Engineer

Repository

et lager, der gemmer versioner af et sæt filer

Hvad det er

Et repository, i daglig tale som regel forkortet repo og på dansk også kaldt et lager, er det sted, hvor et versionsstyringssystem holder hele optegnelsen over et projekt: hver registreret tilstand af hver fil, hvem der lavede hver ændring og hvorfor, og de navne, der fører ind i historikken. Filerne, man redigerer, er én visning af det; repositoryet er alt det, der ligger bag visningen.

Git definerer ordet ret præcist i sin egen ordliste: en samling refs sammen med en objektdatabase, der indeholder alle de objekter, som kan nås fra de refs. Begge halvdele er værd at folde ud. Objektdatabasen rummer indholdet. Hver version af en fil, hver mappeoversigt og hvert commit lægges ind som et objekt og navngives efter en hash af indholdet, hvilket Git-bogen beskriver som et indholdsadresseret filsystem: et nøgle-værdi-lager, hvor nøglen beregnes ud fra værdien. To ens filer er ét objekt, og enhver ændring, hvor lille den end er, giver et nyt navn. Refs er den menneskelige side: grene og tags, korte navne, der hver peger på ét commit og giver folk en vej ind.

De fleste repositories, folk arbejder i, har et arbejdstræ knyttet til sig. Filerne, man åbner i en editor, ligger i projektmappen, og selve repositoryet bor ved siden af dem i en skjult `.git` -mappe. Et bart repository har slet intet arbejdstræ. Mappens navn ender efter konvention på `.git` , den rummer kun historikken, og det er den form, en server holder, fordi filerne dér aldrig redigeres; folk pusher til den og henter fra den. Gits ordliste siger det samme: dem, der udgiver offentlige repositories, stiller som regel bare repositories til rådighed. Git-bogen viser, hvordan man laver et ud fra et eksisterende projekt med `git clone --bare my_project my_project.git` .

Kloning er stedet, hvor Gits design viser sig. En klon modtager næsten alle de data, serveren har: som standard kommer hver version af hver fil i hele projektets historik med. Hver klon er derfor et komplet repository, der kan træde i serverens sted, hvis serveren går tabt. Undtagelserne er små, men de betyder noget, og Git-bogen nævner dem: hooks på serversiden og lignende indstillinger bliver tilbage.

To varianter dukker ofte op. Et shallow repository, lavet ved at klone med `--depth`, beholder kun de nyeste commits; ordlisten siger, at de ældre forældre er »cauterized away«, skåret bort, og historikken kan senere gøres dybere med en fetch. Et submodule er et repository inde i et andet, med sin egen historik, som det ydre repository låser fast til ét bestemt commit.

Brug begrebet i en beslutning

Brug »Hvad er et repository?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Testet, udgivet og til at rulle tilbage, så ændringer når kunderne en tirsdag eftermiddag i stedet for sent en fredag aften.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har automatiseret GIS-SaaS-udrulning med CI/CD — udrulning, data og rapportering, alt sammen uden manuelt arbejde

Har automatiseret GIS-SaaS-udrulning med CI/CD Kontakt os →

Hvorfor det betyder noget

Et repository er projektets hukommelse. Når en linje kode ser forkert ud, fortæller historikken, hvornår den kom, i hvilket commit og med hvilken forklaring. Det er kun så godt som de commits, folk skriver, og derfor sætter teams, der holder af deres historik, regler for commit-beskeder og håndhæver dem med hooks frem for god vilje.

Det er også den enhed, det meste værktøj arbejder på. Adgang tildeles pr. repository, hooks kører pr. repository, og kontinuerlig integration starter som regel fra ét. Når man trækker grænsen for et repository, har man trukket grænsen for, hvad én gennemgang, ét tjek og én tilladelse dækker.

Repositories beholder det, de får. Et commit, der sletter en fil, tilføjer en ny tilstand; alle tidligere tilstande bliver liggende, i hver klon, der allerede har dem. Den varighed er hele pointen med versionsstyring, og den skærer begge veje: en gammel version kan altid hentes frem igen, og det kan en adgangskode, som nogen har committet ved en fejl, også.

Endelig kan et repository rumme langt mere end applikationskode. Infrastrukturdefinitioner, dokumentation og konfiguration får det samme ud af det: en historik, et gennemgangstrin og en måde at genopbygge en kendt tilstand på. En server, hvis opsætning ligger i et repository, kan genopbygges af en, der aldrig har set den.

Sådan ser det ud i praksis

Fire repositories bygget over atten måneder af én person deler én commit-standard: et konventionelt præfiks, der navngiver ændringens art, et virkefelt, en emnelinje under en fast længde, ingen emojis og ingen tilskrivningslinjer. En hook på commit-beskeden afviser alt, der bryder den, i hvert repository, der bærer arbejde, og det gør standarden til en egenskab ved repositoryet frem for ved den, der committer. Historikken viser så, hvad arbejdet var. I den dokumenterede implementeringsfase i juli 2026 viste historikken i det største repository 156 funktions-commits, 133 rettelser, 128 dokumentation, 81 husholdning, 26 omskrivninger, 20 stil, 5 ydeevne og 3 test. Siden nævner også grænsen: formatet tjekkes, og indholdet gør ikke.

For virksomhedens egen infrastruktur, offentligt tilgængelige tjenester i drift og konfiguration af andre komponenter var opgaven at få det beskrevet i et repository frem for i et hoved. Den dokumenterede implementering rummede 12.065 linjer YAML, fordelt på 19 playbooks og 34 roller; tallene beskriver den fase, ikke det nuværende repository. Automatisering af sikkerhedskopiering og mail er konfigureret dér, mens aktivering i drift er et særskilt trin. Miljøet kan reproduceres fra repositoryet: en vært hører til en tier-gruppe, hvis variabler erklærer de roller, den kører, og en konvergeret vært rapporterer nul ændringer. Hver ændring tager længere tid end at rette en fil direkte på serveren, og et preflight-play blev senere sat foran konvergeringen.

Et repository husker også det, der kom med, mens opmærksomheden var et andet sted. Arbejde for en anden virksomhed kørte med gennem en omstrukturering af repositoryet, blandt andet en kladdelog med levende legitimationsoplysninger i klartekst, som lå i træet forbi hver eneste vagtpost gennem det meste af repositoryets historie. Intet af det kørte, og det er præcis derfor, det overlevede. Svaret blev en omfangsregel øverst i repositoryets instruktionssæt, én virksomheds infrastruktur og kun den, plus en karantæne i stedet for en sletning, så historikken forbliver læselig, mens hemmelighedsskanneren, boksvagten og emoji-tjekket stadig læser de filer, der ligger i karantæne.

Hvor det går galt

- **En hemmelighed, der er committet én gang, overlever sin sletning.** Når filen fjernes i næste commit, ligger den stadig i det forrige og i hver klon, der er lavet siden. Behandl legitimationsoplysningen som offentlig: skift den først, og afgør derefter, om det er forstyrrelsen værd at omskrive historikken for alle, der har en kopi.
- **Et push rammer en udtjekket gren.** Et push til et repository med arbejdstræ, hvis aktuelle gren er den, der opdateres, ville bringe arbejdstræet ud af trit med historikken, så Git afviser som standard et sådant push via `receive.denyCurrentBranch`. Et repository, folk pusher til, bør være bart.
- **En shallow klon besvarer spørgsmål om historikken ud fra en del af den.** En klon lavet med `--depth` har kun de nyeste commits, så `git log`, `git blame` og ethvert script, der udleder en version fra det seneste tag, ser kun det udsnit. Bygge- og udgivelsesjob, der har brug for fortiden, bør hente hele historikken.
- **En lokal hook standser ved næste klon.** Hooks på klientsiden ligger i `.git/hooks`, og Git-bogen siger klart, at en klon lader dem blive tilbage. En politik, der kun håndhæves dér, ophører den dag, nogen kloner på ny. Hold hook-scripts under versionsstyring i repositoryet og peg `core.hooksPath` på dem, eller håndhæv reglen på serversiden med `pre-receive`.
- **Store binære filer gør hver klon større for altid.** Fordi en klon bærer hver version af hver fil, bliver hver udskiftet kopi af en stor binær fil liggende i historikken, og hver ny klon henter dem alle.
- **Filer, som intet kører, slipper uden om gennemgang.** En deaktiveret opgave, en forældet inventarliste eller en kladdelog ligger stadig i hver klon, mens test, bygning og udrulning går uden om den. Repositoryets indhold bør høre til det, projektet har ansvar for, og alt andet bør fjernes eller sættes til side, hvor vagtposterne stadig kan se det.

Læs mere

- gitglossary: repository — Gits egen definition, side om side med bare repository, worktree, shallow repository og submodule.
- Getting a Git Repository — de to måder at få et på, at starte forfra i en mappe eller at klone, og hvad en klon henter ned.
- Git Objects — hvordan objektdatabasen lægger indhold ind under en hash af selve indholdet.
- gitrepository-layout — hvad hver fil og mappe inde i et repository bruges til.
- Getting Git on a Server — sådan gøres et eksisterende repository til et bart og deles.

- Git Hooks — hooks på klient- og serversiden, og hvorfor en klon lader dem på klientsiden blive tilbage.

Ordliste

Vælg ud fra jeres faktiske vilkår

Medbring udrulningsforløbet, driftsrisiciene og dem, der har vagten. Vi kan afgrænse en ændring med kontroller, en vej tilbage og tydeligt ansvar.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er et repository?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

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

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

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

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/repository/