GitHubナレッジ共有型プロジェクト 統合運用ガイドライン

版数：2.0 (Integrated)

対象：プロジェクトマネージャー、テックリード、開発メンバー

目的

本ガイドラインは、プロジェクトにおける「コードとドキュメントの完全同期（Docs as Core）」を実現し属人化を排除することを目的としています。
GitHubの機能（Protection、Rules、Actions, Projects）を最大限に活用し、「**ルールを守らないとマージ・リリースできない**」仕組みを構築することで、品質と開発速度を両立させます。

システム構成概要

1. アーキテクチャ

- 管理：GitHub Project (全案件進捗), GitHub Repositories (ソース/ドキュメント)
- 構造：親リポジトリ (Portal) と子リポジトリ (App) を `Git Subtree` で連携
- 公開：AWS Amplify (または GitHub Pages) による自動デプロイ
- 認証：GitHub Pagesには認証機能がないため、情報の取り扱いに注意すること

2. リポジトリ定義

種類	リポジトリ名規則	役割	テンプレート	権限
親リポジトリ (Portal)	`<案件名>_portal`	ドキュメントポータル、案件全体管理、デプロイ	`temp_knowledge_portal`	直接Push禁止 (PR必須)

子リポジトリ (App)	`<アプリ名>_<言語>`	ソースコード、詳細仕様書	`temp_knowledge_app`	直接Push禁止 (PR必須)

セットアップ手順（管理者向け）

1. テンプレートリポジトリの準備（初回のみ）

組織全体のひな型となるテンプレートリポジトリを作成します。

1. **親テンプレート（ `temp_knowledge_portal` ）作成**
    - Privateリポジトリとして作成。
    - `.github/workflows/handle_app_update.yml` を配置（子からの更新通知を受け取り、 `git subtree pull` を実行）。
    - `mkdocs.yml` （基本設定のみ）、 `docs/index.md` （ポータル表紙）を配置。
    - `Settings` > `Template repository` にチェックを入れる。
2. **子テンプレート（ `temp_knowledge_app` ）作成**
    - Privateリポジトリとして作成。
    - `.github/workflows/notify_portal.yml` （親への更新通知・タグ情報を送信）を配置。
    - `mkdocs.yml` , `docs/index.md` を配置。
    - `Settings` > `Template repository` にチェックを入れる。

2. プロジェクト立ち上げ手順（案件開始時）

Step1. リポジトリの払い出し

1. `temp_knowledge_portal` から親リポジトリを新規作成します。
2. `temp_knowledge_app` から子リポジトリを新規作成します。
3. Secret設定：
    - 管理者のPersonal Access Token（PAT）を発行します。
    - 両方のリポジトリの `Settings` > `Secrets` > `Actions` に `PROJECT_REPO_PAT` として登録します。

Step2. 連携設定（ファイル修正&コマンド）

1. ワークフロー修正（子リポジトリ）：
    - `.github/workflows/notify_porttal.yml` 内の `repository` と `app_name` を修正してコミットします。

2. Subtree登録（親リポジトリ）：
    - ローカルで親リポジトリに対して、以下のコマンドを実行して登録します。

```
git remote add [アプリ名] [アプリのリポジトリURL]
git subtree add --prefix=apps/[アプリ名] [アプリ名] main --squash
git push origin main
```

3. 親リポジトリの `mkdocs.yml` にアプリへのリンクを追記します。

Step3. GitHub Projectの準備

1. `<案件名> Management Board` 等の名前で Template Project から作成します。
2. 作成した親・子リポジトリを紐づけます。

GitHub Project運用ルール

1. 必須フィールド設定（Custom Fields）

タスク管理の品質を保つため、以下のフィールドを設定します。

フィールド名	タイプ	選択肢・設定値	用途
System	Single Select	`Portal`, `<アプリ名>` ...	システム/アプリの識別
Doc Status	Single Select	`Not Needed` (リファクタリング等) `Required` (仕様変更あり) `Done` (完了)	ドキュメント更新状態の管理
Status	Single Select	`Todo`, `In Progress`, `Done`	進捗ステータス (自動遷移設定推奨)

2. 標準ビュー（Views）

以下の３つのビューを必ず作成し、運用状況を可視化します。

1. **Main Board（タスクリスト）**
    - 開発のメイン画面。 `Status` で列を作成。
    - カードには必ず `Doc Status` を表示させること。
2. **Roadmap（ガントチャート）**
    - `System` でグループ化し、アプリ間のスケジュール干渉を可視化する。
3. **Audit List（監査用テーブル）**

- 完了したタスクなのに、ドキュメント更新が終わっていないものを撲滅する目的を持ちます。
- Filter: `Status: Done` AND Doc Status: `Required`

ブランチ・タグ戦略

1. バージョン管理

バージョン	更新基準	Branchとの対応	例
Major	破壊的変更	手動昇格 (Discussion必須)	`v2.0.0`
Minor	機能追加	`feature/*` のマージ	`v1.1.0`
Patch	バグ修正・Docs	`bugfix/*`, `docs/*`, `hotfix/*`	`v1.0.1`

2. ブランチ命名規則

自動化ツールがバージョンアップの種類を判別できるよう、以下のPrefixを厳守してください。
マージ完了後のブランチは、GitHubの設定により、**即時削除**されます。

Prefix	用途	SemVer影響	使用例
`feature/`	新機能追加	Minor	`feature/login-api`
`bugfix/`	バグ修正	Patch	`bugfix/header-layout`
`docs/`	ドキュメントのみ修正	Patch	`docs/update-manual`
`hotfix/`	緊急修正	Patch (即時)	`hotfix/security-patch`
`chore/`	ビルド・設定変更	なし	`chore/update-deps`
`release/`	リリース準備	なし	`release/ver.1`

3. タグ保護ルール

- Pattern： `v*`
- 設定： 作成は管理者/Botのみ許可、削除は禁止。

開発プロセス

1. 開発フロー・チートシート

1. **タスク着手（Issue）：**

- GitHub Projectでタスクを作成し、 `Doc Status` を設定する。
- 仕様変更あり = `Required`
- リファクタリング等 = `Not Needed`

2. **ブランチ作成**
   - 命名規則に従ってブランチを切る。
3. **実装**
   - コード修正と同じブランチ・同じPRで、必ず `docs/` 配下のドキュメントも修正する
   - コードとドキュメントはセットでコミットする。
   - コミットメッセージ：
     - `<type>(<scope>): <subject>` （例： `feat(auth): ログイン機能追加` ）

2. Pull Request（PR）作成ルール

PRテンプレートを使用し、以下の要件を満たしてください。

- **Description に `Closes #<Issue番号>` を記述する。**
- **右サイドバーの Projects で該当プロジェクトを選択する。**
- ラベル： `enhancement` , `bug` , `documentation` 等を付与する。

3. レビュー・承認・マージルール（Branch Protection）

品質と履歴の健全性を保つため、システム的に以下のルールを強制します。

- `main` ブランチへの直接Pushの禁止
- CIチェックにて、**ビルド・テスト・Lint**が成功していること。
- 最低１名の承認が必要。セルフマージ禁止。（管理者は除く）
- ドキュメント確認：
  - 機能変更に対し、ドキュメントの修正・追記が含まれているか。
  - Amplify等のプレビュー環境で表示が崩れがないか。
- マージ戦略：
  - 子リポジトリの履歴を綺麗に保つため **Squash and Merge** を必須とする。

自動化パイプライン（CI/CD Architecture）

1. 自動連携フロー

Git Subtreeの手動操作を排除し、以下のフローで**自動同期・デプロイ**を行います。

1. **子リポジトリ**
   - PRマージ を トリガー にCIが起動。
   - Release Drafter 等の自動でタグ（ `v1.1.0` ）を付与し、リリースノートを作成。
   - Repository Dispatch により、親リポジトリへ「更新通知」と「バージョン情報」を送信。
2. **親リポジトリ**
   - 通知を受信し、自動ワークフローが起動。
   - `git subtree pull` を実行し、更新取り込み用 PR を自動作成（または自動マージ）。
   - 変更検知後、Docsを自動で **ビルド&デプロイ** します。
3. **トラブルシューティング（手動コマンド）**

   自動連携が失敗した場合のみ、以下のコマンドで手動同期を実行してください。

```
# 親リポジトリでの作業
# 必ず --squash をつけること
git subtree pull --prefix=apps/[アプリ名] [リモート名] main --squash
```

品質保証・ガバナンス

1. 禁止事項

- **親リポジトリでの直接編集の禁止**

  `apps/` 配下のファイルは、親リポジトリ上で編集することを禁じます。（単方向同期の原則）
- **手動タグ作成の禁止**

  バージョン管理の整合性が崩れるため、禁止します。
- **ドキュメントの後回し**

  `Doc Status: Required` のままタスクを完了にしてはいけません。

2. CODEOWNERS

重要な設定ファイル（ワークフロー定義等）は、特定管理者のみが変更承認できるように `CODEOWNERS` ファイルを配置して保護します。

利用開始時のワンポイント

https://github.com/tinayla696/temp_knowledge_portal

https://github.com/tinayla696/temp_knowledge_app

https://github.com/users/tinayla696/projects/5

このテンプレートセットを使うことで、人間がやるべきことは以下の３点だけになります。

1. テンプレートリポジトリから親・子リポジトリの作成。（ `Use this template` ）
2. 子リポジトリの `notify_portal.yml` 内の `repository` 名を書き換える。
3. 親リポジトリで `git remote add ...` と `git subtree add ...` を１回だけ実行する。

あとは、「**ブランチを切ってPRを出せば、リリースとportal更新まで全自動**」という世界が完成する。