Nom: Ghelani

Prénom: Akram Imed Eddine

Evaluation Outils de développement logiciel

1- Définir le workflow gitflow, pourquoi on l'utilise ?

Le workflow gitflow est une architecture qui vise à isoler les taches et de toucher le moins possible la branche master. On pourra dire donc que cette architecture est une architecture de branche dans un cadre particulièrement utile pour rationaliser la collaboration et faire évoluer les équipes. Gitflow nous donne un ensemble de règles sur la façon de travailler avec des branches Git individuelles en attribuant des rôles spécifiques et en définissant leur interaction. Parce qu'il a été développé dans l'optique d'une version de projet, il varie largement d'un projet à l'autre et devrait idéalement refléter nos propres besoins de développement.

Pourquoi on l'utilise ?

Car il est utile pour les projets qui adopte de l'intégration et la livraison continue, en plus il ne nécessite pas des nouveaux concepts ou d'autres commandes, au contraire il sert à isoler les taches et définir des rôles bien précis et la manière et le temps dans lequel ils doivent interagir,

En plus il permet d'utiliser les branches individuelles pour la préparation, maintenance et l'enregistrement des livraisons.

2- Les avantages du workflow gitflow ?

- Gitflow a été développé pour gérer le mécanisme de branchement avec une approche standardisée lors du développement de fonctionnalités, du traitement des versions et de la gestion des correctifs.
- L'utilisation de plusieurs branches séparées dans Git offre une certaine flexibilité mais devient complexe. C'est facile avec gitflow.
- Gitflow permet aux développeurs d'accélérer le processus grâce à une structure de branches familière.
- Une seule commande pour faire plusieurs choses à la fois.
- Le changement de branche est facile.
- Gardez le dépôt et le processus propre et bien rangé.

3- Les inconvénients du workflow gitflow ?

Les pull requests, qui ralentit considérablement le rythme de développement. Souvent, les développeurs qui examinent le code ont une autre tâche à accomplir, ce qui signifie que vous devez généralement attendre que votre code soit examiné.

4- Les définition et l’utilité des branches suivantes:

Feature:

La branche feature “fonctionnalité” est le type de branche le plus courant dans le flux de travail Git. Elle est utilisée pour ajouter de nouvelles fonctionnalités au code.

Lorsque on travaille sur une nouvelle fonctionnalité, on démarre une branche de featureà partir de la branche develop , puis on fusionne les modifications dans la branche de développement lorsque la fonctionnalité est terminée et correctement révisée.

Hotfix:

Dans le flux Git, la branche hotfix est utilisée pour traiter rapidement les changements nécessaires dans votre branche main.

La base de la branche hotfix doit être la branche principale master et doit être fusionnée (merger)à la fois dans les branches principale main master et de développement develop. Il est essentiel de fusionner les modifications de la branche de correction hotfix dans la branche de développement pour garantir la persistance de la correction lors de la prochaine publication de la branche principale.

Release:

La branche release doit être utilisée lors de la préparation de nouvelles versions de production. En général, le travail effectué sur les branches release concerne les finitions et les bugs mineurs spécifiques à la publication d'un nouveau code, avec un code qui doit être traité séparément de la branche principale de développement.

Develop:

La branche develop est créée au début d'un projet et est maintenue tout au long du processus de développement. Elle contient du code de pré-production avec des fonctionnalités nouvellement développées qui sont en cours de test.

Les fonctionnalités nouvellement créées doivent être basées sur la branche de développement, puis fusionnées à nouveau lorsqu'elles sont prêtes à être testées.

Master:

L'objectif de la branche principale master dans le flux de travail Git est de contenir du code prêt pour la production qui peut être publié.

Dans le flux Git, la branche masterest créée au début d'un projet et est maintenue tout au long du processus de développement. La branche peut être étiquetée à différents commits afin de signifier

différentes versions ou versions du code, et d'autres branches seront fusionnées dans la branche principale après avoir été suffisamment vérifiées et testées.

5/ Donner les commandes git pour créer un tag, sachant que vous êtes sur la branche Develop:

Les tags on les utilises sur les branches release:

Donc il va falloir se mettre quel branch release qui correspond à la branche develop avant de créer le tag.

Supposant qu'il s'agit de release1 est la release correspondante

On est sur develop

Git add .

Git commit –m "just a release"

Git rebase release1 //pour se mettre au niveau

Git push origin develop

//on valide le pull request (mergé)

git merge --no-release1

```
$ git tag <tag_name>
```

6/La correction des bugs:

On commence par stasher les modifications actuelles:

Git add .

Git stash

-On crée la branche du hotfix :

git checkout -b hotfix-x.x.x master

-Ensuite,on fait notre correction

Git add .

git commit -m "problème fixé"

Git push origin hotfix-x

-Enfin on merge dans master et dans develop

git checkout master

git merge --no-ff hotfix-x.x.x

git tag -a x.x.x -am "message"

git checkout develop

git merge --no-ff hotfix-x.x.x

-à la fin on supprime la banche du hotfix

git branch -d hotfix-x.x.x

Et on revient sur la branche où on était avec un checkout.

7/ Donner les commandes git à exécuter après la validation de la branche release pour passer en prod:

//on est sur la branche release

Git add .

git commit --amend --no-edit

//on fait rebase à travers la branche de

Git rebase develop

Git push origin releaseX.X.X

8/git stash:

Elle enregistre temporairement les changements apportés à la copie locale de travail pour que pour passer à d'autres taches en urgences par exemple puis revenir et les réappliquer par la suite. Le stashing est pratique quand on a besoin de changer rapidement de contexte et de travailler sur autre chose, mais que vous êtes en plein dans un changement de code et que n'êtes pas tout à fait prêt à commiter, il faut noter qu'un git stash n'envoie pas l'enregistrement aux serveurs.

donner la commande qui permet d'avoir un retour arrière de git stash

```
git stash pop
ou :
```

```
git stash apply --index
```

Si on veut garder l'état des fichiers:

```
git stash apply --index
```