Connecthive

Guide de mise en œuvre Git

Gitflow

Le workflow Gitflow définit un processus de création de branches Git ayant pour objectif une norme de livraison.

Gitflow définit des rôles très spécifiques aux différentes branches:

- principale (master)
- fonctionnalité (feature)
- correction, maintenance
- enregistrement des versions

Gitflow s'appuie sur la notion de proposition de modification (pull requests).

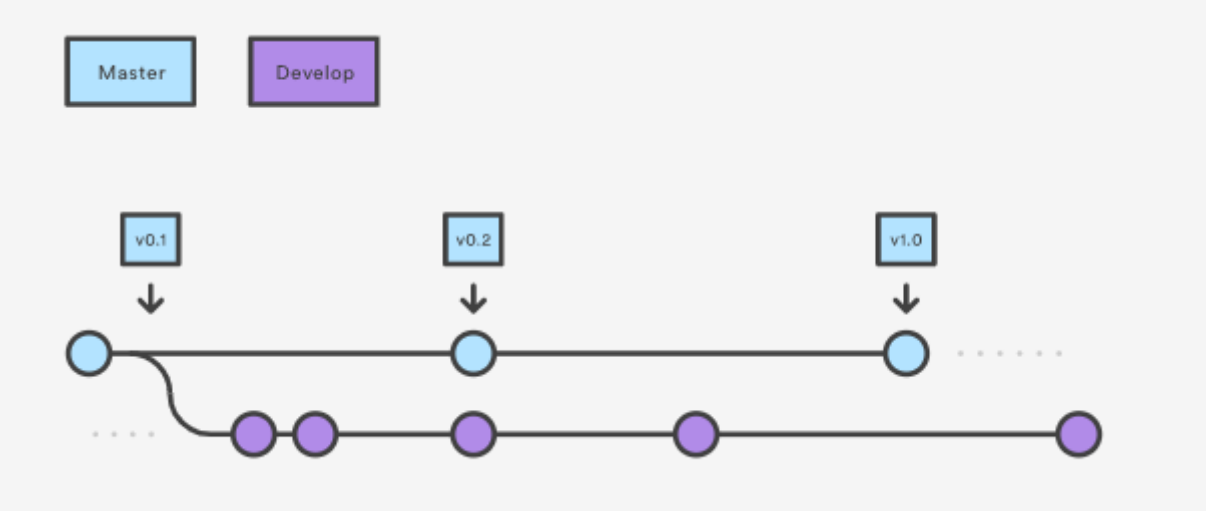

Figure 1: Branches develop et master

Le workflow de la **Figure 1** utilise deux branches pour sauvegarder l'historique du projet. La branche principale (*master*) est dédiée à l'historique des versions. Cette branche doit être taguée avec des numéros de version. La branche de développement (*develop*) est dédiée aux fonctionnalités successives. A l'initialisation, on créée une branche *develop* sous la branche *master*. Celle ci contiendra l'historique complet alors que la branche *master* en contient une version abrégée.

Les membres de l'équipe travaillent sur des branches situées en dessous de la branche *develop*.

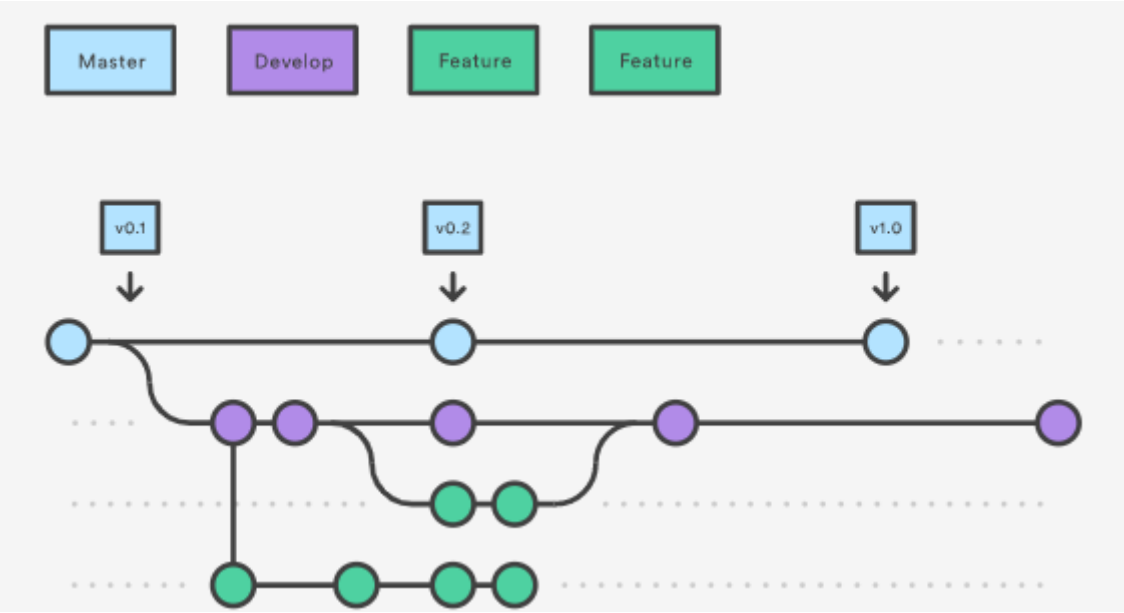

Figure 2: Branches de fonctionnalité

Pour chaque nouvelle fonctionnalité x on crée une branche spécifique (*feature x*) en dessous de la branche *develop* (à partir de la dernière version de cette branche *develop*), voir **Figure 2**. La branche *feature x* est mergée dans *develop* lorsqu'elle est terminée.

La branche *develop* est à son tour mergée dans la branche *master* pour donner lieu aux différents tags de version.

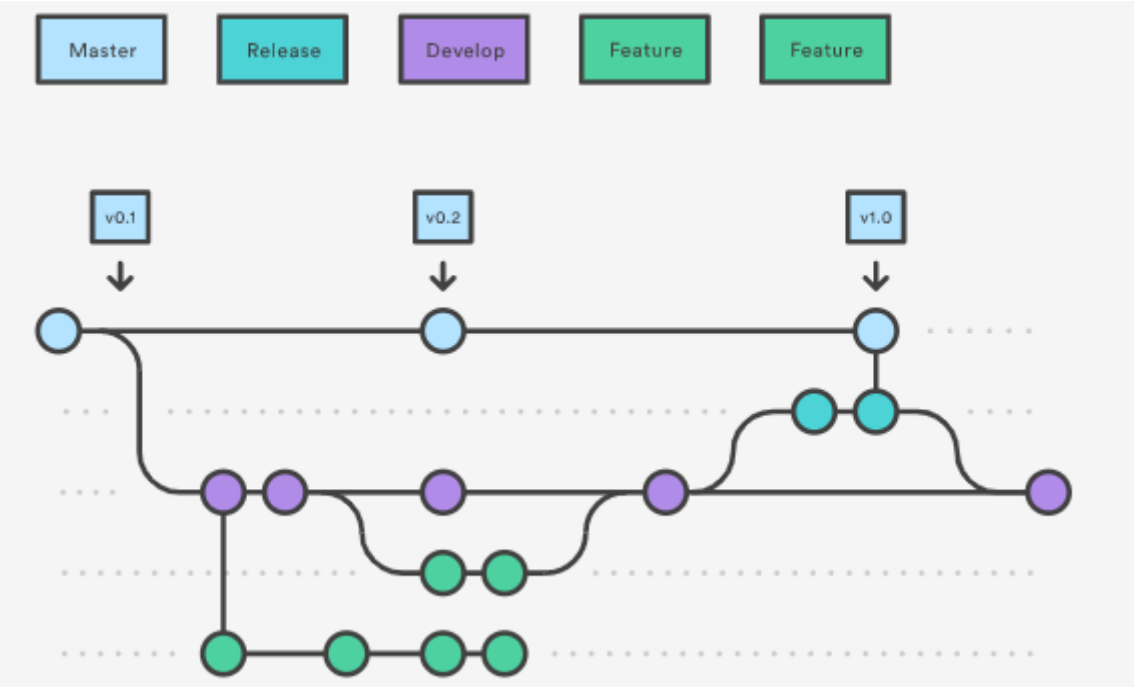

Figure 3: Branches de livraison

Lorsqu'une *feature* est terminée elle est mergée dans la branche *develop*. Une branche *release* (**Figure 3)** est ensuite créée sous la branche *develop* afin d'empêcher qu'aucune fonctionnalité supplémentaire ne doit ajoutée (elle sera ajoutée à la fonctionnalité suivante). La branche *release* est dédiée uniquement aux corrections de bug, à la génération de la documentation et aux opérations de livraison. Lorsqu'elle est prête elle est mergée dans la branche principale (*master*) et taguée avec un numéro de version. Il faut alors merger la branche *master* dans la branche *develop*. Ainsi une équipe peut travailler sur la branche release alors qu'une autre peut travailler sur les features.

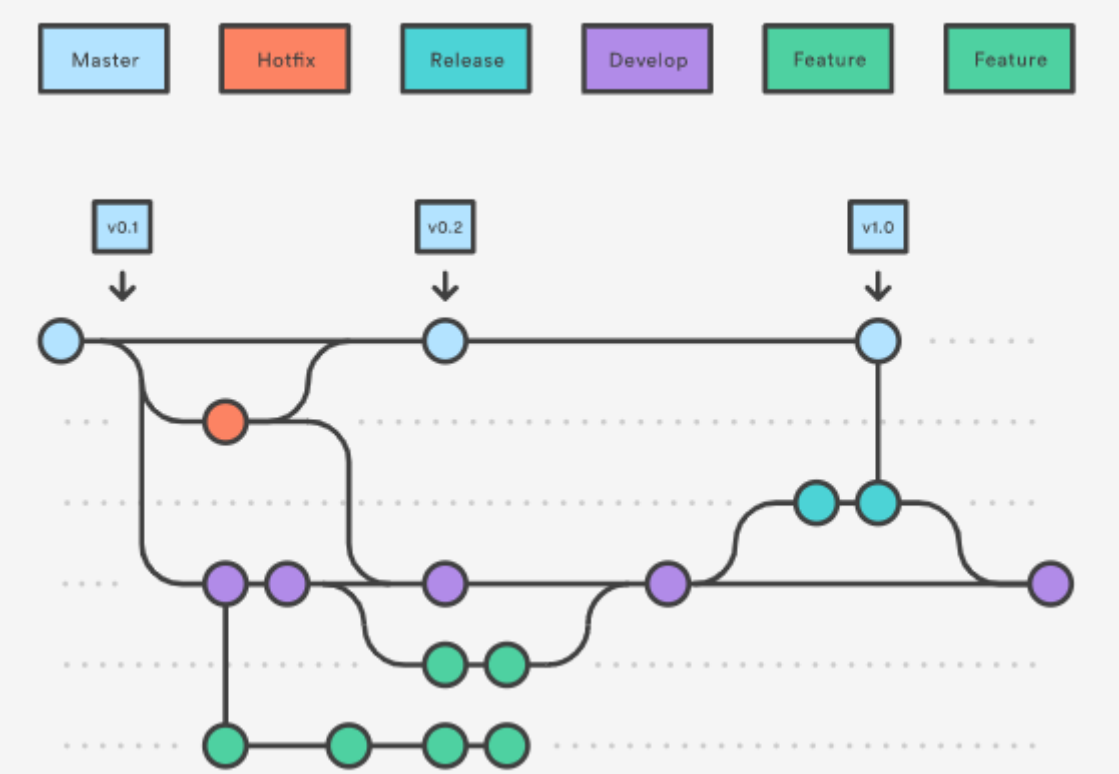

Figure 4: Branches hotfix

Les branches *hotfix* (**Figure 4**) sont dédiées à la maintenance et utilisées pour appliquer des modifications aux versions de production de la branche *master*. Elles sont exclusivement créées sous la branche *master*.

A l'issue d'une correction, il est nécessaire de la merger dans *master* et *develop* (ou dans la branche *release* actuelle), puis taguer master avec un nouveau numéro de version. Ainsi les corrections de bug peuvent être réalisées sans interrompre le reste du workflow.

Git rebase

Un utilitaire d'intégration des changements est *git merge*. Le *merge* est toujours un enregistrement progressif des changements alors que *git rebase* est destiné à la réécriture de l'historique.

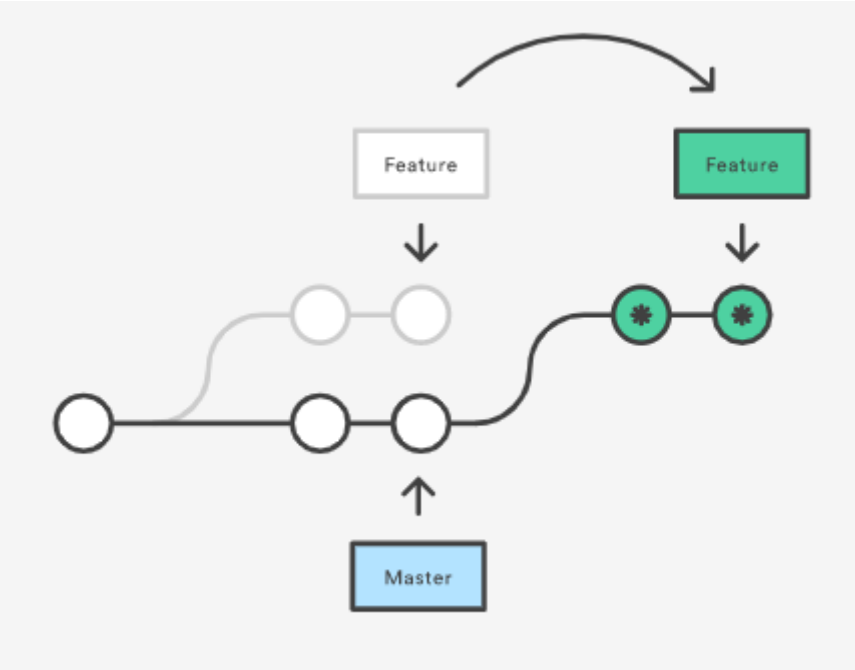

Figure 5: Git rebase

L'opération rebase (**Figure 5**) consiste à changer la base d'une branche d'un commit vers un autre: la branche semble identique mais en réalité elle est composée de nouveaux commits.

L'objectif est de maintenir un historique de projet linéaire. Un cas courant est la mise à jour d'une branche *feature* à partir de la branche *master*, ce qui permettra par la suite de faire un merge propre de la branche *feature* vers la branche *master*.

On ne doit jamais faire un rebase des commits qui ont été pushés vers un dépôt public. Le rebase remplacerait les anciens commits par les nouveaux, en faisant disparaître l'historique du projet.

Références

https://www.atlassian.com/fr/git/tutorials