Cours : CI/CD & Bonnes pratiques Git

Introduction au CI/CD

CI/CD signifie **Intégration Continue** (Continuous Integration) et **Déploiement Continu** (Continuous Delivery/Deployment). Ce sont des pratiques essentielles pour automatiser et fiabiliser le développement logiciel.

- **Intégration Continue (CI)** : fusionner régulièrement les modifications de code dans une branche partagée, avec des tests automatiques.
- **Déploiement Continu (CD)** : automatiser la livraison du code validé vers un environnement de test, puis de production.

Pourquoi utiliser le CI/CD ?

- **Détection rapide des erreurs** grâce aux tests automatiques.
- **Livraison plus fréquente et fiable** du logiciel.
- **Réduction du stress des mises en production** (moins d'erreurs manuelles).
- **Amélioration de la collaboration entre développeurs**.

Pipeline CI/CD typique

1. **Push** du code vers un dépôt Git.
2. **Build** automatique (compilation, packaging).
3. **Tests automatisés** (unitaires, intégration).
4. **Déploiement** vers un environnement staging/test.
5. **Déploiement** vers production après validation.

Bonnes pratiques Git

1. Utiliser des branches dédiées

- `main` ou `master` : branche stable et déployée en production.
- `develop` ou `test` : branche d'intégration où on teste les nouvelles fonctionnalités ensemble.
- Branches fonctionnalités (`feature/xyz`) : une branche par nouvelle fonctionnalité ou correction, issue de `develop`.
- Branches correctives (`hotfix/xyz`) : pour corriger rapidement des bugs en production.

2. Travailler avec des Pull Requests (ou Merge Requests)

- Permettent la revue de code avant fusion.
- Encouragent la discussion et la qualité du code.
- Intègrent souvent des validations automatiques (tests).

3. Commits clairs et atomiques

- Un commit = une modification logique.
- Messages de commit explicites et standardisés (ex : `[FEATURE] Ajout de la page contact`).

4. Intégrer les tests dans le workflow

- Exécuter les tests localement avant push.
- S'appuyer sur CI pour valider automatiquement les branches.

5. Ne jamais pousser directement sur `main`

- Toujours passer par une PR depuis une branche `feature` vers `develop`, puis de `develop` vers `main`.

Exemple de workflow Git + CI/CD

```
main (production)
   ^
   |
develop (intégration)
  ^        ^
  |        |
feature/x  feature/y (nouvelles fonctionnalités)
```

- Tu crées une branche `feature/x` à partir de `develop`.
- Tu développes et testes localement.
- Tu fais une PR vers `develop`.
- CI lance les tests sur `develop`.
- Après validation et intégration, tu fusionnes `develop` dans `main`.
- CD déploie automatiquement `main` en production.

Outils courants CI/CD

- **GitHub Actions** (GitHub)
- **GitLab CI/CD** (GitLab)
- **Jenkins**
- **CircleCI**
- **Travis CI**

Bonnes pratiques pour la configuration CI/CD

- Automatiser tous les tests possibles.
- Garder les pipelines rapides (exécuter les tests longs séparément).
- Mettre en place des règles de merge (ex : tests obligatoires, revue obligatoire).
- Surveiller et analyser les échecs de build pour réagir vite.

Résumé

Concept	Rôle clé
CI	Intégration et tests automatisés à chaque push
CD	Livraison automatique et rapide vers test et production

Concept	Rôle clé
Git main	Branche stable en production
Git develop/test	Branche d’intégration des nouveautés
Feature branch	Développement isolé d’une nouvelle fonctionnalité
Pull Request	Validation et revue du code avant fusion