Règles de Versionning GIT

Repositories et Forks

Le projet principal sera hébergé sur un repository GitHub central, disponible pour être forké par tous les développeurs. Ce repository central servira uniquement d'upstream et ne sera en aucun cas le lieu de travail des développeurs. Chaque développeur devra forker ce repository pour créer leur propre copie (repo origin) où ils effectueront leurs développements.

Upstream

L'upstream est le repository principal du projet. Les développeurs doivent configurer l'upstream pour rester synchronisés avec les dernières modifications du projet. Les mises à jour depuis l'upstream doivent être intégrées régulièrement dans le repo origin du développeur pour éviter les conflits et maintenir la cohérence du code.

Origin

L'origin est le repository personnel du développeur, créé en forkant l'upstream. Tout le travail de développement doit être effectué dans ce repository. Une fois les modifications terminées, fonctionnelles et conformes, le développeur peut envoyer une pull request vers l'upstream. Les pull requests doivent être soumises uniquement lorsque le travail est complètement testé et prêt pour la revue.

Convention de Nommage Angular

Pour assurer la clarté et la cohérence, nous utiliserons la convention de nommage Angular pour les branches et les commits :

- **Branches** : Les noms de branches doivent suivre le format `type/description` (e.g., `feature/login-page`, `bugfix/header-crash`).
- **Commits** : Les messages de commit doivent être structurés comme suit :
  - **type** : type de modification (e.g., `feat`, `fix`, `docs`, `style`, `refactor`, `test`, `chore`).
  - **scope** : contexte de la modification (e.g., `login`, `header`, `api`).
  - **subject** : description concise de la modification.

Exemple : `feat(login): add user authentication`

Utilisation de GitFlow

Nous adopterons la méthodologie GitFlow pour structurer nos branches de développement. Voici les différentes branches utilisées :

- **main** : Contient le code en production. Toute modification ici doit être soigneusement validée.
- **develop** : Contient le dernier code de développement intégré. C'est ici que les nouvelles fonctionnalités et corrections de bugs sont fusionnées avant d'être validées pour la production.

- **feature/** : Branches de fonctionnalités créées à partir de `develop` pour le développement de nouvelles fonctionnalités.
- **release/** : Branches de release créées à partir de `develop` lorsque nous préparons une nouvelle version. Elles permettent de tester et de stabiliser les nouvelles fonctionnalités avant de les fusionner dans `main`.
- **hotfix/** : Branches de correction rapide créées à partir de `main` pour corriger des bugs critiques en production. Une fois résolues, ces corrections sont fusionnées dans `main` et `develop`.
- **bugfix/** : Branches de correction de bug créées à partir de `develop` pour corriger des bugs trouvés lors du développement.

Validation des Pull Requests

Toutes les pull requests doivent être validées par le chef de projet ou le coordinateur, ainsi que par l'architecte logiciel. Ce processus assure que le code est conforme aux standards du projet et que la nomenclature est correcte. Si une pull request ne respecte pas toutes les conditions et conventions établies, elle sera rejetée et renvoyée au développeur pour corrections.

Commandes Git Flow

Pour faciliter l'utilisation de GitFlow, voici un rappel des commandes principales :

1. **Initialisation de GitFlow :**

```
git flow init
```

2. **Démarrer une nouvelle fonctionnalité :**

```
git flow feature start nom_de_la_fonctionnalité
```

3. **Terminer une fonctionnalité :**

```
git flow feature finish nom_de_la_fonctionnalité
```

4. **Démarrer une nouvelle release :**

```
git flow release start numéro_de_version
```

5. **Terminer une release :**

```
git flow release finish numéro_de_version
```

6. **Démarrer un hotfix :**

```
git flow hotfix start nom_du_hotfix
```

7. **Terminer un hotfix :**

```
git flow hotfix finish nom_du_hotfix
```

8. **Démarrer une correction de bug :**

```
git flow bugfix start nom_du_bugfix
```

9. **Terminer une correction de bug :**

```
git flow bugfix finish nom_du_bugfix
```

Commandes Git de Base

Pour assurer une gestion efficace des versions et des branches, voici un rappel des commandes Git de base :

1. **Cloner un repository :**

```
git clone https://github.com/organisation/projet.git
```

2. **Vérifier l'état du repository :**

```
git status
```

3. **Ajouter des fichiers pour le commit :**

```
git add chemin/du/fichier
```

ou pour ajouter tous les fichiers modifiés :

```
git add .
```

4. **Faire un commit :**

```
git commit -m "message de commit"
```

5. **Pousser les modifications vers l'origin :**

```
git push origin nom_de_la_branche
```

6. **Récupérer les modifications depuis l'upstream :**

```
git fetch upstream
```

7. **Fusionner les modifications depuis l'upstream :**

```
git merge upstream/nom_de_la_branche
```

8. **Changer de branche :**

```
git checkout nom_de_la_branche
```

9. **Créer une nouvelle branche :**

```
git checkout -b nom_de_la_branche
```

10. **Supprimer une branche :**

```
git branch -d nom_de_la_branche
```

11. **Lister toutes les branches :**

```
git branch
```

12. **Voir l'historique des commits :**

```
git log
```

Processus de Synchronisation avec Upstream

1. **Ajouter l'upstream :**

```
git remote add upstream https://github.com/organisation/projet.git
```

2. **Récupérer les mises à jour de l'upstream :**

```
git fetch upstream
```

3. **Fusionner les mises à jour de l'upstream dans develop :**

```
git checkout develop
git merge upstream/develop
```

En suivant ces règles et méthodologies, nous garantissons un flux de travail structuré et une qualité de code élevée, facilitant la collaboration et la maintenance du projet.