Engineer

git-hook

⚓ et skript git kjører når noe skjer

Hva det er

En git-hook er et program som git selv starter på et fast punkt i sitt eget arbeid: før en commit blir registrert, etter at en commitmelding er skrevet, før en push forlater maskinen, eller når en server tar imot en push. Hver hook er en fil oppkalt etter øyeblikket sitt, for eksempel `pre-commit`, `commit-msg` eller `pre-push`, og navnet har ingen filendelse. Som standard leter git etter dem i `.git/hooks` inne i repoet, og innstillingen `core.hooksPath` sender git et annet sted. Filen må være kjørbar, fordi git overser en fil som mangler kjørebiten, men selve programmet kan skrives i hvilket som helst språk maskinen kan kjøre.

Hooks på klientsiden kjører i utviklerens egen klone og slår til ved operasjoner som commit og merge. Hooks på serversiden kjører på maskinen som tar imot push, for eksempel en forge, og slår til når pushede commits kommer fram.

Det som gjør en hook til en vaktpost og ikke bare et varsel, er avslutningskoden. `pre-commit`-hooken tar ingen argumenter, og en avslutningskode forskjellig fra null avbryter commiten. `commit-msg`-hooken får stien til en fil med den foreslåtte meldingen og kan både avvise meldingen og endre den. `pre-push`-hooken får navnet og adressen til fjernrepoet som argumenter og én linje per referanse på standard input, og en avslutningskode forskjellig fra null stopper pushen før noe er sendt. På serveren kjører `pre-receive` én gang per push og leser gammelt objektnavn, nytt objektnavn og referansenavn for hver referanse. Avslutter den med noe annet enn null, blir ingen av referansene oppdatert. Hooks som kjører etter hendelsen, som `post-commit`, er ment for varsling og kan ikke endre utfallet.

Når `git init` oppretter et repo, fyller kommandoen hook-katalogen med eksempelskript der navnene slutter på `.sample`. git kjører et slikt skript først når det har fått nytt navn uten endelsen.

Bruk begrepet i en beslutning

Bruk «Hva er en git-hook?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Testet, utgitt og mulig å reversere, slik at endringer når kundene en tirsdag ettermiddag i stedet for sent en fredagskveld.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Automatiserte GIS-SaaS-utrulling i GitHub Actions — produksjonssetting, data og rapportering, alt uten manuelle grep

Automatiserte GIS-SaaS-utrulling i GitHub Actions Ta kontakt →

Hvorfor det er viktig

En hook gjør en kontroll som noen må huske på, til en kontroll som skjer av seg selv, i det billigste øyeblikket. En formaterer eller en skanner etter hemmeligheter som kjører idet commiten lages, fanger et problem mens endringen fortsatt er i hodet på den som skrev den, og ikke i en gjennomgang dager senere eller i en CI-kjøring etter at arbeidet er delt med andre.

Hooks er også måten en regel blir en egenskap ved et repo på, i stedet for ved den som tilfeldigvis committer. En konvensjon i en bidragsveiledning holder bare så lenge folk leser veiledningen. Den samme konvensjonen i en `commit-msg` -hook gjelder for hver eneste commit.

De to gruppene skiller seg i hvor mye de kan love. Boken Pro Git påpeker at hooks på klientsiden ikke blir kopiert når et repo klones, og anbefaler å håndheve retningslinjer på serversiden. En hook på serversiden, som `pre-receive` , gjelder for hver push fra alle, uansett hva de har installert lokalt. Prisen er tid: en hook kjører ved hver commit eller hver push, og en kontroll som gjør det tregt å committe, frister folk til å hoppe over den.

Slik ser det ut i praksis

Repoet til dette nettstedet installerer hookene sine gjennom rammeverket pre-commit, konfigurert i `.pre-commit-config.yaml` og installert av `task setup` , som kjører `pre-commit install --install-hooks` . Filen installerer tre git-hooks. I pre-commit-fasen kjører den Prettier, markdownlint, stylelint, yamllint, codespell, Ruffs lint- og formatkontroller, en gitleaks-skanning etter hemmeligheter, en emojikontroll og en tonekontroll. I commit-msg-fasen kjører den ett skript som holder meldingene til en konvensjonell form uten emojier. I pre-push-fasen kjører den hele kvalitetsporten, `task check` , som tar minutter og ikke sekunder, en pris

kommentaren ved hooken mener er verdt å betale fordi en push skjer sjelden. Toppen av filen sier at hookene aldri skal omgås med `--no-verify`, og kommentaren holder fast ved det: en push som ikke kan vente, er ikke klar.

En hook for commitmeldinger holder også fire repoer til én historikkstandard: et konvensjonelt prefiks og omfang, en emnelinje under en fast lengde, ingen emojier og ingen attribusjonslinjer. Hooken avviser alt som bryter med standarden, i hvert repo der det gjøres arbeid, så standarden tilhører repoet og ikke den som committer.

En hook på serversiden for selskapets egen forge ble prøvekjørt før noen ekte push nådde den. En slik hook kjører i et nakent repo uten arbeidstre, i et miljø som forgen setter opp, og leser de pushede referansene fra standard input, så å kjøre den for hånd på en bærbar maskin beviser lite. Prøvekjøringen oppretter et nakent repo, installerer hooken, kloner, committer, pusher og kontrollerer både veien der pushen godtas og veien der den avvises. Den fant to feil som var usynlige i filen, og begge er beskrevet nedenfor.

I ett prosjekt viste det seg at commit-hooken og kvalitetsporten kjørte ulike kontroller, selv om et dokument sa at hooken kjørte porten. En test kartlegger nå hvilke kontroller de to faktisk kaller, og feiler når settene spriker. De fem som bare hadde kjørt i porten, var de som leser CV-teksten, så koden ble voktet ved hver commit og teksten bare når noen husket det.

En hook på klientsiden klarer seg fint uten rammeverk. Denne, lagret som `.git/hooks/commit-msg` og gjort kjørbar, avviser en commit der emnelinjen ikke begynner med en konvensjonell type:

```
#!/bin/sh
# git passes the path of the proposed commit message as $1
pattern='^(feat|fix|docs|chore)(\([a-z0-9-]+\))?: .+'
if head -n 1 "$1" | grep -Eq "$pattern"; then
    exit 0
fi
echo "commit-msg: start the subject with a type, as in 'fix(nav): ...'" >&2
exit 1
```

Hvor det går galt

- **En fersk klone kommer uten hookene.** Hver klone må installere dem selv, og det er den jobben `task setup` gjør her. En kommentar i nettstedets `Taskfile.yml` beskriver den stille varianten av feilen: står `pre-commit` ikke på maskinen, installerer `task setup` ingenting, hver eneste hook lar i det stille være å kjøre, og resultatet ser nøyaktig ut som et repo der alt består.
- `--no-verify` **går rett forbi dem.** `git commit --no-verify` hopper over hookene `pre-commit` og `commit-msg`, `git push --no-verify` hopper over `pre-push`, og settes `core.hooksPath` til `/dev/null`, slås alle hooks av på én gang. En regel som må holde, hører derfor hjemme i en hook på serversiden også.
- **gits miljø følger hooken inn i andre repoer.** git eksporterer variabler som `GIT_DIR` slik at git-kommandoer i en hook finner riktig repo, og githooks(5) anbefaler å nullstille dem når en hook arbeider i et annet repo. Prøvekjøringen mot forgen traff akkurat dette: kvalitetsporten sjekket ut den pushede revisjonen og kjørte testene inne i den, testene arvet gits peker til det nakne repoet, og nitten av dem ble røde på korrekt kode.
- **Standard input bærer grensetilfellene.** Den andre feilen prøvekjøringen fant, var en reserveløsning som satte inn en plassholder git ikke kan slå opp, når den manglet en referanse å arbeide med. Fordi verdien kommer inn på standard input, dukket svikten opp som en feil uten synlig sammenheng, mye lenger ned. Siden påpeker også at forgen kommer til å få tvungne oppdateringer og slettede grener, som prøvekjøringen ikke lager. I githooks(5) kommer en slettet referanse fram til `pre-push` som `(delete)` med et objektnavn av bare nuller.
- **En treg hook blir hoppet over.** Å ta de fem prosakontrollene inn i commit-hooken betydde å gjøre to av dem raske nok for tidsbudsjettet før en commit, og den siden sier rett ut at kjøretiden vokser med innholdet, og at det finnes et punkt der en treg hook blir omgått.
- **Formatet består, og innholdet slipper gjennom usjekket.** Som siden om de fire repoene formulerer det, sjekkes formatet, men innholdet gjør det ikke. En emnelinje som følger standarden, men som ikke beskriver noe, slipper derfor like lett gjennom som en som forklarer endringen.

Les mer

- githooks-dokumentasjonen — hver hook git støtter, når den kjører, hva den får inn, og hva avslutningskoden avgjør.
- Pro Git: Git Hooks — hooks på klient- og serversiden, hvordan du installerer en, og hvorfor en klone lar dem bli igjen.
- Pro Git: An Example Git-Enforced Policy — en `update`-hook på serversiden som håndhever et meldingsformat og tilgangsregler, med hooks på klientsiden som advarer før en push.

- git config — innstillingen `core.hooksPath`, for å flytte hooks ut av `.git/hooks` eller slå av alle sammen.
- git commit — valget `--no-verify` og de to hookene det hopper over.
- git push — `pre-push`-hooken og valget `--no-verify` som hopper over den.

Ordliste

Velg ut fra deres faktiske rammer

Ta med utrullingsløpet, driftsrisikoene og dem som har vakt. Vi kan avgrense en endring med kontroller, en vei tilbake og tydelig ansvar.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er en git-hook?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

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

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/git-hook/