Université Paris 8

Examens final outils libres

Javnier, 2022

Contents

1 Réponses

1.1 Questions 1: Définir le workflow gitflow, pourquoi on l'utilise

Workflow: Un workflow (flux de travail en français) est un procédé qui consiste a divisé un travail en étapes. Ces étapes sont ensuite developpées et des bilans sont généralement fait de façon quotidienne ou hebdomadaire au sein de l'équipe. C'est un procédé très utilisé dans les projets orientés production et livraison travaillant en Agile. Le workflow gitflow lui répond à la problématique de gestion de gros projets. En effet lorsqu'un projet grandit et qu'il y a beaucoup de développeur il devient difficile de gérer correctement:

- Version actuelle
- Developpement de la prochaine version
- Correction des bugs
- Mise en produciton
- Etc...

Le workflow gitflow permet d'avoir une meilleure organisation pour ce genre de projets. Pour cela il utilise une branche principale de production (main), une branche develop qui va être la branche qui va se merger à une branche production(release), mais qui va aussi être la branche branches parents de toutes les petites branches features qui vont être crées. Par exemple la feature "donner l'heure" va être la branche fille de develop et une fois la feature créer elle sera merge avec develop. Enfin il y a la branche hotfix qui permet la correction de bug rapidement et qui est mergé avec la main et develop.

1.2 Questions 2: Quels sont les avantages du workflow gitflow

Le workflow gitflow a les avantages suivants:

- Permet une meilleure organisation pour des gros projets orientés production.
- Permet de developper une nouvelle version tout en ayant une production.
- Permet une répartition plus claire des équipes, car on peut facilement attitrer un rôle precis au développeurs pour une branche feature préçise.
- Permet d'avoir une meilleure atomicité au niveau du developpement des features. Par exemple un git classique va créer une branche develop et va mettre toutes les features dessus. Alors que là on peut faire une branche par feature ce qui permet un meilleur contrôle et une meilleure visibilité au sein du projet

1.3 Questions 3: Quels sont les inconvénients du workflow gitflow

Le workflow gitflow a les inconvénients suivants:

- Gros risques de conflits, car on est amené a souvent avoir une grosse branche develop avec beaucoup de fonctionalités non mergées par rapport à la branch master.
- Nécessite une très bonne organisation
- Nécessite une forte productivité de l'équipe afin de rentre son usage "rentable" par rapport à un git classique
- Architecture compliquée
- Impossibilité de rebase si l'on souhaite garder une représentation propre.

1.4 Questions 4: Définir et donner l'utilité des branches : Feature, Hotfix, Release, Develop, Master

Voici la définitions des branches suivantes:

- Feature: Dans un gitflow workflow une branche feature est une branche créer à partir de la branche develop. La branche feature contient une fonctionalité qui à terme va être merge avec la branche develop

Hotfix: Une branche Hotfix est une branche ayant la branche master(main) pour parent et permettant la correction rapide d'un bug problèmatique. Elle est ensuite merge à la branche main, realse et develop

Realease est la branche "production). Une fois qu'on estime que develop contient ce qu'il faut pour la livraison on créer une branche dédiée (créer à partir de main) qui va contenir la livraison. Cette branche est utilisée aussi pour du code review et est potentiellement amenée a changée donc on la merge non seulement avec master, mais aussi avec develop afin que develop ait les potentiels changements de la code review etc...

La branche develop est créer à partir de master et permet de créer des branches features qui seront merge avec elle. Elle contient donc toutes les features de la dernière version. Elle push cette feature vers la branche release qui elle même va push avec develop et master

Master la branche master est la branche principale ayant pour enfant la branche hotfix et develop, elle correspond à la version mise en production actuellement (la version live).

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

Voici les commandes permettant de créer un tag en partant du principe que nous sommes dans develop et que l'on souhaite aller dans master:

- git checkout master - On bascule vers master
- git tag - a mon-tag -m "Tag de la branche master" - On ajoute un tag a master
- git add . - Add, commit, push
- git commit -m "Tag added"
- git push

1.6 Questions 6: Vous êtes sur une branche Feature en train de finaliser un développement, on vous informe qu'il y a un bug de prod à corriger très rapidement. Donner les commandes git pour corriger le problème de prod en respectant le workflow git.

Voici les commandes a utilisées afin de corriger le bug (On suppose qu'on est dans la branche develop et que hotfix n'est pas encore créée):

- git checkout master - On bascule vers master
- git branch hotfix - On créer la branche hotfix
- git checkout hotfix - On bascule vers la branche hotfix
- Correction du bug - On corrige le bug
- git checkout master - On bascule vers master
- git merge hotfix - On merge hotfix vers master
- git checkout develop - On bascule vers develop

- git merge hotfix - On merge hotfix vers develop
- git branch -d hotfix - On supprime la branche hotfix
- git checkout master - On bascule vers master
- git tag -a v1.1 -m "Version 1.1 (1.0 without bug)" - On tag le nouveau numéro de version de main
- git add . - Add, commit et push
- git commit -m "Hotfix cleaned"
- git push

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

On suppose donc ici que la branche release a subi un code review, a été modifiée et est sur le point de passer en production. Voici donc les commandes a executées:

- git master - On bascule vers master
- git merge release - On merge release avec master
- git checkout develop - On bascule vers develop
- git merge release - On merger release avec develop
- git branch -d release - On supprime release
- git checkout master - On bascule vers master
- git tag -a v2.0 -m "Release of versin 2.0" - On tag le nouveau numéro de version de main - On tag la nouvelle version
- git add . - Add, commit, push
- git commit -m "Release OK"
- git push

1.8 Questions 8: A quoi sert la commande git stash, donner la commande qui permet d'avoir un retour arrière de git stash

Le git stash permet d'enlever toutes les modification enregistrer dans un git add afin d'avoir un repertoir de tavail propre en cohérence avec le commit.
Si l'on souhaite réappliquer les modification enlevées par le git stash on fait un git stash-apply.