Engineer

git-hook

⚓ ett skript som git kör när något händer

Vad det är

En git-hook är ett program som git själv startar vid en bestämd punkt i sitt eget arbete: innan en commit registreras, när ett commitmeddelande har skrivits, innan en push lämnar maskinen eller när en server tar emot en. Varje hook är en fil som heter efter sitt ögonblick, t.ex. `pre-commit`, `commit-msg` eller `pre-push`, utan filändelse. Som standard letar git efter dem i `.git/hooks` inne i repot, och inställningen `core.hooksPath` pekar git någon annanstans. Filen måste vara körbar, eftersom git bortser från en fil utan körbarhetsbiten, men den kan skrivas i vilket språk som helst som maskinen kör.

Hookar på klientsidan körs i en utvecklares egen klon och utlöses av operationer som commit och merge. Hookar på serversidan körs på maskinen som tar emot pushar, t.ex. en forge, och utlöses när pushade commits kommer fram.

Det som gör en hook till en vakt i stället för en notis är dess slutstatus. Hooken `pre-commit` tar inga argument, och en slutstatus skild från noll avbryter commiten. Hooken `commit-msg` får sökvägen till en fil med det föreslagna meddelandet och kan avvisa meddelandet eller redigera det. Hooken `pre-push` får namnet på och adressen till det mottagande repot som argument och en rad per referens på standard in, och en slutstatus skild från noll stoppar pushen innan något har skickats. På servern körs `pre-receive` en gång per push och läser det gamla objektnamnet, det nya objektnamnet och referensnamnet för varje referens, och avslutas den med en status skild från noll uppdateras ingen av referenserna. Hookar som körs efter händelsen, t.ex. `post-commit`, är till för notifiering och kan inte ändra utfallet.

När `git init` skapar ett repo fyller det hookkatalogen med exempelskript vars namn slutar på `.sample`. Git kör ett sådant skript först när det har döpts om utan ändelsen.

Använd begreppet i ett beslut

Använd ”Vad är en git-hook?” 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

En hook gör en kontroll som någon måste komma ihåg till en som sker av sig själv, i det billigaste ögonblicket. En formaterare eller en sökning efter hemligheter som körs när commiten görs fångar ett problem medan ändringen fortfarande är färsk i författarens huvud, snarare än vid en granskning flera dagar senare eller i en CI-körning efter att arbetet redan har delats.

Hookar är också sättet som en regel blir en egenskap hos repot snarare än hos den som råkar committa. En konvention i en guide för bidragsgivare håller bara så länge folk läser guiden, medan samma konvention i en `commit-msg` -hook gäller för varje commit.

De två grupperna skiljer sig åt i hur mycket de kan lova. Boken Pro Git påpekar att hookar på klientsidan inte kopieras när ett repo klonas, och rekommenderar att policy upprätthålls på serversidan. En hook på serversidan som `pre-receive` gäller för varje push från alla, oavsett vad de har installerat lokalt. Kostnaden är tid: en hook körs vid varje commit eller varje push, och en kontroll som gör det långsamt att committa lockar folk att hoppa över den.

Så ser det ut i praktiken

Den här webbplatsens eget repo installerar sina hookar via ramverket pre-commit, konfigurerat i `.pre-commit-config.yaml` och installerat av `task setup` , som kör `pre-commit install --install-hooks` . Filen installerar tre git-hookar. I steget pre-commit kör den Prettier, markdownlint, stylelint, yamllint, codespell, Ruffs kontroller av lint och formatering, en sökning efter hemligheter med gitleaks, en kontroll av emojier och en kontroll av tonen. I steget commit-msg kör den ett enda skript som håller meddelandena till en konventionell form utan emojier. I steget pre-push kör den hela kvalitetsgrinden, `task check` , som tar minuter snarare än sekunder, ett pris som kommentaren vid hooken anser vara värt att

betala eftersom en push är sällsynt. Filens inledning säger att hookarna aldrig ska kringgås med `--no-verify`, och kommentaren håller fast vid det: en push som inte kan vänta är inte klar.

En hook för commitmeddelanden håller också fyra repon till en och samma standard för historiken: ett konventionellt prefix och scope, en ämnesrad under en fast längd, inga emojier och inga tillskrivningsrader. Hooken avvisar allt som inte följer standarden i varje repo där arbete sker, så att standarden hör till repot snarare än till den som committar.

En hook på serversidan för företagets egen forge provkördes innan någon riktig push nådde den. En sådan hook körs i ett bare-repo utan arbetskatalog, i en miljö som forgen sätter upp, och läser de pushade referenserna från standard in, så att köra den för hand på en laptop bevisar lite. Provkörningen skapar ett bare-repo, installerar hooken, klonar, committar, pushar och verifierar både fallet där pushen godtas och fallet där den avvisas. Den hittade två fel som inte syntes i filen, och båda beskrivs nedan.

I ett projekt visade det sig att commit-hooken och kvalitetsgrinden körde olika kontroller, trots att ett dokument sa att hooken körde grinden. Ett test kartlägger nu vilka kontroller de båda faktiskt anropar och fallerar när mängderna skiljer sig åt. De fem som bara hade körts i grinden var de som läser CV-prosan, så koden skyddades vid varje commit och texten bara när någon kom ihåg det.

En hook på klientsidan behöver inget ramverk alls. Den här, sparad som `.git/hooks/commit-msg` och gjord körbar, avvisar en commit vars ämnesrad inte börjar med en konventionell typ:

```
#!/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
```

Var det går fel

- **En ny klon kommer utan hookarna.** Varje klon måste installera dem, och det är jobbet som `task setup` gör här. En kommentar i webbplatsens `Taskfile.yml` beskriver den tysta varianten av felet: utan `pre-commit` på maskinen installerar `task setup` ingenting, varje hook uteblir utan ett ljud, och resultatet ser precis ut som ett repo som klarar sina kontroller.
- `--no-verify` **går rakt förbi dem.** `git commit --no-verify` kringgår hookarna `pre-commit` och `commit-msg`, `git push --no-verify` kringgår `pre-push`, och att sätta `core.hooksPath` till `/dev/null` stänger av alla hookar på en gång. En regel som måste hålla hör därför också hemma i en hook på serversidan.
- **Gits miljö följer med hooken in i andra repon.** Git exporterar variabler som `GIT_DIR` så att git-kommandon i en hook hittar rätt repo, och githooks(5) råder till att rensa dem när en hook arbetar i ett annat repo. Provkörningen mot forgen stötte på exakt detta: grinden checkade ut den pushade revisionen och körde testerna i den, testerna ärvde gits pekare till bare-repot, och nitton av dem blev röda på korrekt kod.
- **Standard in bär kantfallen.** Provkörningens andra fel var en reservlösning som satte in en platshållare som git inte kan slå upp när den saknade en referens att arbeta med. Eftersom värdet kommer in på standard in visade sig felet som ett orelaterat fel mycket längre ned. Sidan noterar också att forgen kommer att se tvingade uppdateringar och borttagna grenar, som provkörningen inte konstruerar. Enligt githooks(5) når en borttagen referens `pre-push` som `(delete)` med ett objektnamn av bara nollor.
- **En långsam hook hoppas över.** Att föra in de fem prosakontrollerna i commit-hooken krävde att två av dem gjordes snabba nog för en tidsbudget före commit, och den sidan säger rakt ut att körtiden växer med innehållet och att det finns en punkt där en långsam hook kringgås.
- **Formatet godkänns medan betydelsen lämnas okontrollerad.** Som sidan om de fyra repona uttrycker det kontrolleras formatet men inte innehållet, så en ämnesrad som följer standarden men inte beskriver någonting passerar precis lika väl som en som förklarar ändringen.

Läs mer

- githooks documentation — varje hook som git stöder, när den körs, vad den får och vad dess slutstatus avgör.
- Pro Git: Git Hooks — hookar på klient- och serversidan, hur man installerar en och varför en klon lämnar dem kvar.
- Pro Git: An Example Git-Enforced Policy — en `update`-hook på serversidan som upprätthåller ett meddelandeformat och regler för åtkomst, med hookar på klientsidan som varnar före en push.

- git config — inställningen `core.hooksPath`, för att flytta hookarna ut ur `.git/hooks` eller stänga av alla på en gång.
- git commit — flaggan `--no-verify` och de två hookar den kringgår.
- git push — hooken `pre-push` och flaggan `--no-verify` som hoppar över den.

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 en git-hook?” 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/git-hook/