TP Gitflow

A la découverte d'une utilisation avancée du git

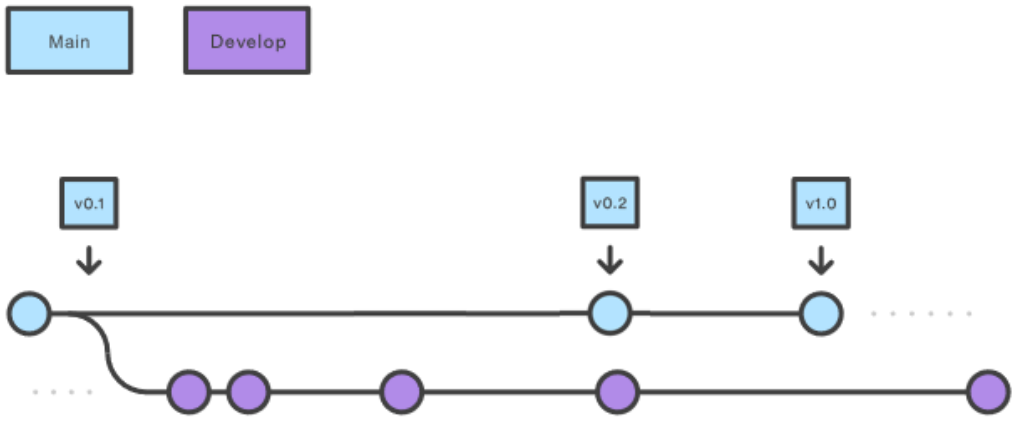
Quand on débute, la plupart du temps lorsque l'on maintient une application, on ne reste que sur une seule branche git (anciennement *master* désormais *main*) avec une progression constante dans le développement marquée par des versions régulières en local (*add*, *commit*) puis sur le serveur (*push*, *pull* ou *fetch*, *rebase* et *merge* en cas de conflits). C'est le **workflow centralisé** que nous avons vu au TD/TP2.

Une utilisation plus avancée de git en équipe s'appelle le **gitFlow**. Il existe de nombreuses variations. Voici la proposition originale[1]. Nous allons vous illustrer la philosophie de cette approche dans un premier temps puis vous proposer de l'appliquer sur votre projet de Gestion des TERs

PARTIE 1 : LE GITFLOW en 4 ETAPES

Etape 1 : Branche develop

Une bonne pratique est de créer une branche *main* qui supporte les versions majeures de l'application (*releases* ou livraisons) et une branche *develop* possédant les versions en cours de développement courantes.



Vous allez travailler en binome, l'un sera le garant de la version stable, l'autre le développeur.

Sur un dépôt existant, le garant (rôle *maintener*) à partir d'une branche *main* va construire une branche *develop* (commande *branch* en local ou *git push origin develop* distante) puis transférer le contenu de *main* dans *develop* (*git checkout -b develop*).

[1] https://nvie.com/posts/a-successful-git-branching-model/

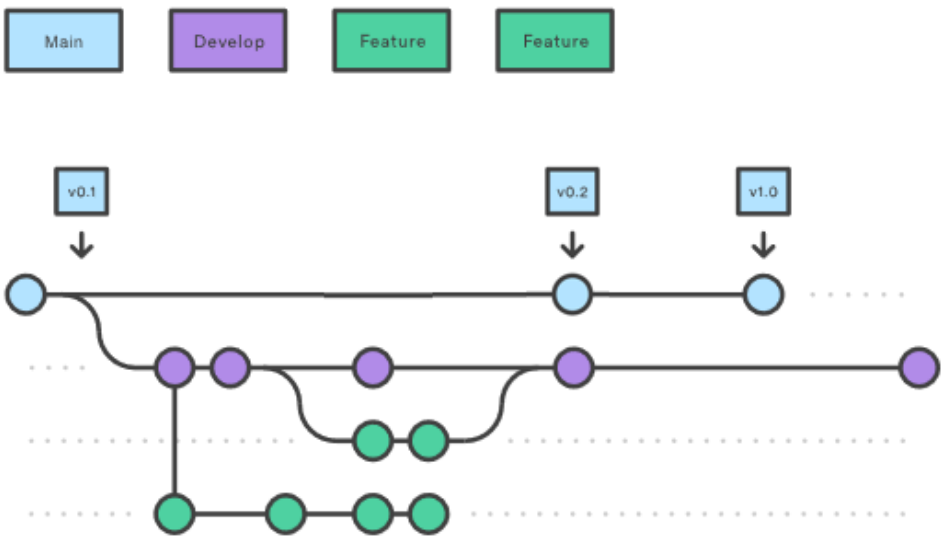
Le (les) développeur(s)(rôle *developer*) vont cloner ce dépôt. Ils travaillent donc sur leur version courante de l'application dans la branche develop (souvent plus avancée que celle livrée). Ils n'ont pas les droits en écriture sur le dépôt initial.

Etape 2 : Branche Feature

En cours de développement chaque développeur ou équipe travaille sur une fonctionnalité (*feature*). Nous allons créer une branche (à partir de *develop*) par feature. Au cours de leur travail, les développeurs produisent plusieurs commits pour finalement fusionner (*merge*) avec la branche d'origine.

Ainsi chaque fonctionnalité avance en parallèle et se resynchronise sur la branche *develop* sans jamais polluer la branche principale (*main*).

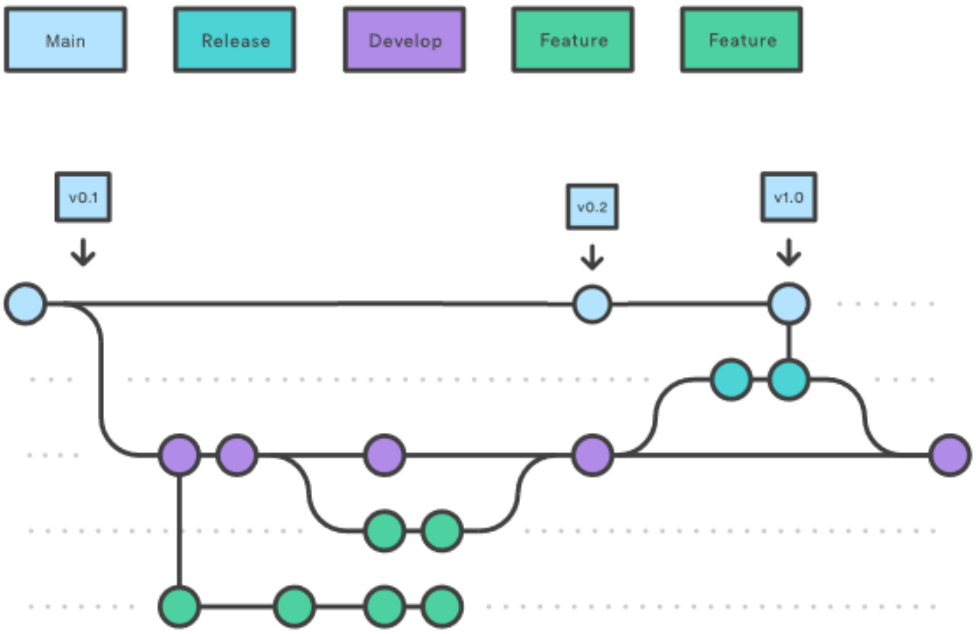
Lors d'un push, ce merge sera réalisé par le garant. Quand il est demandé par les développeurs (via l'interface web) cela se nomme merge_request (ou push_request, question de vocabulaire dépendant des environnements) et seul le garant (une fois celui-ci demandé par le devloper) peut le valider et le réaliser.



Pour gérer les features, par exemple, le développeur va créer une seconde branche pour faire la feature 1, le garant (qui est aussi un développeur) se charge de la feature 2 (non mergée pour l'instant). Les commandes utiles sont *commit*, *checkout develop* (basculer sur une branche), *checkout -b featureX* (raccourci de branch puis checkout).

Etape 3 : Branche Release

Lorsque les fonctionnalités sont validées, il est possible de réaliser une livraison (*release*). Idéalement il faut garder une trace de la préparation de cette release permettant de corriger d'éventuels bugs de dernière minute sur la version à livrer sans être impacté par la suite des développements.

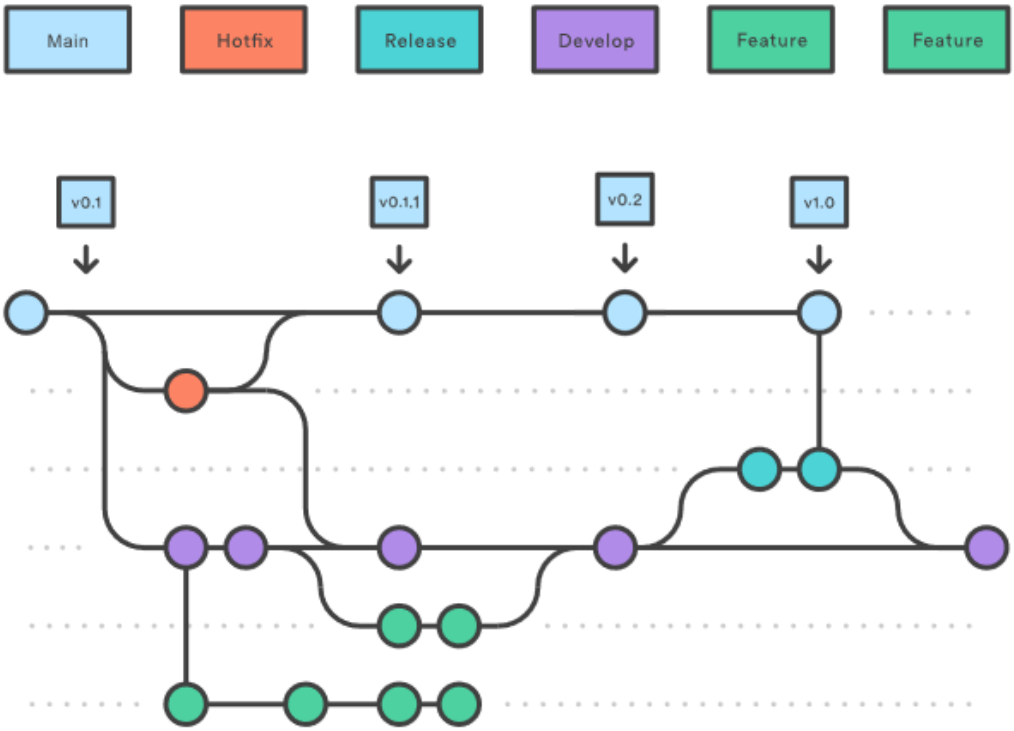


Pour réaliser cela le garant va créer une branche (*release*) à partir de la branche *develop*, réaliser un commit (simulant des corrections de soucis découverts lors de la phase de préparation de la livraison) puis la merger avec la branche principale (*main*) sans oublier de remonter ces modifs aussi dans la branche *develop*. Au final cette branche est supprimée via l'option *-d* de la commande *branch*.

Sur la figure, deux branches feature ont été créées, dont une est déjà réintégrée dans *develop* et fait partie de la livraison.

Etape 4 : Branche hotfix

Lors d'une livraison de release, il se passe très souvent des problèmes aussitôt ou juste après la livraison. C'est ce que l'on appelle des *hotfix* ou alors simplement de la maintenance. Il ne faut pas oublier de tenir au courant l'équipe de développement.

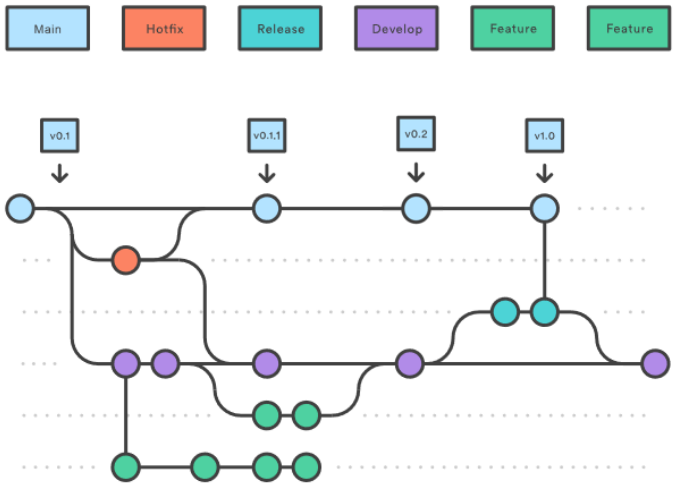


Une gestion fine des numéros de versions sur 3 niveaux (majeure, mineure, correctif) permet de se retrouver facilement dans toutes ces branches (v0.1 démarrage, v0.1.1 update en production suite au hotfix, v0.2 ajout de le première feature, v1.0 première livraison). D'un point de vue gestion de projet, il est même recommandé de faire avancer les numéros de version sur la branche principale au fur et à mesure de l'avancée de la partie développement. Il est ainsi possible de voir la progression de l'application, sans réaliser les merge associés (commit simple où seule la variable de version évolue).

PARTIE 2 : MISE EN APPLICATION

Si vous n'êtes pas familier des différentes commandes git présentées précédemment, nous vous conseillons de pratiquer ces étapes, une par une par petits essais/erreurs sur de mini projets artificiels via gitlab.com.

Dans un second temps, nous allons reproduire le workflow de la figure ci-dessous pour la gestion de TER comme si nous étions une équipe de développement. A nouveau, nous vous conseillons de repartir d'un dépôt vierge dans lequel vous allez placer vos sources courantes concernant la gestion des TERs. N'oubliez pas de faire évoluer une variable globale de la classe principale permettant d'afficher la version courante.



Déroulement

- L'un de vous sera le garant de l'application (*maintener*), l'autre le développeur d'une nouvelle feature (*developer*).
- Le garant crée les branche *main* (importe ses sources), commit et crée la branch *develop* (inc).
- Le développeur se lance dans la première feature.
  Feature1 : Permettre de réaliser la génération d'un planning de soutenances (sans rapporteur). En sortie, chaque groupe affecté à un sujet doit avoir un créneau de soutenance défini. Attention, un prof encadrant potentiellement plusieurs groupes, ne doit pas avoir deux soutenances en même temps.
- Durant le même temps, le garant va développer une seconde feature.
- Feature2 : générer une page HTML simple (<table>, <tr>, <td>) pour afficher la liste courante des affectations groupes-sujets.
- Durant le développement de Feature2, Le garant corrige un problème en maintenance. La limite de la taille des groupe (initialement de 5) est exceptionnellement passée à 6. Modifier le code sur la branche *main* et met à jour la branche *develop*.
- Le développeur fini sa Feature1 par un commit, valide sa livraison par un fetch et devra merger sa branch Feature1 sur develop. Il demande alors un merge_request pour l'intégration dans la branche officielle.
- Le garant réalise la livraison à partir de *develop* sur une branche de *release* en mettant à jour la version en 1.0 puis remonte cette info dans la branch *develop*.

Comparez votre graphe des commits avec la figure d'origine …