WORKFLOW GIT
Babel

—

---

1. Démarrage d'une fonctionnalité

- **git checkout master**
- **git pull**
- **git checkout -b *nomDeLaBranche***

2. Développement de la fonctionnalité

- **git add *mesModifications***
- **git commit -m "branchName(feat/fix/docs/refactor/test): description"**
- **git pull**
- **git push origin *nomDeLaBranche***

3. Test d'une fonctionnalité (compatibilité avec master)

Workflow afin de tester si notre branche ne comporte aucun conflit ni aucune régression par rapport à master:

- **@ETAPE_2**
- **git checkout master**
- **git pull**
- **git checkout nomDeLaBranche**
- **git merge master**
- ***(résolution des conflits si il y en a)***
- **git push origin nomDeLaBranche**

4. Test d'une fonctionnalité (compatibilité avec les fonctionnalités d'autres personnes | dev)

Workflow afin de tester si notre branche, couplée avec les fonctionnalités en développement des autres ne contient aucun conflit ni aucune régression par rapport à master:

- **@ETAPE_2**
- **git checkout dev**
- **git pull**

- **git merge nomDeLaBranche**
- ***(résolution des conflits si il y en a)***
- **git push origin dev**

5. Livraison d'une fonctionnalité

Quand une fonctionnalité est prête à être livrée (toutes les @ETAPE_2 ont été finies), alors on va s'assurer de sa compatibilité avec le reste du projet (@ETAPE_3, @ETAPE_4), puis on livre sur master. (Il est avisé de demander une mini code review avant les push de fonctionnalités qui représentent plus que des changements mineurs [typos, etc..]).

- **@ETAPE_2**
- **@ETAPE_3**
- **@ETAPE_4**
- **git checkout master**
- **git pull**
- **git merge nomDeLaBranche**
- **git branch -D nomDeLaBranche**
- **git push origin master**

Notes:

A. Il est important de ne **jamais** push un travail inachevé, non testé ou buggé sur master. Cette branche doit être fonctionnelle à **TOUT** moment.

B. Il est primordial de ne **jamais** merge **dev** sur une autre branche. Le workflow d'utilisation de la branche dev est unilatéral, c'est à dire qu'elle accepte de nouvelles fonctionnalités, mais n'est jamais exportée elle-même sur une autre branche, **JAMAIS**.

   (En somme, vous ne devez jamais avoir à utiliser ***git merge dev***)

C. Si **dev** se retrouve "cassée" (trop de features de tests ont cassés le comportement normal du projet, ou bien un fail sur les push a été commis), alors il est important de notifier le channel Slack **advanced_cpp1** afin de demander un reset de la branch **dev** sur **master** (une remise au propre).

Commandes:

git merge nomDeLaBranche

Fusionne le travail de votre branche actuelle avec nomDeLaBranche

git pull

Récupère les modifications effectuées sur votre branche et livrées sur *origin*.

git add listeDeFichiers

Ajoute la listeDeFichiers au *staging*, cela représente les différentes modifications qui seront prises en compte lors d'un commit.

git commit -m "Message de commit"

Un commit est un objet qui regroupe différentes modifications ainsi qu'un message et qui est identifié par une clé unique. Lors d'un push, on pousse tous les commits effectués sur la branche locale, sur la branche d'origine.

La commande **git commit** permet de regrouper toutes les modifications listées par **git add** en un objet de commit.

git push origin nomDeLaBranche

Le push permet d'envoyer tous les commits locaux sur la branche voulue en remote (origin). Ils seront ainsi accessibles par tous les autres utilisateurs de la branche via un pull.

git checkout nomDeLaBranche

Le checkout permet de se déplacer de branche en branches.

git checkout -b nomDeLaBranche

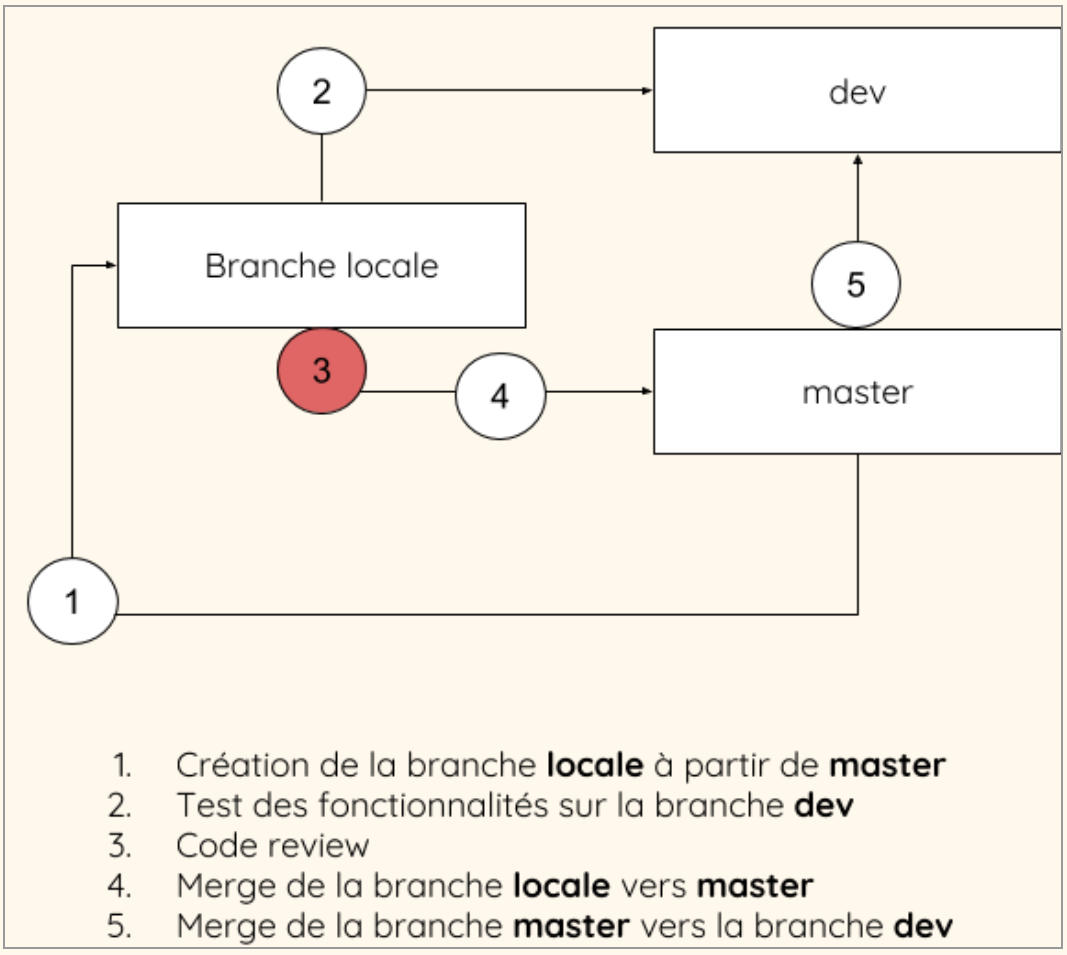
Lorsque l'option -b est donnée à **git checkout**, alors une nouvelle branche va être créée en local (en attendant d'être push sur origin) à partir de la branche actuelle (c'est à dire que l'état du repository lors de la création de la branche sera le même que celui sur la branche précédente (celle depuis laquelle la commande a été lancée)).

git branch -D/-d nomDeLaBranche

L'option **-d** permet de supprimer la branche nomDeLaBranche en local (origin ne sera pas impactée)

L'option **-D** permet de supprimer la branche nomDeLaBranche en local ET sur origin.

Schéma récapitulatif (*attention, ici les étapes sont simplifiées, et les numéros de correspondent pas forcément au listage fait au début de ce document*) du workflow git pour le BABEL.



En cas de doute: **Slack | advanced_cpp1**