Engineer

Repo

ett arkiv som behåller sparade versioner av en samling filer

Vad det är

Ett repo, kort för engelskans repository och den form som nästan alla använder i tal, är där ett versionshanteringssystem håller den fullständiga historiken för ett projekt: varje sparat tillstånd hos varje fil, vem som gjorde varje ändring och varför, och de namn som leder in i den historiken. Filerna du redigerar är en vy av det. Repot är allt som finns bakom den vyn.

Git definierar ordet i sin egen ordlista med viss precision: en samling referenser, refs, tillsammans med en objektdatabas som innehåller alla objekt som kan nås från dem. Båda halvorna är värda att packa upp. Objektdatabasen rymmer innehållet. Varje version av en fil, varje kataloglistning och varje commit lagras som ett objekt och namnges efter en hash av sitt innehåll, vilket Git-boken beskriver som ett innehållsadresserat filsystem: ett nyckel-värde-lager där nyckeln räknas fram ur värdet. Två identiska filer är ett och samma objekt, och varje ändring, hur liten den än är, ger ett nytt namn. Referenserna är den mänskliga sidan: grenar och taggar, korta namn som var och en pekar på en commit och ger människor en väg in.

De flesta repon som människor arbetar i har ett arbetsträd kopplat till sig. Filerna du öppnar i en editor ligger i projektkatalogen, och själva repot bor bredvid dem i en dold katalog, `.git`. Ett bart repo har inget arbetsträd alls. Dess katalognamn slutar enligt konvention på `.git`, det rymmer bara historiken, och det är den form som en server håller, eftersom ingen redigerar filer där. Människor pushar till det och hämtar från det. Gits ordlista säger detsamma: den som publicerar ett offentligt repo gör det oftast tillgängligt som ett bart repo. Git-boken visar hur man skapar ett ur ett befintligt projekt med `git clone --bare my_project my_project.git`.

Kloning är där Gits design syns. En klon tar emot nästan all data som servern har: som standard följer varje version av varje fil under projektets hela historik med. Varje klon är därför ett fullständigt repo, som kan ersätta servern om servern går förlorad. Undantagen är få men viktiga, och Git-boken namnger dem: hookar på serversidan och liknande inställningar stannar kvar.

Två varianter dyker ofta upp. Ett grunt repo, som skapas genom kloning med `--depth`, behåller bara de senaste commits. Ordlistan säger att de äldre föräldrarna har ”cauterized away”, bränts bort, och historiken kan fördjupas senare med en fetch. En submodul är ett repo som hålls inuti ett annat, med sin egen historik, fastlåst av det yttre repot vid en exakt commit.

Använd begreppet i ett beslut

Använd ”Vad är ett repo?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Testade, släppta och möjliga att backa, så att ändringar når kunderna en tisdagseftermiddag i stället för sent en fredagskväll.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Automatiserade GIS-driftsättning i GitHub Actions — driftsättning, data och rapportering, helt utan manuellt arbete

Automatiserade GIS-driftsättning i GitHub Actions Hör av er →

Varför det spelar roll

Ett repo är ett projekts minne. När en kodrad ser fel ut säger dess historik när den kom in, i vilken commit och med vilken förklaring. Det är bara så bra som de commits som människor skriver, vilket är skälet till att team som bryr sig om sin historik sätter regler för commitmeddelanden och upprätthåller dem med hookar i stället för med god vilja.

Repot är också den enhet som de flesta verktyg arbetar på. Åtkomst beviljas per repo, hookar körs per repo, och en CI-körning utgår oftast från ett. Den som drar gränsen för ett repo har därmed dragit gränsen för vad en granskning, en kontroll och en behörighet täcker.

Repon behåller det de får. En commit som raderar en fil lägger till ett nytt tillstånd och lämnar varje tidigare tillstånd där det var, i varje klon som redan har det. Den beständigheten är hela poängen med versionshantering, och den skär åt båda hållen: en gammal version går alltid att återställa, och det gör även ett lösenord som någon committade av misstag.

Slutligen kan ett repo rymma mycket mer än applikationskod. Infrastrukturdefinitioner, dokumentation och konfiguration vinner alla samma saker på det: en historik, ett granskningssteg och ett sätt att bygga upp ett känt tillstånd igen. En server vars uppsättning finns i ett repo kan byggas upp igen av någon som aldrig har sett den.

Så ser det ut i praktiken

Här bär repot mer än kod och historik: en regel för hur commits skrivs, en hel IT-miljö i YAML och en gräns för vad som får ligga i trädet.

Fyra repon som byggdes under arton månader av en enda person delar en standard för commits: ett konventionellt prefix som anger typen av ändring, ett scope, en ämnesrad under en fast längd, inga emojier och inga tillskrivningsrader. En hook för commitmeddelanden i varje repo där arbete sker avvisar allt som bryter mot standarden, vilket gör den till en egenskap hos repot snarare än hos den som committar. Historiken visar sedan vad arbetet var. I det dokumenterade genomförandeskedet i juli 2026 visade historiken i det största repot 156 commits för funktioner, 133 rättningar, 128 för dokumentation, 81 för underhållssysslor, 26 refaktoreringar, 20 för stil, 5 för prestanda och 3 för test. Sidan anger också gränsen: formatet kontrolleras, men innehållet gör det inte.

För företagets egen infrastruktur, publika tjänster i drift och konfiguration av andra komponenter var uppgiften att få den beskriven i ett repo i stället för i någons huvud. Den dokumenterade implementeringen innehöll 12 065 rader YAML, fördelade på 19 playbooks och 34 roller; siffrorna beskriver det skedet, inte det nuvarande repot. Automatisering av säkerhetskopiering och e-post är konfigurerad där, medan aktivering i drift är ett separat steg. IT-miljön går att återskapa från repot: en värddator ingår i en nivågrupp vars variabler deklarerar vilka roller den kör, och en konvergerad värddator rapporterar noll ändringar. Varje ändring tar längre tid än att redigera en fil direkt på servern, och en preflight-play lades senare till framför konvergeringen.

Det tredje projektet visar vad ett repo behåller när uppmärksamheten flyttar någon annanstans. Arbete åt ett annat företag följde med genom en omstrukturering av repot, bl.a. en kladdlogg med aktiva inloggningsuppgifter i klartext som låg kvar i trädet förbi varje skydd under större delen av repots historik. Allt detta låg stilla, och det är precis därför det överlevde. Svaret blev en avgränsningsregel överst i repots instruktioner, ett enda företags infrastruktur

och bara den, plus en karantän i stället för en radering, så att historiken förblir läsbar medan hemlighetsskannern, vault-skyddet och emojikontrollen fortfarande läser de karantänsatta filerna.

Var det går fel

- **En hemlighet som har committats en gång överlever att den raderas.** Att ta bort filen i nästa commit lämnar den kvar i den föregående och i varje klon som har gjorts sedan dess. Behandla inloggningsuppgiften som offentlig: byt ut den först, och avgör sedan om det är värt att skriva om historiken, med det besvär det innebär för alla som har en kopia.
- **En push möter en utcheckad gren.** Att pusha till ett repo som inte är bart, där den aktuella grenen är den som uppdateras, skulle lämna arbetsträdet i otakt med historiken, så Git vägrar som standard en sådan push genom `receive.denyCurrentBranch`. Ett repo som människor pushar till bör vara bart.
- **En grund klon svarar utifrån en del av historiken.** En klon som görs med `--depth` bär bara de senaste commits, så `git log`, `git blame` och varje skript som härleder ett versionsnummer från den senaste taggen ser bara den skivan. Bygg- och releasejobb som behöver det förflutna bör hämta hela historiken.
- **En lokal hook slutar gälla vid nästa klon.** Hookar på klientsidan ligger i `.git/hooks`, och Git-boken säger uttryckligen att en klon inte kopierar dem. En policy som bara upprätthålls där upphör den dag någon klonar på nytt. Håll hookskripten versionshanterade i repot och peka `core.hooksPath` mot dem, eller upprätthåll regeln på serversidan med `pre-receive`.
- **Stora binärfiler sväller varje klon för alltid.** Eftersom en klon bär varje version av varje fil blir varje ersatt kopia av en stor binärfil kvar i historiken, och varje ny klon laddar ner dem allihop.
- **Filer som inget kör förblir olästa.** En avaktiverad task, en inaktuell inventory eller en kladdlogg ligger fortfarande i varje klon, men inget test, inget bygge och ingen driftsättning rör den, så den undgår granskning. Repots innehåll bör motsvara det som projektet ansvarar för, och allt annat bör tas bort eller ställas åt sidan där skydden fortfarande ser det.

Läs mer

- gitglossary: repository — Gits egen definition, bredvid bare repository, worktree, shallow repository och submodule.
- Getting a Git Repository — de två sätten att få ett repo, att börja om i en katalog eller att klona, och vad en klon för med sig.
- Git Objects — hur objektdatabasen lagrar innehåll under en hash av just det innehållet.

- gitrepository-layout — vad varje fil och katalog inuti ett repo är till för.
- Getting Git on a Server — att göra om ett befintligt repo till ett bart repo och dela det.
- Git Hooks — hookar på klient- och serversidan, och varför en klon lämnar dem på klientsidan bakom sig.

Ordlista

Välj utifrån era faktiska förutsättningar

Ta med driftsättningsvägen, driftsriskerna och dem som har jour. Vi kan avgränsa en förändring med kontroller, en väg tillbaka och tydligt ansvar.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är ett repo?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

welcome@engineer.company +45 50 10 61 84 Köpenhamn, Danmark

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/repository/