Anas elyassai

1)

Vincent Driessen a conçu Git-flow, une procédure de gestion des changements dans Git qui est supportée par plusieurs extensions Git pour gérer ce flux. L'idée principale derrière git-flow est de toujours avoir de nombreuses branches uniques, chacune dans un but particulier : "master, develop, feature, release et hotfix". Le développement de fonctionnalités ou de bogues progresse d'une "branche" à l'autre avant d'être publié.2)

2)

Inconvenant :

- Dans une stratégie de déploiement continu (ou quasi continu), le processus git-flow est difficile à contrôler.
- Complexité : Les cycles de vie de cinq types distincts de branche doivent être compris: `master`, `develop`, `feature`, `hotfix`, `release`
- Les branches master et develop ne sont pas nécessaires dans certains projets.
- Dans certaines circonstances, vous devez effectuer non pas une mais deux "merges" sur des branches distinctes à la suite.
- Régressions après merges automatisées : Git fait un excellent travail de gestion des merges par lui-même. S'il n'est pas en mesure d'effectuer la merge par lui-même, il faudra l'intervention de l'utilisateur pour résoudre les problèmes de merge.
- Il est difficile de garder une trace de l'historique de git**.**
- Workflow pas adapté à toutes les applications

3)

Avantage :

- git-flow fonctionne bien pour les produits avec un plan de publication plus traditionnel, comme les publications hebdomadaires.

- Développement isolé d'une nouvelle fonctionnalité
- Adapté aux méthodologies agiles
- La gestion des bugs

4/

Feature : Code en cours qui implémente une fonctionnalité qui sera incluse dans la future version du programme

Hotfix : Cette branche est l'endroit où nous corrigeons les bugs dans le code de la branche master (prod)

Anas elyassai

Release : Nous corrigerons les problèmes rencontrés lors du processus d'acceptation sur cette branche.

Develop : une branche permanente contenant la progression de l'application dans la file d'attente des versions dans son ensemble, avec des branches de fonctionnalités produites à partir de cette version et intégrées à celle-ci

Master : une branche permanente qui correspond à la version finale des ajustements effectués entre chaque version du programme comme référence De ce fait, elle correspond toujours à la version la plus récente du programme.

)/On doit se possitioner sur la branche Develop (git checkout develop) et on créer le tag

```
$ git tag -a <tag_name> -m "message"
$ git push –tags
```

6)

On doit reviens en Hotfix, Une fois terminé, le correctif de bogue doit être fusionné dans le maître, mais doit également être fusionné dans le développement, afin de garantir que le correctif de bogue soit également inclus dans la prochaine version. Ceci est complètement similaire à la façon dont les branches de version sont terminées.

Nous devons revenir à Hotfix. Une fois cela fait, le correctif du problème ne doit pas seulement être fusionné dans la branche principale(master), mais également dans la branche de développement, pour s'assurer que le correctif du bogue est inclus dans la future version. Ceci est assez similaire à la façon dont les branches de version sont fermées.

-PREMIERMENT, update master et tag the release.

En prend Exemple de version hotfix : » 1.2.1 »

```
$ git checkout master //Switched to branch 'master'
$ git merge --no-ff hotfix-1.2.1
$ git tag -a 1.2.1
```

-Ensuite, incluez également le correctif dans develop:

```
$ git checkout develop //Switched to branch 'develop'
$ git merge --no-ff hotfix-1.2.1.
```

-Enfin, supprimez la branche temporaire :

```
$ git branch -d hotfix-1.2.1 //supprime branch hotfix-1.2.1
```

7)

Git add . git commit --amend --no-edit

Git push origin releaseX.X.X

8)

-STASH : Pour éviter de perdre votre travail après un paiement, utilisez stash pour renvoyer votre espace de travail et l'état de l'index vers stash. Cela vous évite de faire un travail à moitié terminé.

-Commande qui permet d'avoir un retour arrière de git stash: git stash pop