入門版の次に読む

Eclipseで学ぶ
実務GitHubワークフロー

ブランチ / Pull Request / レビュー / マージ

この説明書のゴール

mainを直接変更せず、作業ブランチで安全に開発する

Pull Requestでレビューを受け、確認後にmainへ統合する

競合・レビュー修正・CI失敗にも落ち着いて対応する

対象: コミット・プッシュ・プルを理解したGitHub初心者

入門版と実務版の違い

02

実務では「mainを常に動く状態に保つ」ための仕組みが追加されます。

入門版

mainで作業

→ コミット

→ プッシュ

実務で多い形

作業ブランチ

→ Pull Request

→ レビュー → main

追加される4つの仕組み

ブランチ

作業ごとに分けた安全な作業場所

Pull Request

変更をmainへ入れてよいか相談する場所

コードレビュー

別の人が内容・安全性を確認

CI

自動ビルド・自動テストで事故を検知

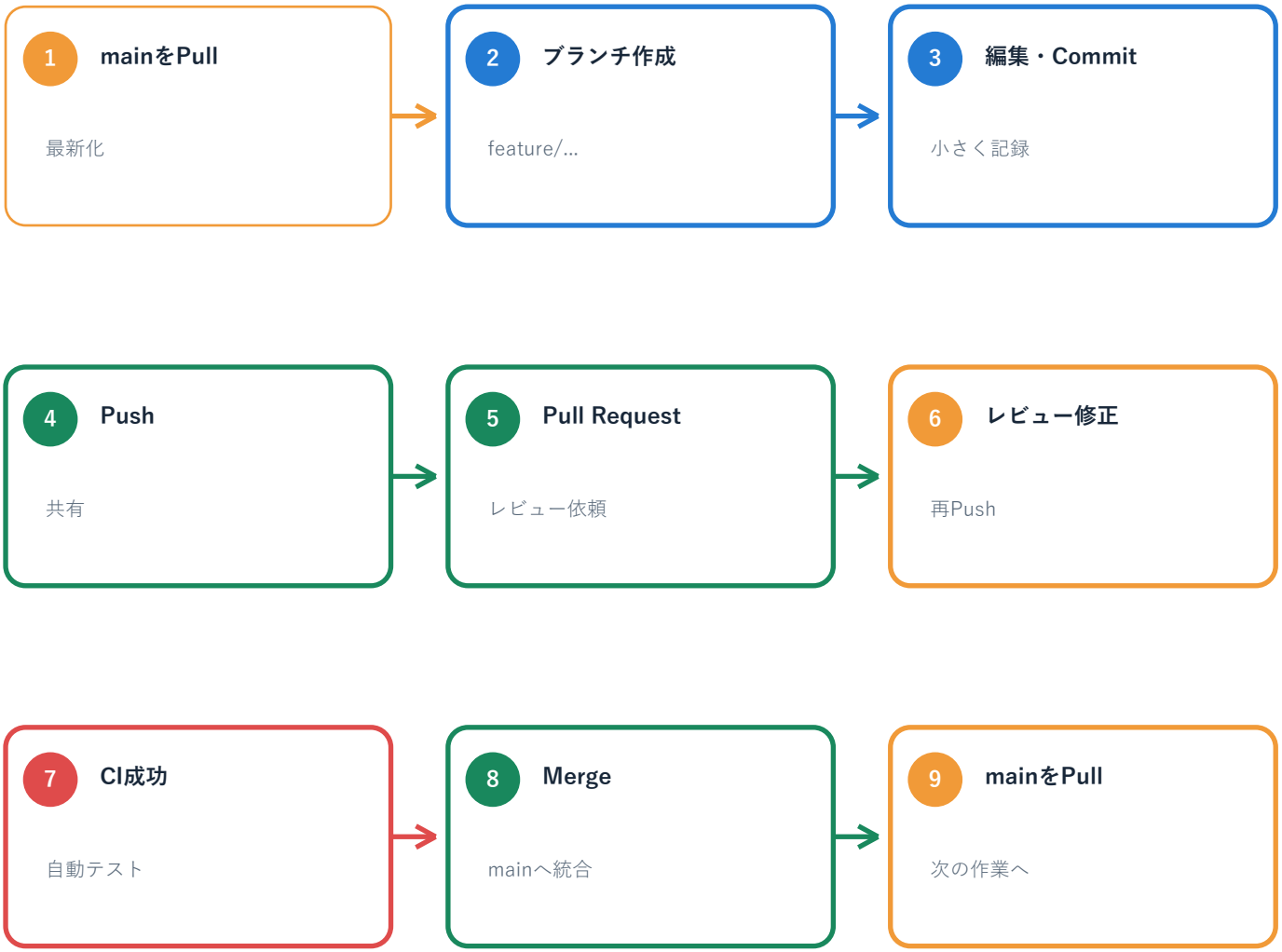
基本原則

mainは「みんなが使う完成版」。未確認の作業を直接入れない。

実務の全体フロー

03

最初に地図を見てから、各操作を順番に練習します。



チームで先に決める

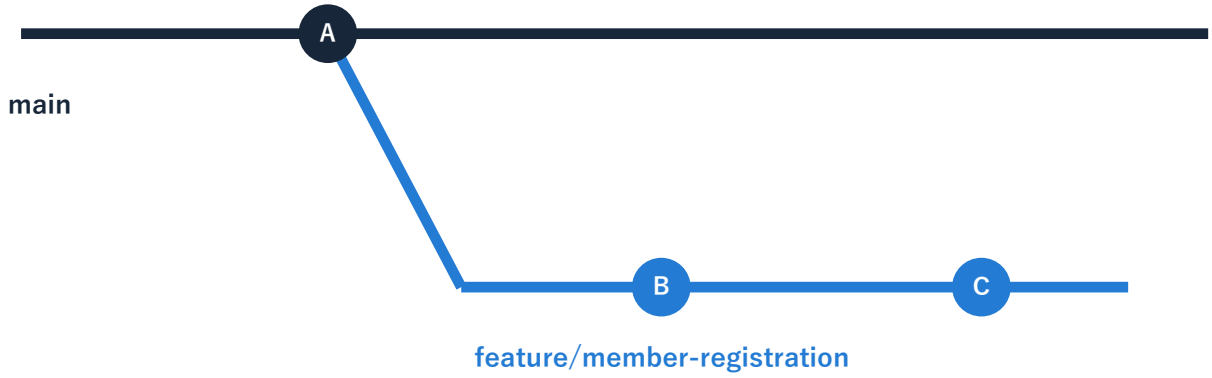
ブランチ名、レビュー担当、マージ条件、緊急修正の手順。

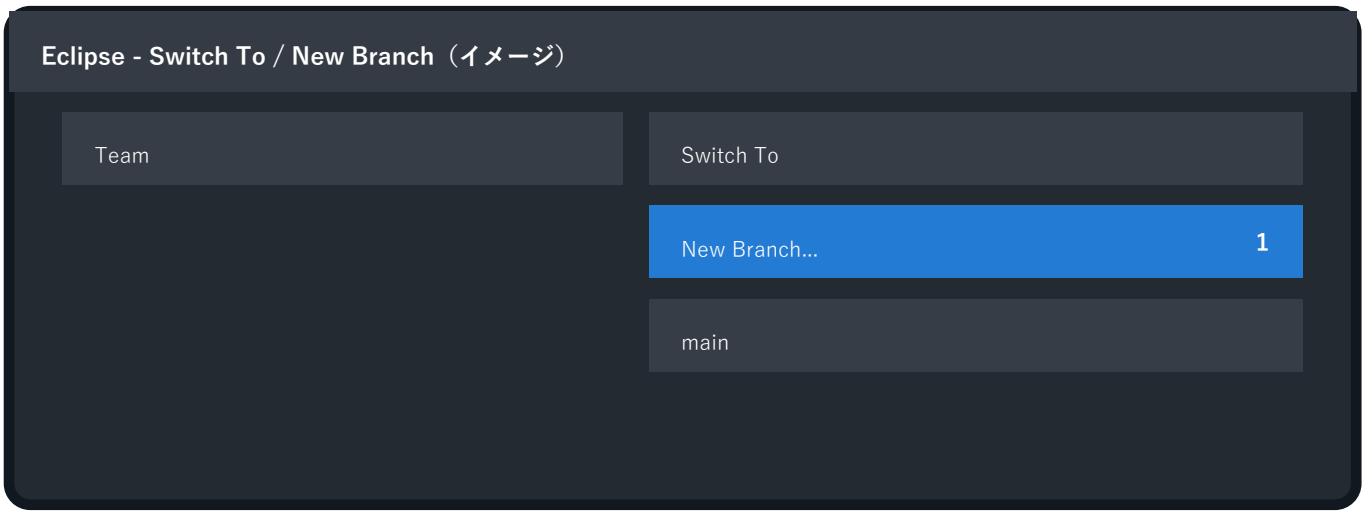
ブランチ: 作業専用の場所を作る

04

例: 会員登録を担当するなら feature/member-registration

ブランチのイメージ





操作

プロジェクトを右クリック → Team → Switch To → New Branch...

名前例: feature/member-registration　作成元: 最新のmain

1ブランチ = 1目的。複数機能を混ぜない。

ブランチ名とコミットの実務ルール

05

履歴を見ただけで、目的が分かる名前にします。

feature/機能名 新機能

例: feature/cart

fix/不具合名 通常の修正

例: fix/login-validation

docs/内容 文書変更

例: docs/setup-guide

refactor/対象 挙動を変えない整理

例: refactor/order-service

良いコミット 「カート数量の在庫チェックを追加」

避けたいコミット 「修正」「作業中」「いろいろ変更」

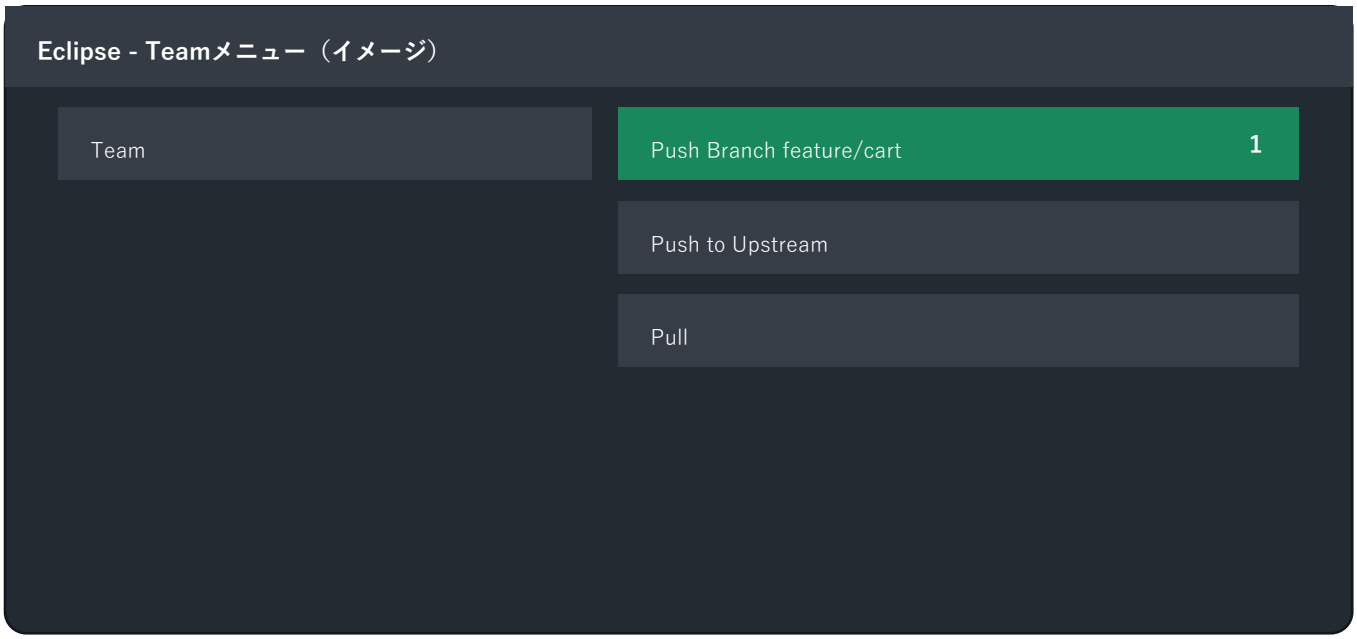
目安

1つの説明で表せる変更を1コミットにする。

作業ブランチをGitHubへプッシュ

06

初回プッシュ後、GitHub上にも同じ名前のブランチができます。



2

初回プッシュ

Push Branch “feature/...” を選び、送信先を確認します。

3

2回目以降

同じブランチなら Push to Upstream またはプッシュ操作。

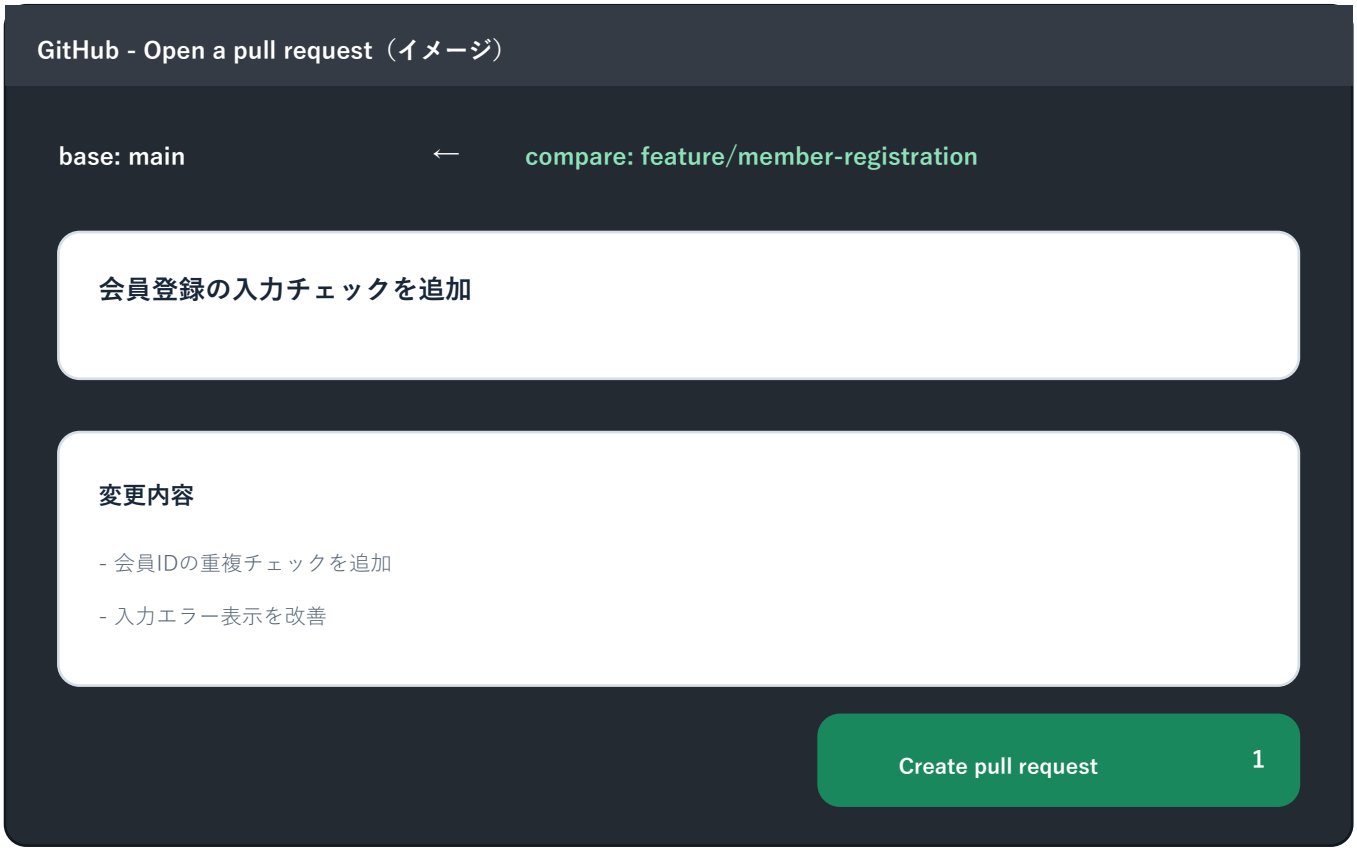
4

GitHubで確認

mainではなく、作業ブランチへコミットが届いたか確認。

Pull Request: mainへ入れる前の相談 07

GitHubのWeb画面で作成します。これは「プル」とは別物です。



説明欄に書くこと

1. 何を変更したか　2. なぜ必要か　3. 確認方法　4. 気になる点

間違えやすい言葉

Pull = GitHubから変更を取得する操作

Pull Request = 自分の変更をmainへ取り込んでもらう依頼

コードレビュー: 別の人が変更を確認

08

目的は責めることではなく、品質をチームで守り知識を共有することです。

設計

既存の作り方と合っているか

正しさ

要件どおりで境界値も扱えるか

安全性

パスワードや権限に問題がないか

読みやすさ

名前・分割・コメントが分かりやすいか

テスト

変更を確認するテストがあるか

伝わるコメント例

「在庫が0の場合、この処理はどうなりますか？」

「共通メソッドへ分けると再利用しやすそうです」

避けるコメント

「ダメ」「普通こうする」だけで、理由や提案がない。

レビュー指摘を修正する

09

同じ作業ブランチで修正し、コミット・プッシュすればPRへ自動反映されます。

1

コメントを読む

意図が分からなければ、修正前に質問する。

2

Eclipseで修正

PRに使った同じブランチで作業する。

3

テスト

指摘箇所だけでなく、関連機能も確認する。

4

コミット・プッシュ

例:「レビュー指摘: 在庫0の処理を追加」

5

GitHubで返信

どのように直したかを簡潔に伝える。

ポイント　**PRを作り直す必要はありません。**

CI: 自動チェックの結果を読む

PR作成・プッシュをきっかけに、ビルドやテストを自動実行する現場があります。

✓ All checks have passed

マージ候補。レビュー承認も確認。

● Checks in progress

実行中。完了するまで待つ。

× Some checks were not successful

失敗。ログから最初の原因を探す。

CIが失敗したら

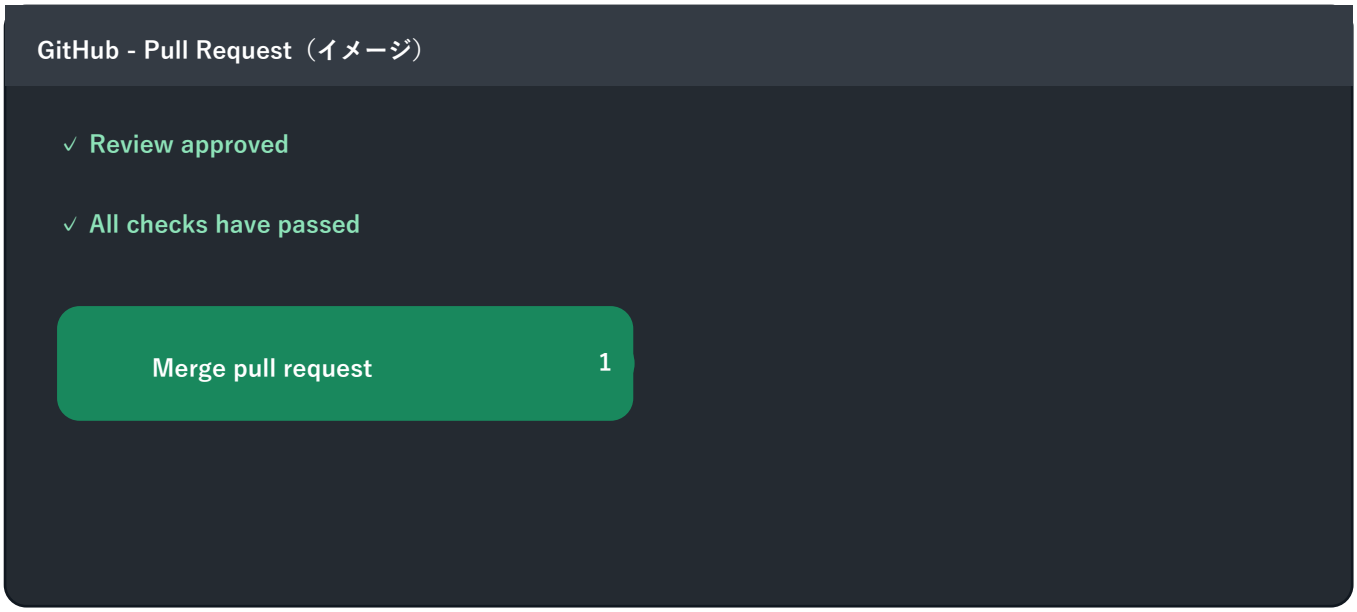
1. 失敗したテスト名を確認
2. ログの最初のERROR付近を読む
3. 自分のPCでもテスト
4. 修正して同じブランチへ再プッシュ

禁止 **テスト失敗を無視してmainへマージしない。**

承認後にマージし、後片付けする

11

チームのルールとCI条件を満たしてからmainへ統合します。



2

GitHubでブランチ削除

マージ済みの作業ブランチは削除して整理。

3

Eclipseでmainへ切替

Team → Switch To → main。

4

mainをPull

統合された最新版を自分のEclipseへ取得。

次の作業　**最新mainから新しい作業ブランチを作る。**

実務ワークフロー チェックリスト

チームの運用に合わせて印刷・共有してください。

作業開始

- [ ] mainへ切り替え、Pullで最新化
- [ ] 1目的の作業ブランチを作成
- [ ] 担当と完了条件を確認

Pull Request前

- [ ] 差分を自分でレビュー
- [ ] テスト成功を確認
- [ ] 分かるタイトル・説明・確認方法を記載

レビュー対応

- [ ] 質問と指摘を区別
- [ ] 同じブランチで修正して再Push
- [ ] 対応内容を返信

マージ前後

- [ ] 承認とCI成功を確認
- [ ] マージ後に不要ブランチを削除
- [ ] Eclipseのmainへ戻りPull

合言葉　「mainは守る、変更はPR、確認してMerge」