Anypli Git Flow

Introduction:

Pour bien organiser la gestion des versions dans notre projet, on doit respecter quelques règles liées à la procédure de création des branches et des commits.

Ces règles suivent le branchement model GitFlow et sont adaptées pour être compatibles sur notre modèle de gestion des projets chez anypli.

Branches:

master:

C’est la seule branche dans laquelle on trouve aucune commit de dev, on trouve que des commits du fusion(merge), et chaque commit représente une version livrée aux utilisateurs(mise en prod). Et elle doit avoir un tag avec la version.

develop:

C’est la branche qui contient les commits de développement du sprint en cours. Généralement les développeurs ne travaillent que sur cette branche (sauf en cas de correction des bugs après la livraison).

release/version:

C’est un groupe des branches, et chaque branche est liée à une version. Elle contient la version du code entre la livraison et la mise en prod.(correction des bugs entre la livraison et la validation du sprint)

hotfix/version:

C’est un groupe des branches, et chaque branche est liée à une version. Elle contient les corrections des bugs après la mise en prod).

sprint/num:

Idéalement, on essaye de développer qu'un seul sprint à la fois qui est sur la branche **develop**. Mais parfois on est obligé de commencer un nouveau sprint avant la fin du sprint en cours. Donc on le fait sur une nouvelle branche dans le groupe **sprint**. Dès que le sprint en cours est fini, on merge **sprint** sur **develop** et n'utilise plus cette branche.

Messages des commits:

Les descriptions des commits doivent être en **Anglais**.

Si le commit a été fait sans un ticket associé, idéalement il faudrait prendre l'initiative de créer le ticket vous même. Il ne faut pas commiter plusieurs tickets/fix dans un seul commit.

Dans le cas d'une commit liée à une ticket:
#Ticket_ID: Description_de_la_commit

Exemple:
#345632: add camera permission to info.plist

Dans le cas de correction d'une vieille commit:
#fix previous commit: Description_de_la_commit (si c'est la commit juste avant)
#fix Commit_ID: Description_de_la_commit

Exemple:
#fix previous commit: remove pushed test code
#fix 5d6hf43: add unpushed asset

Dans le cas d'une commit non liée à aucune ticket :
#Type: Description_de_la_commit

Exemple:
#fix bug: fix overlapping labels on Edit Profile Screen
#init project: cocoapods integration
#init project: targets configuration

Pour les commits des merge on laisse le message auto-généré par le git.

Dans un commit, il faudrait commiter que les fichiers concernés par la modification afin de faciliter la gestion de l'historique (en cas de revert par exemple).

Ce qu'il ne faut pas faire : Évitez les commits style 550, fix, etc. Dans ce cas, n'importe quel intervenant sur le projet ne comprendra pas ce qui a été fait exactement dans ce commit.

Ne **jamais** pusher **un code de test** et ne jamais pusher un code sans l'avoir testé

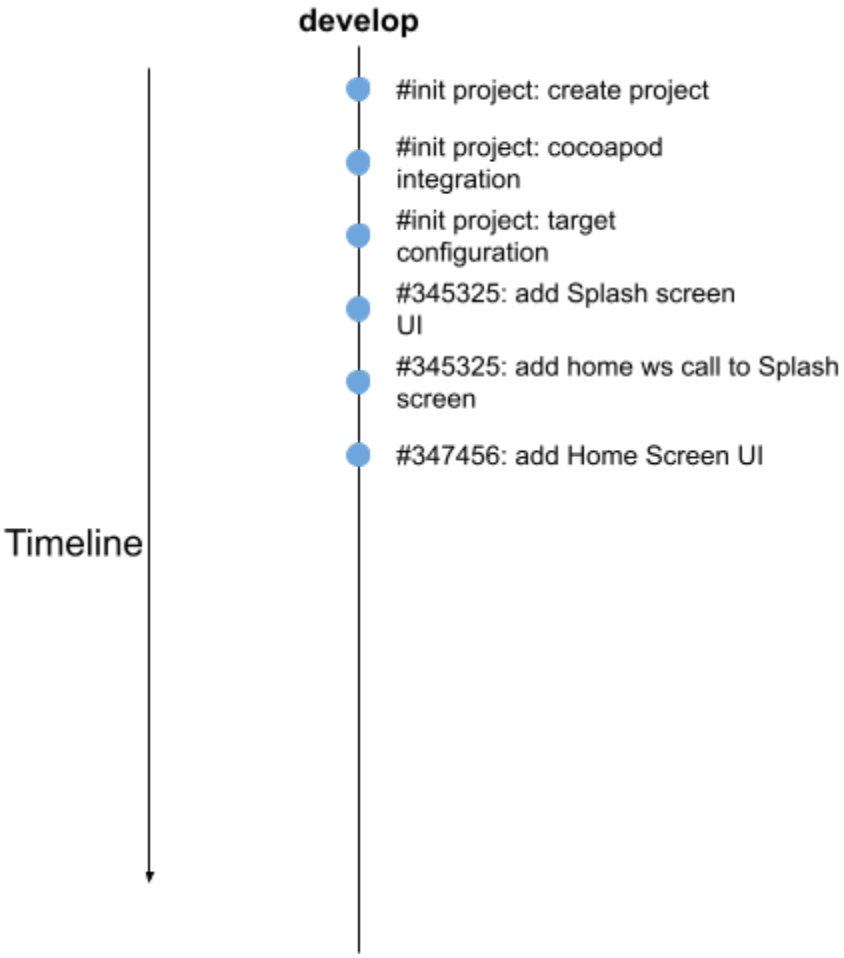
Commit Early and Often please

Flow:

On prend l'exemple d'un projet du zéro.

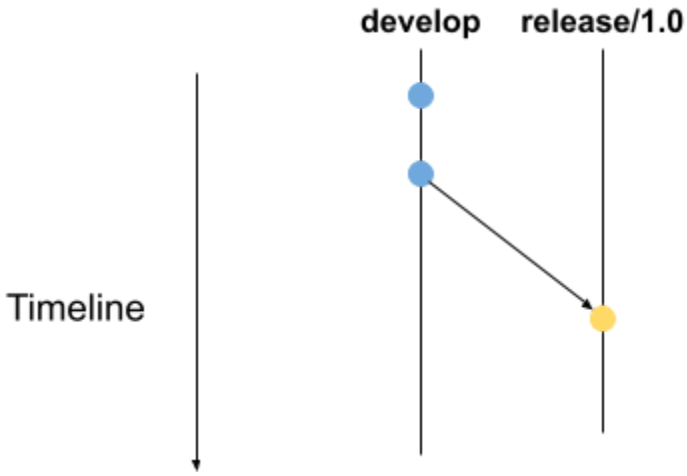
Initialisation du projet:

On commence par la création de la branche develop:



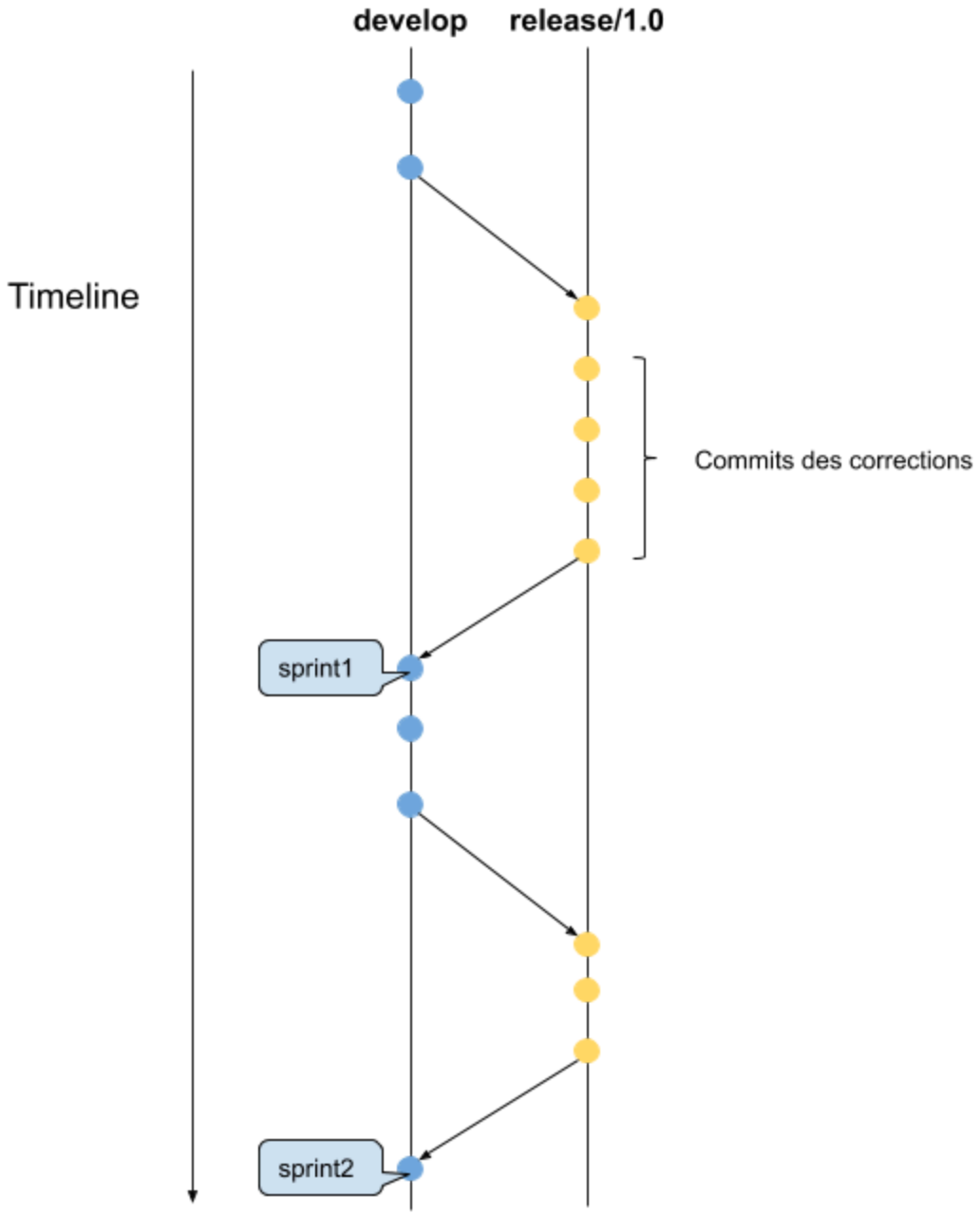
Livraison du sprint:

À la livraison du sprint, on crée une nouvelle branche release



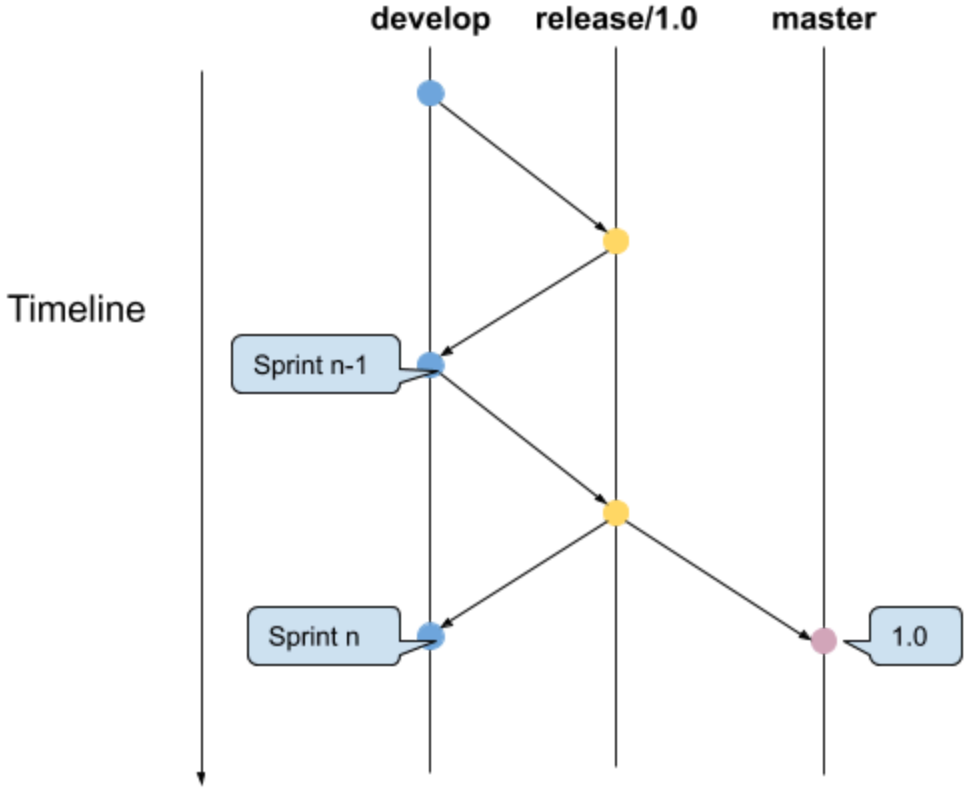
Validation du sprint:

Après tous les corrections seront sur la branche release jusqu'à la validation du sprint. À ce moment, on merge **release** sur **develop** et on ajoute un tag qui indique la fin du sprint. Ensuite on commence le développement du sprint suivant sur **develop**.


develop
release/1.0
Timeline
Commits des corrections
sprint1
sprint2

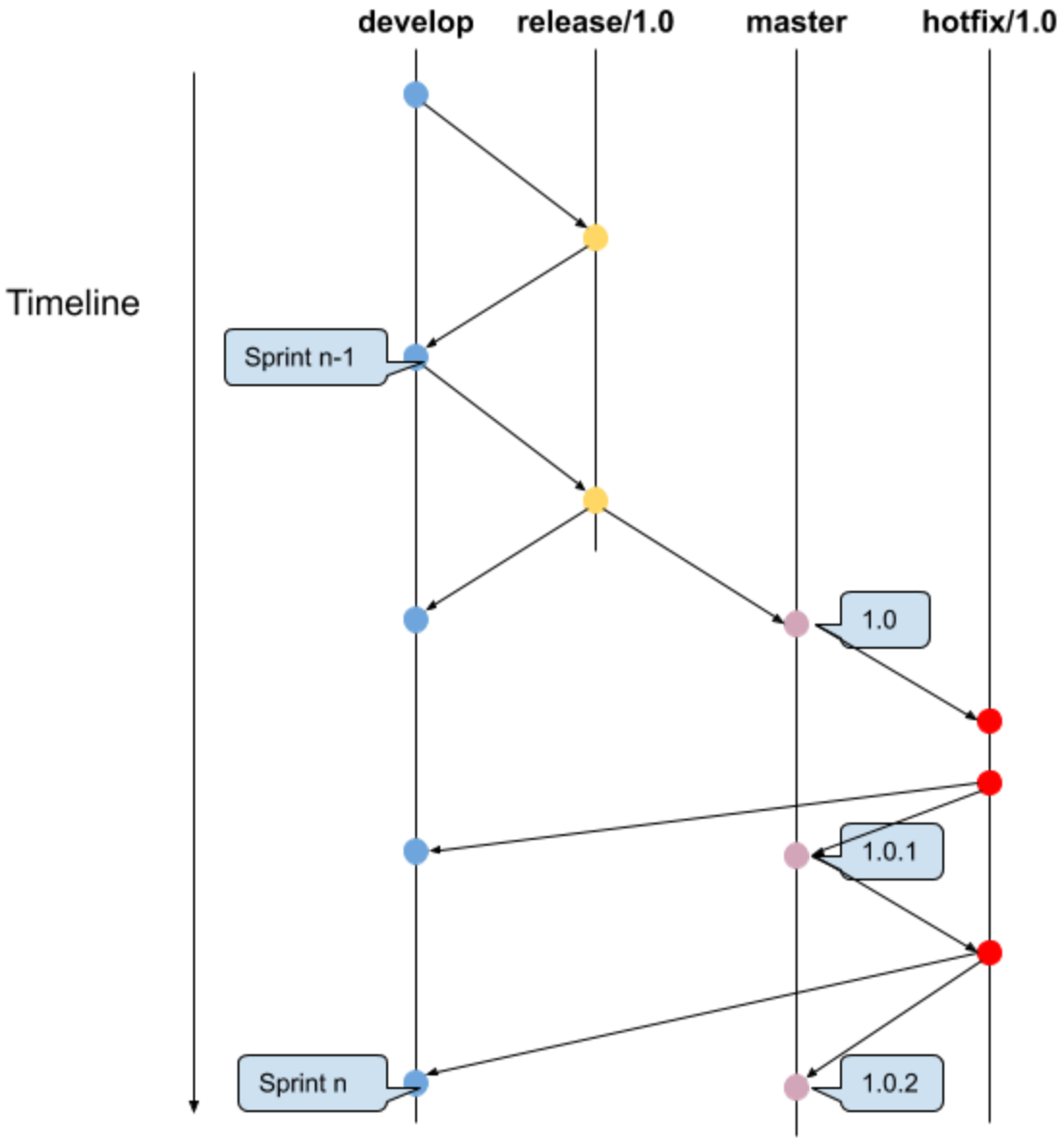
Mise en production:

Après la fin la mise en prod d'une version, on merge la branche release sur master. Notez bien qu'après les corrections ne sont faite ni sur **develop** ni sur **release**. De préférence on stoppe le développement des nouvelles fonctionnalités en attendant la validation de la version en prod



Corrections des bugs après la mise en prod:

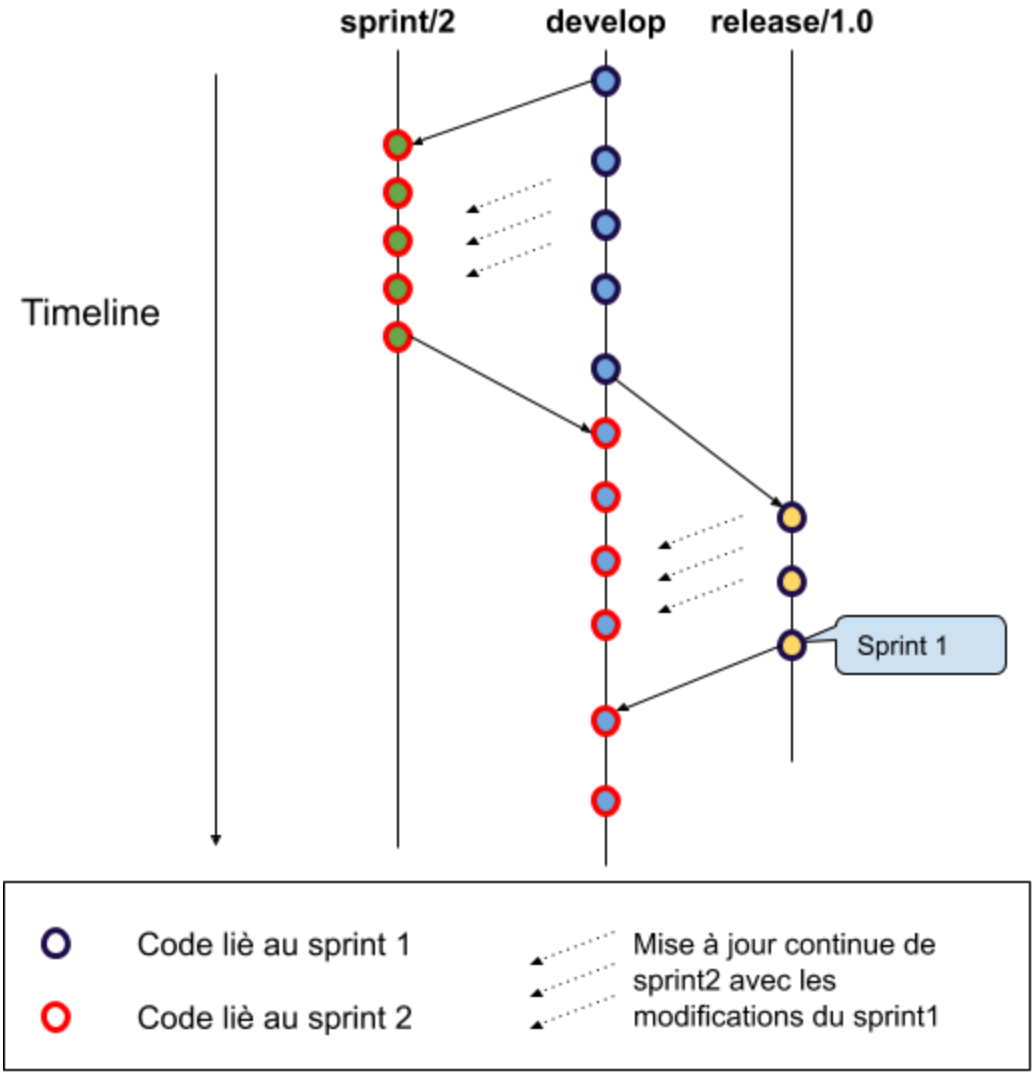
En cas ou il y a des corrections à faire sur la version en prod, on le fait sur une nouvelle branche **hotfix**. Après la validation de ces corrections, on remerge sur **master** et on met un nouveau tag en incrémentant l'indice de la **hotfix** dans la version. En merge sur **develop** aussi en déplaçant le tag du sprint final.



Le même flow se répète pour les prochaines versions mais on utilisant des nouvelles branches release et hotfix.

Parallèles sprints:

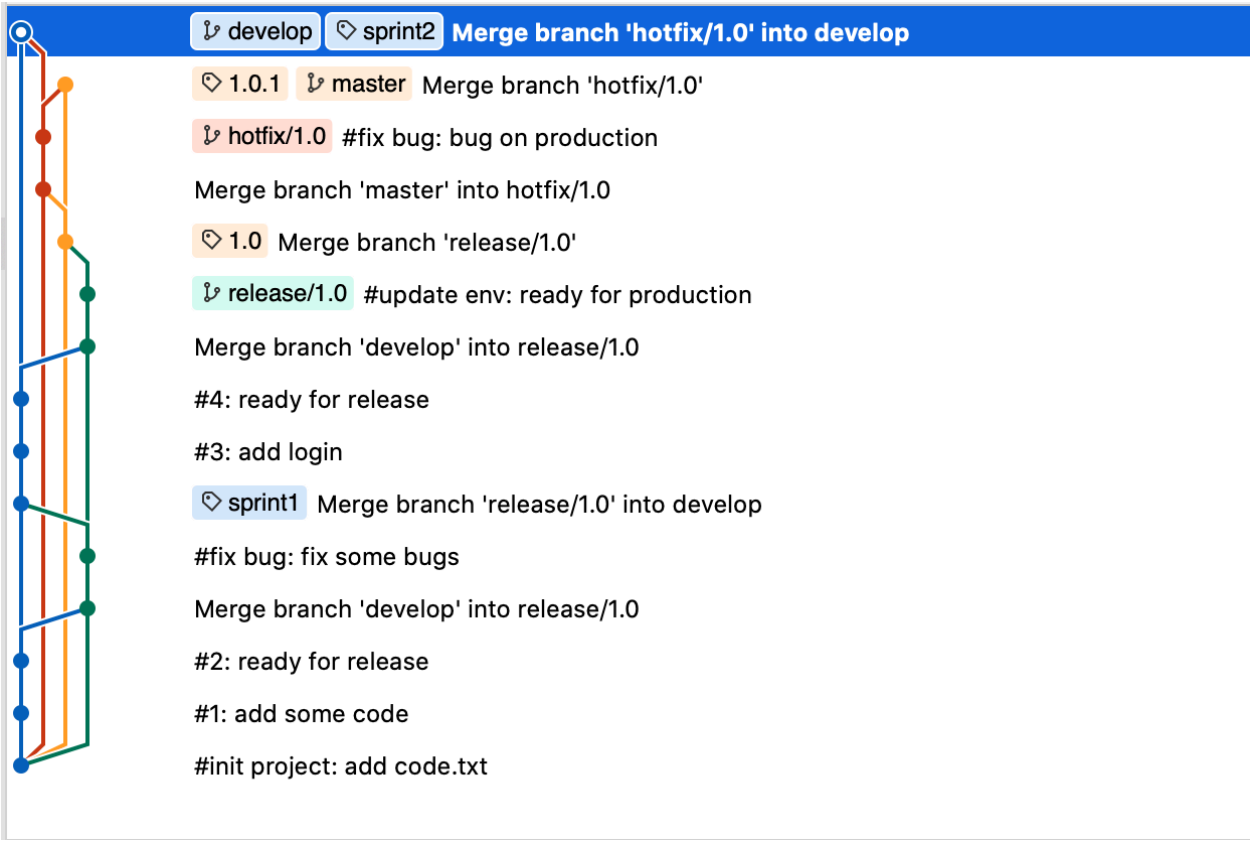
En cas de besoin, on peut lancer deux sprints en parallèles, le sprint en cours sera toujours sur la branche develop, par contre le nouveau sprint sera sur une nouvelle branche. En retourne sur **develop** dès que le sprint en cours est fini. On essaye de limiter la différence entre la branche **develop** et la branche sprint pour ne pas avoir beaucoup des conflits au moment du merge.



Dans ce cas, il faut merger le code de sprint 1 vers sprint 2 (develop -> sprint ou bien release -> develop) après chaque commit lié au sprint 1. Si c'est pas possible, on le fait d'une façon quotidienne a la fin de la journée.

GitFlow Sample:

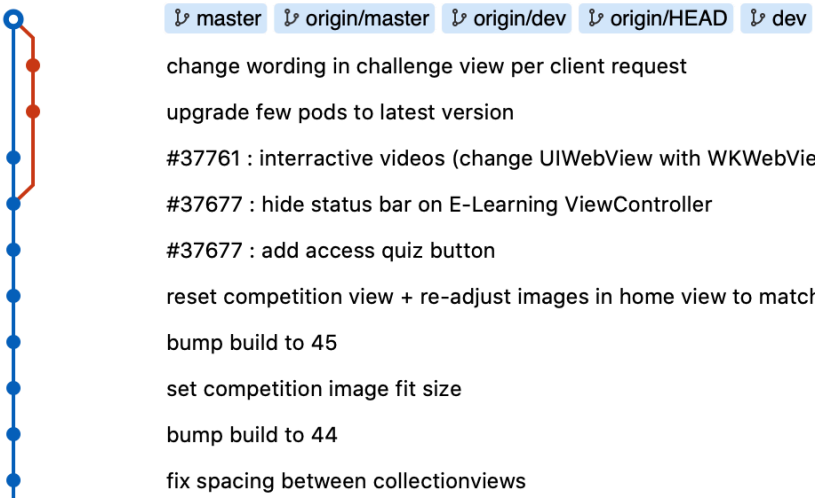
https://bitbucket.org/anypli/gitflow/commits/all



Autres Regles:

Les commits des merge:

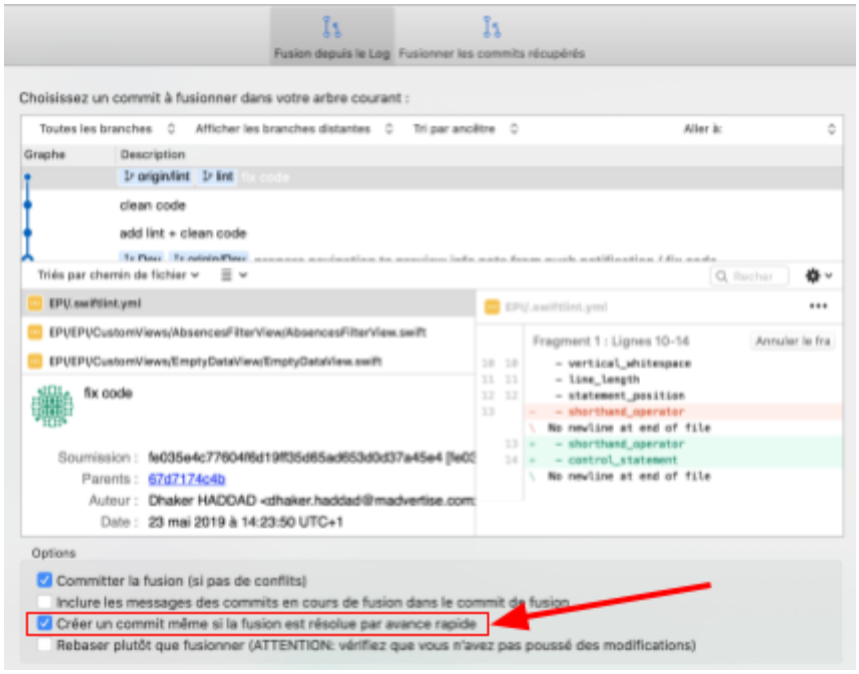
Par défaut, quand on merge on est pas sûre qu'une nouvelle commit vas etre creee, dans ce cas on trouves des commits liées à plusieurs branches. Et le flow ne sera pas lisible.
On peut pas savoir chaque commit sur quelle branche a été faite.



La solution: au moment du merge, il faut forcer le git de faire une nouvelle commit.
On peut le faire sur terminal avec:

```
git merge --no-ff
```

Ou bien avec sourceTree:



Versioning:

Version de l'application: M.m.C, dans le cas où **C** = 0, on peut mettre M.m
Version du build: M.m.C.B ou bien B

M: Version majeur.
m: Version mineur. **m** se remet à 0 pour chaque nouvelle version majeur
C: Corrections. **C** se remet à 0 pour chaque nouvelle version mineur.

B: Numéro du build. Si on utilise le model M.m.C.B, **B** se remet à 0 pour chaque nouvelle version de l'application.

Rôles:

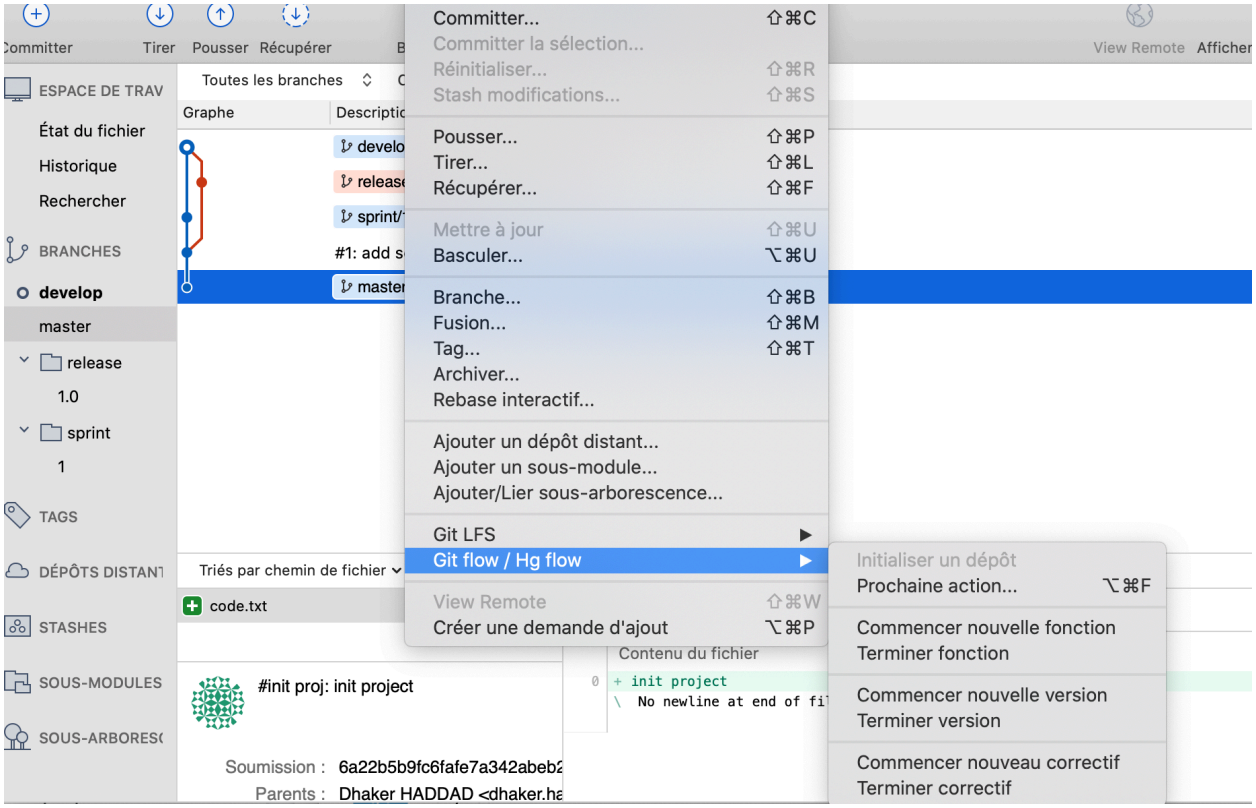
Le chef du projet est le responsable du suivie du Git Flow, c'est lui qui fait les merges entre les branches. Il doit notifier toute l'équipe en cas de passage d'une branche à une autre.

Le développeur doit savoir la branche adéquate pour chaque tâche.

Il faut concevoir une procédure de communication. Par exemple on peut ajouter dans les tickets le type des tâches (Task et bug pour develop, release_bug pour release,....).

On peut utiliser un plugin git flow, mais je pense que le faire manuellement sera mieux
https://github.com/nvie/gitflow
https://gist.github.com/JamesMGreene/cdd0ac49f90c987e45ac

Même dans sourceTree ca existe, mais il force une mise en prod après chaque release



Pull Request:

Pour la validation du code chaque tâche doit être sur une branche séparée, autre que les branches de git flow. Après il faut créer un Pull Request pour merger cette branche sur les branches de git flow (develop/release/hotfix). Sur chaque projet il faut avoir un dev responsable de la validation des Pull Request.