ChatGPT

混乱ポイント一覧 (P0/P1/P2)

- **P0: リポジトリ状態の不一致** – 例：ローカルでコミットしたまま `git push` せず放置し、他者が同じブランチを更新。そのまま作業を続けると `git pull` 時に競合や非FAST-FORWARDエラーが発生し、履歴が分岐する。原因は「ローカルとリモートを同期せず、ブランチ保護ルール（必要なStatus Check）を無視したマージ」である。対策として、作業前に必ず `git pull` / `fetch` で最新化し、**コミット後速やかにpush**する [1] [2]。リモートに更新がある場合は `git pull --rebase` でマージ前に自動解決し、問題が起きたら `git merge --abort` で元に戻す [1] [3]。
- **P0: マージ競合の未解消** – 例：複数人が同一ファイルを更新し、`git merge` 時に「<<<<<<<」「=======」「>>>>>>>」マーカーが残ったままコミットされる。原因は「競合発生時に手動解決せずにマージ作業を完了してしまった」ことである。Gitでは競合時に自動でマーカーを挿入し、利用者が手動で選択・編集する必要がある [4] [3]。対策としては、競合したファイルを手動で編集し不要なマーカーを削除してから `git add` し（あるいは `git merge --abort` でマージ前の状態に戻して再試行）、コミットを完了させる [3] [4]。
- **P0: 強制プッシュ/直接コミットの誤用** – 例：`main` や `integrate` など重要ブランチへ直接 `git push --force` したり、PRを経由せずにコミットを上書きしてしまう。原因は「ブランチ保護ルールの不適用や運用手順違反」であり、これにより他者の作業が破壊される危険がある。GitHubのブランチ保護では**デフォルトで強制プッシュと削除を禁止**できる [5]。対策として必ず **PR経由でマージ** し、直接pushやforce pushは禁止する [5] [2]。
- **P1: 大規模・複数目的なPR混在** – 例：バグ修正、用語統一、フォルダ整理など異なる目的を1つのPRに混ぜる、または10ファイル以上の大きなPRにする。原因は「1ブランチ1目的の原則を守っていない」ことで、レビュー困難やリバート困難を招く。GitHub Flowでは「作業内容ごとに別ブランチを切り、短く説明的な名前を付ける」ことが推奨されている [6] [7]。対策として、**変更は最小差分に分割**し、1PR＝1タスク/1機能にする。
- **P1: ブランチ名・コミットメッセージの曖昧さ** – 例：ブランチ名が `fix` や `update` のまま作業、コミットメッセージが `修正` だけ。原因は「わかりやすい命名ルールがない」ため。GitHub Docsでは「短く説明的なブランチ名を付ける」「コミットも単一変更にまとめ説明的に書く」ことを推奨している [6] [8]。対策として、ブランチ名にタスクや機能名を含め、コミットメッセージには変更点を明確に記述する。
- **P2: ステータスチェックやレビュー漏れ** – 例：CI/VerifyチェックがFailしているのにマージしようとする、レビュー承認なしでマージしようとする。原因は「ブランチ保護ルール（Status Check必須、レビュー必須）の遵守不足」である。保護対象ブランチでは**必須チェックがすべて通過しないとマージ不可**となる [2]。対策として、PR作成時にCI/Verify結果やレビューを得たことを確認し、チェックが全てPassしてからマージする習慣を徹底する。

“迷いゼロ”運用フロー（チェックリスト）

- [ ] **最新状態の確認**：作業前に `git fetch` / `git pull` でリモートを最新化し、ローカル・リモートの差異を解消 [1]。
- [ ] **新規ブランチ作成**：`main` または `integrate` から機能／課題別にブランチを切り、ブランチ名は短く説明的に命名する [6] [7]。（例：`feat-xxxx`、`fix-xxxx`）
- [ ] **変更・コミット**：作業を行い、**1コミット＝1目的**で小さな変更単位にまとめる。コミットメッセージは内容を明確に記載する [8]。
- [ ] **プッシュ・同期**：ローカル変更をリモートにプッシュする前に、再度 `git pull --rebase` 等で同期しコンフリクトがないか確認。コンフリクトが発生したら手動解決または `git merge --abort` で一旦やり直す [1] [4]。

- [ ] **PR作成**：リモートでPull Requestを作成し、タイトル・説明に変更内容と目的、関連IssueやADRを記載する。PRテンプレートがある場合は必須項目（Verify結果、関連リンクなど）を埋める [9]。レビュワーを設定し、必要な承認を得る。
- [ ] **Fast Verifyの実行**：PRマージ前にFast Verify（4点チェック）を手動または自動で実行し、全てPASSを確認する [1] [4]。失敗があれば修正し再実行する。
- [ ] **コードレビュー**：レビュー担当者によりコードと証跡（実行ログ・Evidence）が適切であるか確認され、承認が得られていることを確認する。
- [ ] **マージ**：すべてのステータスチェック・レビューが完了したら、`main`または`integrate`ブランチへマージする。ブランチ保護ルールがある場合はそれに従い（例：Status Check必須、Linear History必須） [2]。マージ時は通常`--no-ff`（マージコミット付き）で統合する。
- [ ] **ブランチ削除**：マージ完了後、使用したフィーチャーブランチを削除する（GitHubで自動設定可能） [2] [5]。
- [ ] **リリース・証跡**：Release作業前に、最新のVerify結果と操作ログ・差分を含むEvidence Packを保存し、Release Gate条件が満たされているか確認する。

設計書に追記する文章案

- **MUST: Pull Request経由のみで変更を反映** – リポジトリの主要ブランチ（`main` / `integrate`）への変更は、必ずプルリクエストによるマージで実施する。直接の`git push`や`--force`オプション使用は禁止する [5] [2]。
- **MUST: ブランチ命名・粒度の徹底** – ブランチ名は短く明確な名前とし、1ブランチ1目的の原則に従う。無関係な変更は同一PRに混在させず、目的別にブランチを分けて作業する [6] [7]。
- **MUST: コミット規約の遵守** – コミットは小さな差分に絞り、一貫したスタイルで記述する。コミットメッセージは変更内容を説明的に記述し、必要に応じて関連IssueやADRへのリンクを含める [8]。
- **MUST: PR作成時の情報記載** – プルリクエストの概要には必ず変更内容・目的を記載し、PRテンプレートの項目（Verify結果や承認状況、参照Issue/ADRなど）に沿って情報を完備する [9]。
- **MUST: 事前検証の徹底** – PRマージ前にはFast Verify (4点検証) とCIテストを実行し、すべてPASSさせることを必須とする [1] [2]。検証に失敗した場合は問題を修正し、検証を再実行する。
- **MUST: 証跡保持** – すべての変更に対して実行ログ・検証ログ・差分などの証跡を生成し、`evidence/`ディレクトリ下に保存する。証跡は削除禁止とし、必要に応じて時系列でアーカイブ管理する [10]。
- **SHOULD: ブランチの最新化** – 大規模な差分・競合を避けるため、マージ前にベースブランチ（`main` / `integrate`）を自ブランチに取り込んで最新化しておく。GitHubの自動ブランチ更新機能（Require branches to be up to date）を活用することも推奨する [1] [2]。

Verify Gate の改善案

- **拡張子フィルタリング**：Fast/Full Verifyで検証不要なファイル（バイナリ、画像、マイナー言語ファイルなど）は検出対象から除外する。例えば`.gitignore`や検証スクリプト側で除外リストを定義し、検証時の誤検知を防ぐ。
- **削除ポリシーの統一**：ファイル削除操作は特例的に許可制とする。`sources/`以下を削除する場合は事前にADR承認とバックアップを必須化し、Verifyチェックでは削除操作を検知して許可済み削除か判定するルールを追加する。削除が必要な場合はGit歴からの完全抹消（`git filter-repo`等）とその証跡記録をワークフローに組み込む。
- **誤検知防止**：誤検知となる定型パターン（例：自動生成コードやライブラリのアップデート差分など）を検出ロジックに追加し、必要に応じてホワイトリストやフラグ付け機能を設ける。例えば禁止語彙チェックで誤検知するワードをリスト化して無視する。

- **命名規則の統一**：Verify結果レポートや証跡ファイルの命名規則を標準化する。例えば日時＋チェック名形式（例：`YYYYMMDD_HHMMSS_<checkname>.md`）やカテゴリ識別子を付与して管理性を高める [11]。
- **保持方針の明確化**：証跡ログは「追記のみ・削除禁止」とし、蓄積したログを整理するため最新3件を最新フォルダに配置する等の運用ルールを策定する [10]。古い証跡は年次アーカイブに移し、必要な履歴は保持しつつリポジトリの肥大化を抑制する。

最低限のトラブルシュート（症状→原因→手順）

- **症状**: `git push` 時に「non-fast-forward」や「updates were rejected」エラーが出る。
  原因:　リモートブランチに新しいコミットが存在し、ローカルブランチが古い状態のままプッシュしようとした。
  手順: `git pull --rebase` または `git pull` を実行してリモートの変更を取り込む。競合が発生した場合は該当ファイルを手動で編集し、`git add` してから再度 `git rebase --continue` または `git merge --continue` を実行。その後再度 `git push` する [1] [4]。

- **症状**: コードに「<<<<<<< HEAD」などの競合マーカーが残り、テストが通らない／マージできない。
  原因: マージ操作中に競合が発生し、競合マーカーの解消を行わずにコミットしてしまった。
  手順: `git merge --abort` で現在のマージを一旦中止し、該当ファイルを開いて `<<<<<<<` 〜 `>>>>>>>` のブロックを適切に編集・削除する。編集後に `git add <ファイル>` で解決済みとし、`git merge` または `git pull --rebase` を再実行する [3] [4]。最終的に `git commit` でマージコミットを作成する。

- **症状**: Pull Requestのマージがブロックされ、GitHub上で「Required status checks」やレビュー承認不足と表示される。
  原因:　ブランチ保護ルールにより、CIテストやVerifyチェックがFail、あるいは必要レビューが未完了のためマージ不可となっている。
  手順:　GitHubのPR画面でFail箇所を確認し、問題を修正して再度CI/Verifyを実行する。必要なレビュー承認が得られていない場合はレビュワーに依頼する。全てのチェックがPassし承認が得られたら、再度Merge操作を試みる [2] [1]。

- **症状**: Verify Gateが「FAIL」または証跡不足と判定される（証跡ファイルが存在しない）。
  原因: Verify実行前にローカル検証不足、またはVerifyを通さずにマージした可能性。
  手順: `checks/verify_repo.ps1`（または対応スクリプト）を手動実行し、Fast VerifyがPASSするよう修正を加える。ログが `evidence/verify_reports/` に出力されることを確認する。必要な4点（リンク/用語/Part整合/ソース整合）証跡が揃うまでこのVerify-Repairループを繰り返す [1] [4]。

---

[1] Getting changes from a remote repository - GitHub Docs
https://docs.github.com/en/get-started/using-git/getting-changes-from-a-remote-repository

[2] Merging a pull request - GitHub Docs
https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/incorporating-changes-from-a-pull-request/merging-a-pull-request

[3] [4] Git - Basic Branching and Merging
https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging

[5] About protected branches - GitHub Docs
https://docs.github.com/en/repositories/configuring-branches-and-merges-in-your-repository/managing-protected-branches/about-protected-branches

6 7 8 9 GitHub flow - GitHub Docs
https://docs.github.com/en/get-started/using-github/github-flow

10 11 DESIGN_MASTER_20260111_230526.txt
file://file_000000001fd471fd8b30c1ad532c4eb7