Engineer

Репозиторій

сховище, що зберігає версії набору файлів

Що це таке

Репозиторій, у розмові найчастіше просто «репо», англійською repository, — це місце, де система контролю версій тримає повний запис проєкту: кожен зафіксований стан кожного файлу, хто вніс кожну зміну і навіщо, а також імена, що ведуть усередину цієї історії. Файли в редакторі — лише один його вигляд; репозиторій — це все, що стоїть за ним.

Git дає цьому слову досить точне визначення у власному глосарії: набір посилань (refs) разом із базою об'єктів, що містить усі об'єкти, досяжні з цих посилань. Обидві половини варто розгорнути. База об'єктів тримає вміст. Кожну версію файлу, кожен перелік каталогу і кожен коміт записано як об'єкт, названий за хешем його вмісту; книга Git описує це як файлову систему з адресацією за вмістом, де ключ обчислюється із самого значення. Два однакові файли — це один об'єкт, а будь-яка зміна, хоч яка мала, дає нове ім'я. Посилання — людський бік: гілки й теги, короткі імена, кожне з яких вказує на один коміт і дає людям вхід.

Більшість репозиторіїв, з якими працюють люди, мають прикріплене робоче дерево. Файли, які відкривають у редакторі, лежать у каталозі проєкту, а сам репозиторій живе поруч у прихованому каталозі `.git`. Голий (bare) репозиторій робочого дерева не має взагалі. Назва його каталогу за домовленістю закінчується на `.git`, він містить тільки історію, і саме в такій формі його тримає сервер, бо файли там ніхто не редагує; туди надсилають зміни і звідти їх забирають. Глосарій Git каже те саме: ті, хто публікує відкриті репозиторії, зазвичай роблять доступними саме голі. Книга Git показує, як створити такий з наявного проєкту: `git clone --bare my_project my_project.git`.

Клонування — місце, де задум Git видно найкраще. Клон отримує майже всі дані, які має сервер: типово разом із ним надходить кожна версія кожного файлу за всю історію проєкту. Тож кожен клон — повноцінний репозиторій, здатний заступити сервер, якщо той буде втрачено. Винятки невеликі, але важливі, і книга Git їх називає: хуки на боці сервера та подібні налаштування лишаються позаду.

Два різновиди трапляються часто. Неглибокий (shallow) репозиторій, створений клонуванням із `--depth`, тримає лише недавні коміти; глосарій каже, що старших батьків у нього «cauterized away», тобто відсічено, а історію згодом можна поглибити через fetch. Підмодуль — це репозиторій усередині іншого, зі своєю історією, який зовнішній репозиторій прив’язує до одного конкретного коміту.

Застосуйте поняття до рішення

Застосуйте «Що таке репозиторій?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Перевірено, випущено і з можливістю відкату, тож зміни доходять до клієнтів у вівторок удень, а не пізно в п'ятницю.

Переглянути наші послуги Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. Автоматизувала розгортання GIS SaaS через CI/CD — розгортання, дані та звітність — усе без ручної роботи

Автоматизувала розгортання GIS SaaS через CI/CD Зв'язатися з нами →

Чому це важливо

Репозиторій — пам’ять проєкту. Коли рядок коду виглядає неправильно, історія розповідає, коли він з’явився, у якому коміті та з яким поясненням. Користь від цього не більша за коміти, які пишуть люди, тому команди, яким не байдужа власна історія, встановлюють правила для повідомлень комітів і застосовують їх хуками, а не доброю волею.

Це також одиниця, з якою працює більшість інструментів. Доступ надають на рівні репозиторію, хуки виконуються в репозиторії, а безперервна інтеграція зазвичай стартує з одного репозиторію. Провести межу репозиторію означає провести межу того, що охоплює один перегляд, одна перевірка й один дозвіл.

Репозиторій тримає все, що йому дали. Коміт, який видаляє файл, додає новий стан; усі попередні стани лишаються на місці, у кожному клоні, що вже їх має. Ця тривкість — сама суть контролю версій, і вона ріже в обидва

боки: стару версію завжди можна повернути, як і пароль, який хтось помилково закомітив.

Зрештою, репозиторій може містити значно більше, ніж код застосунку. Опис інфраструктури, документація та конфігурація отримують із нього те саме: історію, крок перегляду і спосіб відновити відомий стан. Сервер, чиє налаштування лежить у репозиторії, може відбудувати людина, яка ніколи його не бачила.

Як це виглядає на практиці

Чотири репозиторії, збудовані за вісімнадцять місяців однією людиною, мають один стандарт комітів: конвенційний префікс, що називає різновид зміни, область, тему, коротшу за встановлену межу, без емодзі й без рядків приписування. Хук на повідомлення коміту відхиляє все, що порушує цей стандарт, у кожному репозиторії, що несе роботу, тож стандарт є властивістю репозиторію, а не того, хто комітить. Тоді історія показує, якою була робота. На зафіксованому етапі реалізації в липні 2026 року історія найбільшого репозиторію містила 156 комітів із новими функціями, 133 із виправленнями, 128 із документацією, 81 зі службовими змінами, 26 із рефакторингом, 20 зі стильовими змінами, 5 з оптимізацією продуктивності та 3 з тестами. Сторінка називає й межу: формат перевіряють, а вміст — ні.

Для власної інфраструктури компанії, розгорнутих публічних сервісів і конфігурації інших компонентів завданням було мати опис у репозиторії, а не в голові. Зафіксована реалізація містила 12 065 рядків YAML, розподілених між 19 плейбуками й 34 ролями; ці числа описують той етап, а не поточний репозиторій. Там налаштовано автоматизацію резервного копіювання й пошти, але її введення в дію є окремим кроком. Інфраструктуру можна відтворити з репозиторію: хост належить до групи рівня, змінні якої оголошують його ролі, а хост, уже приведений до заданого стану, звітує про нуль змін. Кожна зміна займає більше часу, ніж редагування безпосередньо на сервері, а перед приведенням до заданого стану згодом додано попередній перевірочний сценарій.

Репозиторій тримає й те, що потрапило в нього, поки увага була деінде. Робота, зроблена для іншої компанії, проїхала разом із перебудовою репозиторію, а з нею чорновий журнал із чинними обліковими даними відкритим текстом, який лишався в дереві непоміченим усіма перевірками більшу частину історії репозиторію. Нічого з цього не виконувалося, і саме

тому воно вижило. Відповіддю стало правило меж на початку набору інструкцій репозиторію, інфраструктура однієї компанії і тільки її, а також карантин замість видалення, щоб історія лишалася читною, тоді як сканер секретів, перевірка сховища секретів та перевірка емодзі й далі читають файли в карантині.

Де виникають проблеми

- **Секрет, закомічений один раз, переживає своє видалення.** Якщо файл прибрати в наступному коміті, він лишається в попередньому та в кожному клоні, зробленому відтоді. Облікові дані варто вважати публічними: спершу замінити їх, а тоді вирішити, чи варте переписування історії незручностей для всіх, хто має копію.
- **Надсилання (push) влучає в поточну гілку.** Push до репозиторію з робочим деревом, поточна гілка якого саме оновлюється, розсинхронізував би робоче дерево з історією, тому Git типово відхиляє такий push через `receive.denyCurrentBranch`. Репозиторій, до якого надсилають зміни, має бути голим.
- **Неглибокий клон відповідає на питання про історію за її частиною.** Клон, зроблений із `--depth`, має лише недавні коміти, тож `git log`, `git blame` і будь-який скрипт, що виводить версію з останнього тегу, бачать тільки цей зріз. Завданням збирання й випуску, яким потрібне минуле, слід забирати повну історію.
- **Локальний хук зупиняється на наступному клоні.** Хуки на боці клієнта лежать у `.git/hooks`, і книга Git прямо каже, що клон лишає їх позаду. Політика, яку застосовують лише там, закінчується того дня, коли хтось клонує наново. Скрипти хуків варто тримати під контролем версій у репозиторії й спрямувати на них `core.hooksPath` або застосовувати правило на боці сервера через `pre-receive`.
- **Великі двійкові файли роздувають кожен клон назавжди.** Оскільки клон несе кожну версію кожного файлу, кожна замінена копія великого двійкового файлу лишається в історії, і кожен новий клон завантажує їх усі.
- **Файли, які ніщо не запускає, оминають перегляд.** Вимкнене завдання, застарілий інвентар чи чорновий журнал лежать у кожному клоні, проте тести, збирання й розгортання до них не торкаються. Вміст репозиторію має обмежуватися тим, за що проєкт несе відповідальність, а все інше слід прибрати або відкласти туди, де захисти й далі його бачать.

Що почитати далі

- gitglossary: repository — власне визначення Git поруч із bare repository, worktree, shallow repository і submodule.

- Getting a Git Repository — два способи отримати репозиторій, створити в каталозі з нуля або клонувати, і що приносить клон.
- Git Objects — як база об’єктів записує вміст під хешем самого вмісту.
- gitrepository-layout — для чого кожен файл і каталог усередині репозиторію.
- Getting Git on a Server — як перетворити наявний репозиторій на голий і відкрити до нього доступ.
- Git Hooks — хуки на боці клієнта й сервера, і чому клон лишає клієнтські позаду.

Словник

Обирайте за фактичними умовами

Опишіть шлях розгортання, експлуатаційні ризики та чергову команду. Ми можемо визначити зміну з перевірками, шляхом відновлення та чіткою відповідальністю.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке репозиторій?» стосується вашої організації. Конкретного прикладу з поточної системи достатньо для початку розмови.

Переглянути наші послуги

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

Дбайливо зроблено з любов'ю — D.

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

Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.

Надруковано з https://engineer.company/uk/glossary/repository/