve. Le symbole **`*`** peut l'indiquer aussi.

On peut toujours avoir un conflit lorsqu'on modifie des lignes de code à la fois dans la branche actuelle et la branche parent dans un même fichier. Le conflit apparaîtra uniquement à la fusion des branches (*merge*).

On peut toujours lister les branches locales. Ce sera simple si on n'a pas encore de branches dans un entrepôt quelconque :

```
$ git branch -l
* master
```

2. EXERCICE SIMPLE

On créera un petit fichier texte de 4 lignes et on fera quelques *commits* pour montrer la progression et l'indépendance des modifications entre la branche *master* et une branche appelée **`secondaire`**. On pourra afficher le contenu du fichier à chaque étape pour montrer le résultat concret. On changera le mot « non » pour un « oui » au bon endroit pour introduire un changement approprié. Voici le contenu du fichier au départ :

L'exemple si dessous montre que l'utilisateur a tapé la commande `ls` puisque le terminal lui avait donné l'invité `$` et le résultat de la commande est affiché en deuxième ligne et les suivantes s'il y a lieu :

```
$ cat branches.txt
modification avant la création de la branche : non
modification dans la branche secondaire : non
ligne non modifiée pour aider à la fusion...
modification dans la branche master : non
```

2.1. Avant le branchement

On suppose ici qu'il y a déjà un entrepôt cloné localement dans le répertoire courant à **`/home/sibru/inf1900/ing1234`**.

```
$ pwd
/home/sibru/inf1900/ing1234 (ou une sortie similaire...)
$ mkdir gitTest
$ cd gitTest
$ code .
```

Après avoir placé le contenu du fichier texte précédent dans l'éditeur et changé le « non » de la première ligne pour un « oui », on fera le premier *commit* avant la création de la branche. On sauvegarde le fichier sous le nom **`branches.txt`** dans l'éditeur et on effectue un premier *commit*.

```
$ git add branches.txt
$ git commit -m "Ajout de branches.txt"
```

2.2. On branche !

Avant, vérifions le contenu du fichier :

```
$ cat branches.txt
modification avant la création de la branche : oui
modification dans la branche secondaire : non
ligne non modifiée pour aider à la fusion...
modification dans la branche master : non
```

On crée la branche et on bascule vers cette branche en 2 commandes :

Avant, vérifions le contenu du fichier :

```
$ git branch secondaire
$ git checkout secondaire
$ git branch -l -v
      master          8fe5b704  Ajout de branches.txt
*     secondaire      8fe5b704  Ajout de branches.txt
```

[homologue] git checkout -b secondaire

2.3. Travail supplémentaire dans la branche

On peut faire autant de *commits* qu'on veut dans une branche et même créer une autre branche qui dérive de celle-ci. On reste avec notre intention simple du départ avec le changement à la deuxième ligne de notre fichier texte pour marquer notre passage dans la branche.

```
$ code .
$ git add branches.txt
$ git commit -m "Modification au sein de la branche secondaire"
modification avant la création de la branche : oui
modification dans la branche secondaire : oui
ligne non modifiée pour aider à la fusion...
modification dans la branche master : non
```

Ce sera la seule modification dans la branche. Il est important d'avoir fait un *commit* de toutes les modifications dans une branche avant de changer de branche, car on revient toujours dans une branche au dernier *commit* dans cette branche pour ce qui est de l'état des fichiers.

On peut contourner ce problème et donc conserver des modifications non « *commitées* » dans une branche et les retrouver plus tard (commande **`git stash`**), mais c'est déjà un peu plus compliqué… On l'évitera pour cet exercice.

2.4. On revient dans la branche *master*

On revient dans la branche *master* et on regarde le contenu du fichier d'intérêt :

```
$ git checkout master
$ git branch -l
*    master
     secondaire
$ cat branches.txt
modification avant la création de la branche : oui
modification dans la branche secondaire : non
ligne non modifiée pour aider à la fusion...
modification dans la branche master : non
```

On voit que la ligne 2 n'est pas modifiée ici car le changement est uniquement dans la branche **`secondaire`**. On fait la modification à la ligne 4 et on fait un *commit* :

Avant, vérifions le contenu du fichier :

```
$ code .
$ git add branches.txt
$ git commit -m "Modification au sein de la branche master"
```

2.5. On observe le résultat !

On a donc ceci dans la branche *master* :

```
$ cat branches.txt
modification avant la création de la branche : oui
modification dans la branche secondaire : non
ligne non modifiée pour aider à la fusion...
modification dans la branche master : oui
```

On peut observer graphiquement un peu l'état de nos branches :

Avant, vérifions le contenu du fichier :

```
$ git log --graph --oneline --decorate --all
*     f576e16a (HEAD -> master)  Modification au sein de la branche master
| *   fcf0b6c4 (secondaire)      Modification au sein de la branche secondaire
|/
...
```

Cette commande n'est pas idéale et on peut obtenir un meilleur rendu visuel d'un outil graphique (*gitk*, *GitKraken*, etc). Voir d'autres options ici. Dans le cas de quelques branches, la commande proposée ci-haut peut être suffisante. Le plugiciel intégré à *VSCode pour Git* montre très bien les branches également.

2.6. On fusionne !

On veut ramener le contenu de la branche **`secondaire`** dans la branche *master* avec une fusion :

```
$ git merge -m "Fusion de secondaire" secondaire
Auto-merging simon/gitTest/test1/branches.txt
Merge made by the 'recursive' strategy.
simon/gitTest/test1/branches.txt | 2 +-
1 file changed, 1 insertion(+), 1 deletion(-)
$ cat branches.txt
modification avant la création de la branche : oui
modification dans la branche secondaire : oui
ligne non modifiée pour aider à la fusion...
modification dans la branche master : oui
```

Une fusion est le résultat pris de 3 *commits* (celui de chacune des branches [*master*, **`secondaire`**] et de celui à partir duquel a eu lieu le branchement [le tout premier *commit* dans notre cas]). Ceci est appelé le « *3 way merge* ». Le résultat d'une fusion doit aussi être un *commit*, d'où l'option `-m` sur la ligne. Les modifications ont été ramenées dans la branche `master`. Il peut aussi arriver que la branche master n'a pas été modifiée au moment de faire la fusion de la branche. Dans ce cas, il n'y a pas de possibilité de conflits de fusion et on n'a pas de « *3 way merge* », mais un « *fast-forward merge* » qui est donc plus simple.

On peut considérer que la branche **`secondaire`** n'est plus utile (dans bien des circonstances). On peut la détruire :

```
$ git branch -d secondaire
Deleted branch secondaire (was 4c9eb5db).
```

3. UN MOT SUR *GIT REBASE*

Il peut arriver qu'une branche ait une durée de vie assez longue. Le cas le plus évident est lorsqu'une nouvelle fonctionnalité est longue à implémenter et à tester. On retarde donc la fusion dans la branche parent. Le problème est que la branche parent continue d'être modifiée (ajout de *commits*) par d'autres développeurs. Les changements dans les deux branches s'accumulent et la dissemblance augmente entre les deux dans les fichiers…

On a donc des changements indépendants qui s'accumulent et le stress augmente en vue d'un éventuel **`git merge`** qui risque d'exposer des conflits qui seront difficiles à réconcilier… Dit autrement, le point de départ de la création de la branche devient de plus en plus éloigné dans le temps. Est-ce qu'on peut ramener ce point plus proche du moment présent ? Oui !

On suppose alors que les changements dans la branche parente sont assez stables. Pourquoi alors ne pas les ramener dans la branche secondaire pour garder un bon niveau de ressemblance entre les deux branches pour augmenter les chances d'avoir une fusion sans conflit ou sans trop de conflits trop gros ou trop nombreux ? *Git* le permet.

On fait un **`git rebase`** dans la branche pour inclure les modifications de la branche parent dans la branche secondaire. Dit autrement, et plus près de la réalité, la branche parent et la branche secondaire diverge à partir d'un point plus rapproché dans la branche parent.

4. POUR ALLER PLUS LOIN AVEC *GIT*

Les références suivantes sont intéressantes pour approfondir sa maîtrise de *Git*, notamment au niveau des branches :

- Les branches avec *Git* [sections 1-4 du chapitre 3]
- Guide *Atlassian* des branches
- Guide alternatif pas à pas sur les branches
- Une introduction aux branches en vidéo