講義：CI/CDとGitのベストプラクティス

CI/CDの概要

CI/CDは、**継続的インテグレーション（CI）**と**継続的デリバリー／デプロイメント（CD）**の略です。ソフトウェア開発の自動化と信頼性向上のために不可欠な手法です。

- **継続的インテグレーション（CI）**：コード変更を共有ブランチに定期的に統合し、自動テストを実行する。
- **継続的デリバリー／デプロイメント（CD）**：検証済みコードをテスト環境、さらに本番環境へ自動で配信する。

CI/CDを使う理由

- **自動テストによる早期エラー検出**
- **より頻繁かつ信頼性の高いリリース**
- **本番デプロイ時のストレス削減**（手作業のミスを防ぐ）
- **開発者間のコラボレーション向上**

一般的なCI/CDパイプライン

1. コードをGitリポジトリに**プッシュ**
2. **自動ビルド**（コンパイル、パッケージング）
3. **自動テスト**（ユニットテスト、結合テスト）
4. ステージング／テスト環境への**デプロイ**
5. 承認後、本番環境への**デプロイ**

Gitのベストプラクティス

1. 専用ブランチの使用

- **`main`または`master`**：本番用の安定ブランチ
- **`develop`または`test`**：新機能の統合用ブランチ
- 機能ブランチ（`feature/xyz`）：`develop`から分岐し、1機能ごとに1ブランチ
- 修正ブランチ（`hotfix/xyz`）：本番のバグを即時修正

2. プルリクエスト（またはマージリクエスト）を使う

- マージ前にコードレビューを実施
- 議論とコード品質向上を促進
- 自動バリデーション（テストなど）を統合

3. 明確で一貫性のあるコミット

- 1コミット = 1つの論理的な変更
- わかりやすく統一されたメッセージ（例：`[FEATURE] お問い合わせページの追加`）

4. ワークフローにテストを組み込む

- プッシュ前にローカルでテストを実行
- CIを使ってブランチを自動的に検証

5. `main` に直接プッシュしない

- 必ず`feature`ブランチから`develop`へのPRを経て、`develop`から`main`へマージ

Git + CI/CDのワークフロー例

```text main（本番） ^ | develop（統合） ^ ^ | | feature/x feature/y（新機能） ```

よく使われるCI/CDツール

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

CI/CD設定のベストプラクティス

- 可能な限りテストを自動化
- パイプラインを高速化（重いテストは別に実行）
- マージルールを設定（例：必須テスト、レビュー必須）
- ビルド失敗を監視・分析し、素早く対応

まとめ

概念	役割
CI	プッシュごとの統合と自動テスト
CD	テスト・本番への自動デプロイ
Git main	本番用の安定ブランチ
Git develop/test	新機能の統合ブランチ
Featureブランチ	機能ごとの個別開発
プルリクエスト	マージ前のコードレビューと検証
```