

## 1. 混乱ポイント一覧 (P0/P1/P2) : 例と原因

- **P0 (最高優先: 事故直結の混乱):** ブランチ戦略の曖昧さ (main/integrate/featの使い分け)。例: integrate/20260111のような日付ベースブランチが乱立し、featブランチとの併用でマージ競合が発生。原因: ドキュメント (Part14) の1Part=1Branch原則がGitHubのPRフローと整合せず、conflict marker (<<<<<<) が残る事故を招く (根拠: Git公式ドキュメントのmerge conflict説明 [https://git-scm.com/docs/git-merge#\\_how\\_conflicts\\_are\\_presented](https://git-scm.com/docs/git-merge#_how_conflicts_are_presented))。
- **P0: PR作成/マージのタイミング。** 例: ローカルで複数Partを編集後、GitHub PRで競合が発生し、証跡が失われる。原因: ローカル作業のコミット粒度が未定義で、GitHub同期が遅れ、再現性が低下 (根拠: GitHub DocsのPRガイド <https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests>)。
- **P1 (中優先: 再現性低下の混乱):** ローカル vs GitHubの使い分け。例: ローカルでVerifyを実行後、GitHubにpushせずに進むと、チーム共有の証跡が欠落。原因: ドキュメント (Part10) のVerify Gateがローカル実行前提だが、GitHubリポジトリの共有性を考慮せず、初心者がローカル限定作業で迷う。
- **P1: 証跡維持 (recent-3) の運用。** 例: evidence/verify\_reports/ の古いログが蓄積し、削除ポリシーがないためディスク圧迫。原因: Part12のEvidence管理が保存義務を強調するが、保持ポリシー (e.g., recent-3のみ保持) が未定義で、事故防止が不十分。
- **P2 (低優先: 初心者耐性の混乱):** Verify Gateの誤検知。例: 拡張子違い (.md vs .txt) でリンク切れ検知が失敗。原因: Part10の機械判定 (V-0001等) が拡張子厳格すぎ、初心者が誤検知を無視し事故を招く。
- **P2: リリース時のブランチマージ。** 例: integrateからmainへのマージでconflict markerが混入。原因: ドキュメント (Part13) のRelease Packageがマージ手順を詳細化せず、git-mergeの自動解決に頼りすぎる (根拠: Gitリリースノート v2.39.0以降のconflict改善 <https://github.com/git/git/blob/master/Documentation/RelNotes/2.39.0.txt>)。

## 1. 改善後の“迷いゼロ”運用フロー (チェックリスト形式)

・初期セットアップ (ローカル):

1. git clone <repo-url> でリポジトリをローカルに取得。
2. git checkout main でmainブランチに切り替え。
3. git pull origin main で最新同期。

・新機能/修正作業 (featブランチ、ローカル中心):

1. git checkout -b feat/<part-number>-<short-desc> (e.g., feat/part10-verify-fix) でfeatブランチ作成。
2. ローカルで最小差分編集 (1Part=1Branch原則)。
3. git add <files> && git commit -m "feat: <desc> (ADR-XXXX)" でコミット (最小粒度、ADR参照必須)。
4. pwsh ./checks/verify\_repo.ps1 -Mode Fast でFast Verify実行、PASS確認。
5. git push origin feat/<branch> でGitHubにpush。

・PR作成/レビュー (GitHub中心):

1. GitHub UIでPR作成 (base: integrate/<date> e.g., integrate/20260111、title: "Feat: <desc>")。
2. PR説明に変更理由/影響/Verifyログを記載。
3. レビュー依頼 (HumanGate承認必須)。
4. 競合発生時: git pull origin integrate/<date> --rebase でローカル再同期、conflict手動解決 (<<<<< marker全削除)。
5. PR承認後、GitHub UIでSquash Merge (conflict防止)。

・マージ/統合 (integrateブランチ、GitHub中心):

1. git checkout integrate/<date> でintegrateブランチ切り替え。
2. git merge --no-ff feat/<branch> でマージ (no-ffで履歴明確化)。
3. pwsh ./checks/verify\_repo.ps1 -Mode Full でFull Verify実行、PASS確認。
4. git push origin integrate/<date> でpush。

- ・リリース (mainブランチ、GitHub中心):
  1. git checkout main でmain切り替え。
  2. git merge --no-ff integrate/<date> でマージ。
  3. git tag v<yyyy.mm.dd> && git push origin v<yyyy.mm.dd> でタグ付け。
  4. evidence/verify\_reports/ にrecent-3ログのみ保持 (古いものはアーカイブ or 削除)。
- ・証跡維持:
  1. 各ステップ後、evidence/verify\_reports/YYYYMMDD\_HHMMSS\_<type>.md にログ保存。
  2. recent-3ポリシー: find evidence/verify\_reports/ -mtime +7 -delete で7日超削除 (cron推奨)。
- 1. 設計書へ追記する文章案 (そのまま貼れる)

## R-XXXX: ローカル vs GitHub 使い分け 【MUST】

- ・ローカル優先: 個人編集/コミット/Fast Verify (再現性確保のため最小差分)。
- ・GitHub優先: PR作成/レビュー/マージ/Release (共有証跡のためUI操作推奨)。
- ・同期ルール: 編集後即push、pull/rebaseで競合防止 (根拠: GitHub Docs <https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/addressing-merge-conflicts/resolving-a-merge-conflict-on-github>)。

## R-XXXX: ブランチ戦略 【MUST】

- ・main: 安定版のみ (リリース専用)。
- ・integrate/<yyyy.mm.dd>: 統合/テスト用 (日付ベース、週次作成)。
- ・feat/<part-number>-<desc>: 機能追加/修正用 (1Part=1Branch)。
- ・マージフロー: feat → integrate (Squash Merge) → main (no-ff Merge) (根拠: Git公式 [https://git-scm.com/docs/git-merge#\\_fast\\_forward\\_merge](https://git-scm.com/docs/git-merge#_fast_forward_merge))。

## R-XXXX: マージ競合防止 【MUST NOT】

- ・conflict marker (<<<<< / ===== / >>>>>) をコミット禁止。検出時即修復。
- ・防止策: git merge --abort で中止後、git pull --rebase 使用。自動ツール (git mergetool) 推奨 (根拠: Git ドキュメント [https://git-scm.com/docs/git-merge#\\_how\\_to\\_resolve\\_conflicts](https://git-scm.com/docs/git-merge#_how_to_resolve_conflicts))。

## R-XXXX: 証跡維持 (recent-3) 【MUST】

- ・evidence/verify\_reports/ にログ保存、recent-3保持 (最新3件のみ)。
- ・削除ポリシー: 7日超のログを自動削除 (cron or script)。
  1. Verify Gate 改善案 (誤検知/漏れ/証跡運用/命名/保持ポリシー)
    - ・誤検知対策: 拡張子を柔軟化 (.md/.txt/.ps1許可)、V-0001(リンク切れ検出) に--ignore-pattern '\*.bak' オプション追加。誤検知例: バックアップファイルのリンクを無視 (根拠: Git公式ignore <https://git-scm.com/docs/gitignore>)。
    - ・漏れ対策: Full Verifyにgit diff --check を追加 (whitespace/競合marker検出)。漏れ例: 未解決conflictをFail扱い。
    - ・証跡運用: PASS/FAIL両方を保存 (FAILはdebug用)、運用ループでFAILをRepair後PASSのみ採用。運用例: pwsh ./checks/verify\_repo.ps1 -LogLevel Debug。
    - ・命名: YYYYMMDD\_HHMMSS\_<verify-mode>\_<status>.md (e.g., 20260111\_230526\_Fast\_PASS.md)。一意性確保。
    - ・保持ポリシー: recent-3 (最新3件保持)、古いものはgit archive or 外部ストレージへ。ポリシー理由: ディスク節約、再現性確保 (根拠: GitHub Docsのarchive <https://docs.github.com/en/repositories/archiving-a-github-repository/archiving-repositories>)。
- 1. 最低限のトラブルシュート (症状→原因→手順)

- ・症状: マージ競合で<<<<< marker残存 → 原因: git mergeの手動解決漏れ → 手順: git diff --check で検出、vi <file> でmarker削除、git add/commit後再Verify。
- ・症状: Verify誤検知 (リンク切れ) → 原因: 拡張子違い or ignore漏れ → 手順: .gitignoreに \*.tmp追加、pwsh ./checks/verify\_repo.ps1 -Ignore '\*.tmp' で再実行。
- ・症状: 証跡ログ欠落 → 原因: 口一カル実行後push忘れ → 手順: git log --oneline で確認、evidence/ をadd/commit/push。
- ・症状: ブランチ乱立 → 原因: featブランチ削除忘れ → 手順: git branch -d feat/<old> (口一カル)、git push origin --delete feat/<old> (GitHub)。