Version Control (Contrôle de Version)

1. Git

Définition

Git est un système de contrôle de version distribué qui permet de suivre les modifications apportées au code source au fil du temps. Il facilite la collaboration entre les développeurs et permet de gérer des versions multiples d'un projet.

Concepts Clés

- **Branches** : Les branches permettent de travailler sur des fonctionnalités ou des corrections de bogues isolément sans affecter la branche principale (généralement appelée `main` ou `master`).
- **Merge** : Le merge (fusion) est l'action de combiner les modifications d'une branche dans une autre. Cela permet d'intégrer les changements d'une fonctionnalité développée dans une branche vers la branche principale.
- **Rebase** : Le rebase permet de déplacer ou de combiner une série de commits d'une branche vers une autre, ce qui peut rendre l'historique des commits plus linéaire et propre.
- **Pull Requests (PR)** : Les PR sont des demandes de fusion qui permettent aux développeurs de proposer des modifications à la branche principale. Cela inclut généralement une révision de code avant la fusion.

Exemple de Commandes Git

```
# Cloner un dépôt
git clone <url-du-depot>

# Créer une nouvelle branche
git checkout -b nouvelle-fonctionnalite

# Ajouter des modifications au staging
git add .

# Commiter les modifications
git commit -m "Ajout d'une nouvelle fonctionnalité"

# Fusionner une branche dans la branche principale
git checkout main
git merge nouvelle-fonctionnalite

# Rebaser une branche
git checkout nouvelle-fonctionnalite
git rebase main

# Envoyer les modifications sur le dépôt distant
git push origin nouvelle-fonctionnalite
```

```
# Créer une Pull Request sur GitHub
# (Cela se fait généralement via l'interface web de GitHub)
```

---

2. Trunk-based Development

Définition

Le Trunk-based Development est une approche où tous les développeurs travaillent sur une seule branche principale (le "trunk"). Cela favorise l'intégration continue et réduit le risque de conflits de fusion, car les modifications sont régulièrement fusionnées dans la branche principale.

Avantages

- **Moins de conflits de fusion** : Comme les développeurs fusionnent fréquemment leurs modifications, il y a moins de risques de conflits lorsque les fonctionnalités sont intégrées.
- **Intégration continue** : Cette approche encourage des intégrations fréquentes, ce qui signifie que le code est toujours en état de fonctionnement.
- **Simplicité** : Moins de branches signifie moins de complexité dans la gestion du code.

Exemple de Workflow

1. Les développeurs créent de petites fonctionnalités ou corrections de bogues.
2. Chaque développeur travaille sur leur code localement.
3. Ils effectuent des commits fréquents sur la branche principale.
4. Les tests automatisés sont exécutés pour vérifier que le code fonctionne toujours après chaque commit.
5. En cas de problème, il est plus facile de diagnostiquer les régressions.

---

3. CI/CD (Intégration et Déploiement Continu)

Définition

CI/CD est une pratique qui combine l'intégration continue (CI) et le déploiement continu (CD). Cela permet d'automatiser le processus de livraison du code, du développement au déploiement, en s'assurant que les nouvelles modifications n'introduisent pas de régressions.

Concepts Clés

- **Intégration Continue (CI)** : Les développeurs intègrent fréquemment leurs modifications dans un dépôt partagé, où des tests automatisés sont exécutés pour détecter les erreurs rapidement.
- **Déploiement Continu (CD)** : Cela va au-delà de la CI, où chaque changement qui passe les tests est automatiquement déployé dans un environnement de production ou de préproduction.

Exemple de Configuration de CI/CD avec GitHub Actions

Voici un exemple simple de configuration de CI/CD pour une application Node.js utilisant GitHub Actions :

```yaml
# .github/workflows/ci-cd.yml
name: CI/CD Pipeline

on:
  push:
    branches:
      - main # Déclencher le pipeline lors des push sur la branche principale

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Setup Node.js
        uses: actions/setup-node@v2
        with:
          node-version: '14' # Version de Node.js à utiliser

      - name: Install dependencies
        run: npm install # Installer les dépendances du projet

      - name: Run tests
        run: npm test # Exécuter les tests du projet

      - name: Deploy
        run: |
          # Commandes pour déployer l'application
          echo "Déploiement de l'application"
```

Avantages du CI/CD

- **Détection rapide des erreurs** : Les tests automatisés permettent de détecter rapidement les erreurs, ce qui facilite leur correction.
- **Livraison rapide** : Les nouvelles fonctionnalités peuvent être déployées rapidement, offrant ainsi une meilleure expérience utilisateur.
- **Moins de risques** : Les déploiements fréquents et automatisés réduisent les risques de régressions en s'assurant que chaque changement est testé avant d'être déployé.

Ces concepts de contrôle de version et de CI/CD sont essentiels pour assurer un développement efficace et collaboratif.