Engineer

git hook

⚓ et script, git kører, når noget sker

Hvad det er

En git hook er et program, som git selv starter på et fast punkt i sit eget arbejde: før et commit bliver registreret, når en commitbesked er skrevet, før et push forlader maskinen, eller når en server modtager et. Hver hook er en fil opkaldt efter sit øjeblik, for eksempel `pre-commit`, `commit-msg` eller `pre-push`, uden filendelse. Som standard leder git efter dem i `.git/hooks` inde i repositoryet, og indstillingen `core.hooksPath` peger det et andet sted hen. Filen skal være eksekverbar, for git ignorerer en fil uden eksekveringsbit, men den kan skrives i et hvilket som helst sprog, maskinen kan køre.

Klientside-hooks kører i en udviklers egen klon og udløses af handlinger som commit og merge. Serverside-hooks kører på den maskine, der modtager push, for eksempel en forge, og udløses, når pushede commits ankommer.

Det, der gør en hook til en vagt og ikke bare en besked, er dens exitstatus. `pre-commit`-hooken tager ingen argumenter, og en exitstatus forskellig fra nul afbryder commitet. `commit-msg`-hooken får stien til en fil med den foreslåede besked og kan afvise beskeden eller redigere den. `pre-push`-hooken får fjernrepositoryets navn og placering som argumenter og én linje pr. reference på standardinput, og en exitstatus forskellig fra nul standser pushet, før noget bliver sendt. På serveren kører `pre-receive` én gang pr. push og læser det gamle objektnavn, det nye objektnavn og referencenavnet for hver reference, og afslutter den med en anden status end nul, bliver ingen af referencerne opdateret. Hooks, der kører efter hændelsen, som `post-commit`, er beregnet til notifikationer og kan ikke ændre udfaldet.

Når `git init` opretter et repository, fylder det hooks-mappen med eksempelscripts, hvis navne ender på `.sample`. git kører først et af dem, når det er omdøbt uden endelsen.

Brug begrebet i en beslutning

Brug »Hvad er en git hook?« 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

En hook gør et tjek, som nogen skal huske, til et, der sker af sig selv, og på det billigste tidspunkt. En formatter eller en scanner efter hemmeligheder, der kører, mens commitet bliver lavet, fanger et problem, mens ændringen stadig er frisk i forfatterens hoved, i stedet for i en gennemgang dage senere eller i en CI-kørsel, efter at arbejdet er delt med andre.

Hooks er også måden, en regel bliver en egenskab ved et repository i stedet for ved den, der tilfældigvis committer. En konvention i en bidragsvejledning holder kun, så længe folk læser vejledningen; den samme konvention i en `commit-msg`-hook gælder for hvert eneste commit.

De to grupper kan love forskelligt meget. Pro Git-bogen gør opmærksom på, at klientside-hooks ikke bliver kopieret, når et repository klones, og anbefaler at håndhæve regler på serversiden. En serverside-hook som `pre-receive` gælder for hvert push fra alle, uanset hvad de har installeret lokalt. Prisen er tid: en hook kører ved hvert commit eller hvert push, og et tjek, der gør det langsomt at committe, frister folk til at springe det over.

Sådan ser det ud i praksis

Dette websteds eget repository installerer sine hooks gennem pre-commit-frameworket, konfigureret i `.pre-commit-config.yaml` og installeret af `task setup`, der kører `pre-commit install --install-hooks`. Filen installerer tre git hooks. I pre-commit-fasen kører den Prettier, markdownlint, stylelint, yamllint, codespell, Ruffs lint- og formattjek, en gitleaks-scanning efter hemmeligheder, et emojitjek og et tonetjek. I commit-msg-fasen kører den ét script, der holder beskederne til en konventionel form uden emojis. I pre-push-fasen kører den hele kvalitetsporten, `task check`, som tager minutter og ikke sekunder, en pris, der ifølge kommentaren ved hooken er værd at betale, fordi et push er sjældent.

Filens indledning siger, at hooks aldrig må omgås med `--no-verify`, og kommentaren holder fast i det: arbejde, der ikke kan vente på porten, er ikke færdigt.

En commit-msg-hook holder også fire repositories på én standard for historikken: et konventionelt præfiks og et scope, en emnelinje under en fast længde, ingen emojis og ingen attributionslinjer. Hooken afviser alt, der afviger fra standarden, i hvert repository, der bærer arbejde, så standarden hører til repositoryet og ikke til den, der committer.

En serverside-hook til virksomhedens egen forge blev prøvekørt, før noget rigtigt push nåede den. Sådan en hook kører i et bart repository uden arbejdstræ, i et miljø, som forgen sætter op, og læser de pushede referencer fra standardinput, så det beviser ikke meget at køre den i hånden på en bærbar. Prøvekørslen opretter et bart repository, installerer hooken, kloner, committer, pusher og kontrollerer både den vej, hvor pushet accepteres, og den vej, hvor det afvises. Den fandt to fejl, som var usynlige i filen, og som er beskrevet nedenfor.

I et projekt viste commit-hooken og kvalitetsporten sig at køre forskellige tjek, selv om et dokument sagde, at hooken kørte kvalitetsporten. En test fører nu begge tilbage til de tjek, de faktisk kalder, og fejler, når de to mængder skilles ad. De fem, der kun havde kørt i kvalitetsporten, var dem, der læste CV-teksten, så koden blev beskyttet ved hvert commit og teksten kun, når nogen huskede det.

En klientside-hook behøver slet ikke et framework. Denne her, gemt som `.git/hooks/commit-msg` og gjort eksekverbar, afviser et commit, hvis emnelinje ikke begynder med en konventionel type:

```
#!/bin/sh
# git sender stien til den foreslåede commitbesked som $1
pattern='^(feat|fix|docs|chore)(\([a-z0-9-]+\))?: .+'
if head -n 1 "$1" | grep -Eq "$pattern"; then
    exit 0
fi
echo "commit-msg: begynd emnelinjen med en type, som i 'fix(nav): ...'" >&2
exit 1
```

Hvor det går galt

- **En ny klon kommer uden hooks.** Hver klon skal installere dem, og det er den opgave, `task setup` løser her. En kommentar i dette websteds `Taskfile.yml` beskriver den stille udgave af fejlen: uden `pre-commit` på maskinen installerer `task setup` ingenting, hver hook bliver i stilhed ikke kørt, og resultatet ser præcis ud som et repository, der består.
- `--no-verify` **går lige forbi dem.** `git commit --no-verify` omgår både `pre-commit` - og `commit-msg` -hooken, `git push --no-verify` omgår `pre-push` , og sætter man `core.hooksPath` til `/dev/null` , er alle hooks slået fra på én gang. En regel, der skal holde, hører også hjemme i en serverside-hook.
- **gits miljø følger hooken ind i andre repositories.** git eksporterer variabler som `GIT_DIR` , så git-kommandoer i en hook finder det rigtige repository, og githooks(5) råder til at rydde dem, når en hook arbejder i et andet. Prøvekørslen på forgen ramte netop dette: kvalitetsporten tjekkede den pushede revision ud og kørte testene i den, testene arvede gits henvisning til det bare repository, og nitten af dem blev røde på kode, der var korrekt.
- **Standardinput bærer grænsetilfældene med sig.** Prøvekørslens anden fejl var en fallback, der, når den stod uden en reference at arbejde med, indsatte en pladsholder, som git ikke kan slå op. Fordi værdien kommer på standardinput, viste det sig som en tilsyneladende urelateret fejl langt længere nede. Siden nævner også, at forgen vil se tvungne opdateringer og sletninger af grene, som prøvekørslen ikke konstruerer. I githooks(5) når en slettet reference frem til `pre-push` som `(delete)` med et objektnavn af lutter nuller.
- **En langsom hook bliver sprunget over.** At få de fem teksttjek ind i commit-hooken krævede, at to af dem blev gjort hurtige nok til et pre-commit-budget, og den side siger ligeud, at køretiden vokser med indholdet, og at der er et punkt, hvor en langsom hook bliver omgået.
- **Formen består, og meningen bliver ikke tjekket.** Som siden om de fire repositories udtrykker det, bliver formatet tjekket og indholdet ikke, så en emnelinje, der følger reglerne, men intet beskriver, slipper lige så godt igennem som en, der forklarer ændringen.

Læs mere

- githooks documentation — alle de hooks, git understøtter, hvornår de kører, hvad de modtager, og hvad deres exitstatus afgør.
- Pro Git: Git Hooks — klientside- og serverside-hooks, hvordan man installerer en, og hvorfor en klon ikke får dem med.
- Pro Git: An Example Git-Enforced Policy — en `update` -hook på serveren, der håndhæver et beskedformat og adgangsregler, med klientside-hooks, der advarer før et push.

- git config — indstillingen `core.hooksPath`, til at flytte hooks ud af `.git/hooks` eller slå dem alle fra.
- git commit — indstillingen `--no-verify` og de to hooks, den omgår.
- git push — `pre-push`-hooken og indstillingen `--no-verify`, der springer den over.

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 en git hook?« 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/git-hook/