Engineer

リポジトリー

ファイル群の各バージョンを記録して保持する場所

どのようなものか

リポジトリーは、会話ではほぼ常にrepoと略されます。バージョン管理システムがプロジェクトの全記録を保持する場所であり、各ファイルの記録済みのすべての状態、誰が何のために変更したか、そしてその履歴にたどり着くための名前を含みます。編集する作業ファイルは、その一つの見え方にすぎません。リポジトリーは、その背後にあるものすべてです。

Git自身の用語集は、リポジトリーを「参照の集合と、それらの参照から到達可能なすべてのオブジェクトを含むオブジェクトデータベースを合わせたもの」と、厳密に定義しています。二つの部分をそれぞれ見てみましょう。オブジェクトデータベースは内容を保持します。ファイルの各バージョン、各ディレクトリーの内容一覧、各コミットがオブジェクトとして保存され、その内容のハッシュを名前にします。Gitの解説書は、これをコンテンツアドレス型ファイルシステムと呼んでいます。値からキーを計算するキーバリューストアです。同じ内容の二つのファイルは一つのオブジェクトになり、どれほど小さな変更でも新しい名前を生みます。参照は、人が履歴を扱うための仕組みです。ブランチやタグという短い名前が、それぞれ一つのコミットを指し、履歴への入り口を作ります。

普段作業するリポジトリーの多くには、作業ツリーが付いています。エディターで開くファイルはプロジェクトのディレクトリーに置かれ、リポジトリー自体は、その隣の隠しディレクトリー `.git` にあります。ベアリポジトリーには、作業ツリーがまったくありません。ディレクトリー名は慣例として `.git` で終わり、履歴だけを保持します。サーバーが保持するのはこの形式です。そこでは誰もファイルを編集せず、利用者はそこへプッシュし、そこからフェッチするためです。Gitの用語集も、公開リポジトリーの提供者は通常ベア形式を公開すると説明しています。Gitの解説書は、既存のプロジェクトから `git clone --bare my_project my_project.git` で作る方法を示しています。

クローンには、Gitの設計がよく表れます。クローンは、サーバーにあるほぼすべてのデータを受け取ります。既定では、プロジェクトの全履歴にわたる、全ファイルの全バージョンを取得します。そのため、各クローンは完全なリポジトリーとなり、サーバーを失っても代わりになれます。引き継がれないものは少数ですが重要で、Gitの解説書も挙げています。サーバー側のフックや、それに類する設定は引き継がれません。

よく使われる形態が二つあります。`--depth` を指定してクローンした浅いリポジトリーは、最近のコミットだけを保持します。用語集は、それより古い親コミットが「焼き切られている」と表現しており、後からフェッチして履歴を深くできます。サブモジュールは、別のリポジトリーの中に置かれるリポジトリーで、独自の履歴を持ち、外側のリポジトリーから特定の一つのコミットに固定されます。

概念を判断に生かす

「リポジトリーとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 変更を検証し、リリースし、元に戻せる仕組みを整え、金曜の深夜ではなく火曜の午後にお客様へ届けられるようにします。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 GitHub ActionsでGIS SaaSのデプロイを自動化 — デプロイ、データ、報告を手作業から解放

GitHub ActionsでGIS SaaSのデプロイを自動化 お問い合わせ →

なぜ重要なのか

リポジトリーはプロジェクトの記憶です。あるコード行がおかしく見えるとき、履歴を調べれば、それがいつ、どのコミットで、どのような説明とともに入ったか分かります。ただし、それが役立つかは、書かれたコミットの質次第です。そのため、履歴を大切にするチームは、コミットメッセージに規則を定め、善意に任せずフックで強制します。

また、ほとんどのツールが扱う単位でもあります。アクセス権はリポジトリー単位で付与され、フックはリポジトリー単位で実行され、継続的インテグレーションも通常は一つのリポジトリーから始まります。リポジトリーの境界を決めることは、一回のレビュー、一回の検査、一つの権限が及ぶ範囲を決めることでもあります。

リポジトリーは、渡されたものを保持します。ファイルを削除するコミットは新しい状態を追加するだけで、それ以前の状態は、すでに保持しているすべてのクローンに、そのまま残ります。その永続性こそバージョン管理の目的であり、両

面があります。古いバージョンはいつでも復元できますが、誤ってコミットしたパスワードも同じです。

さらに、リポジトリーに保持できるものは、アプリケーションのコードに限りません。インフラストラクチャーの定義、文書、設定も、同じように履歴、レビューの段階、既知の状態を再構築する手段を得られます。構成がリポジトリーに収められたサーバーは、一度も見たことのない人でも再構築できます。

実際の活用例

一人が18か月かけて構築した四つのリポジトリーでは、コミットに共通の基準を使っています。変更の種類を示す慣例的な接頭辞、スコープ、所定の文字数未満の件名を使い、絵文字も帰属を記すトレーラーも付けません。作業に使う各リポジトリーのコミットメッセージ用フックが、違反を拒否します。基準を守るかどうかが、コミットする人ではなく、リポジトリーの性質になるわけです。その履歴から、何に取り組んだかも分かります。記録された2026年7月の実装段階で、最大のリポジトリーの履歴は、機能追加156件、修正133件、文書128件、保守作業81件、リファクタリング26件、スタイル20件、性能関連の変更5件、テスト3件でした。ただし、フックが検査するのは形式までです。内容の品質は検査しません。

自社のインフラストラクチャー、デプロイ済みの公開サービス、その他の構成要素の設定については、人の頭の中ではなく、リポジトリーに記述することが要件でした。記録された実装には12,065行のYAMLがあり、19のプレイブックと34のロールに分かれていました。この数値は、その段階を示すもので、現在のリポジトリーの規模ではありません。バックアップとメールの自動化も設定されていますが、運用上の有効化は別の段階です。構成一式はリポジトリーから再現できます。ホストは層別のグループに属し、グループの変数が、そのホストで実行するロールを宣言します。収束済みのホストは変更ゼロを報告します。変更には、サーバー上で直接編集するより時間がかかり、後には収束処理の前に事前検査のプレイも追加されました。

リポジトリーは、注意が他へ向いていた間に入ったものも保持します。別の会社の作業が、リポジトリーの再編を経ても残っていました。そこには有効な認証情報を平文で含む作業用ログもあり、リポジトリーの履歴の大半にわたり、どの検査にも見つからずツリーに残っていました。どれも実行されず、まさにそのために残りました。対策は、リポジトリーの指示の冒頭に、この会社のインフラストラクチャーだけを扱うというスコープの規則を置き、削除せず隔離することでした。履歴を読める状態に保ちつつ、秘密情報スキャナー、ボールトの検査、絵文字の検査は、隔離したファイルも引き続き読みます。

問題が起きるところ

- **一度コミットした秘密情報は、削除後も残ります。** 次のコミットでファイルを削除しても、前のコミットと、それ以降に作られたすべてのクローンに残ります。認証情報は公開されたものとして扱い、まず更新してから、コピーを持つ全員に混乱を起こしてまで履歴を書き換える価値があるか判断します。
- **チェックアウト中のブランチにプッシュします。** ベア形式ではないリポジトリーで、現在のブランチを更新するプッシュを受けると、作業ツリーと履歴が食い違います。そのためGitは、既定では `receive.denyCurrentBranch` によって拒否します。プッシュ先にするリポジトリーは、ベア形式にします。
- **浅いクローンで、履歴の一部から履歴について答えます。** `--depth` で作ったクローンは最近のコミットだけを保持するため、 `git log` 、 `git blame` 、最後のタグからバージョンを求めるスクリプトも、その範囲しか見ません。過去が必要なビルドやリリースのジョブでは、全履歴をフェッチします。
- **ローカルのフックは、次のクローンには届きません。** クライアント側のフックは `.git/hooks` にあり、Gitの解説書も、クローンではコピーされないと明記しています。そこでだけ強制する方針は、誰かが新たにクローンした日に効かなくなります。フックのスクリプトをリポジトリーで追跡し、 `core.hooksPath` でそこを指定するか、サーバー側の `pre-receive` で規則を強制します。
- **大きなバイナリーは、すべてのクローンをいつまでも膨らませます。** クローンは全ファイルの全バージョンを保持するため、大きなバイナリーを差し替えるたびに、古いコピーが履歴に残り、新しいクローンは毎回それらをすべてダウンロードします。
- **どこからも実行されないファイルは、読まれなくなります。** 無効なタスク、古いインベントリー、作業用ログもすべてのクローンに残りますが、テスト、ビルド、デプロイのどれも触れないため、レビューから漏れます。リポジトリーの内容は、そのプロジェクトが責任を負うものに合わせ、それ以外は削除するか、検査からは見える場所に分けて置きます。

参考資料

- gitglossary：リポジトリー — Git自身による定義です。ベアリポジトリー、作業ツリー、浅いリポジトリー、サブモジュールの項目も隣にあります。
- Gitリポジトリーの取得 — ディレクトリーから新たに始める方法とクローンする方法の二つ、それにクローンが何を取得するかを説明します。
- Gitのオブジェクト — オブジェクトデータベースが、内容のハッシュをキーにして保存する仕組みです。
- gitrepository-layout — リポジトリー内の各ファイルやディレクトリーの役割を説明します。

- サーバーでのGitの利用 — 既存のリポジトリーをベア形式にして、共有する方法です。
- Gitフック — クライアント側とサーバー側のフック、およびクローンがクライアント側のフックを引き継がない理由を説明します。

用語集

実際の制約に合わせて選ぶ

デプロイの流れ、運用上のリスク、緊急対応の担当者をお知らせください。検査、復旧手段、明確な責任の所在を伴う変更の範囲を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「リポジトリーとは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/repository/