Engineer

Git-Hook

⚓ Skript, das Git bei einem bestimmten Ereignis ausführt

Was es ist

Ein Git-Hook ist ein Programm, das Git selbst an einem festgelegten Punkt seiner Arbeit startet: bevor ein Commit aufgezeichnet wird, nachdem eine Commit-Nachricht geschrieben wurde, bevor ein Push den Rechner verlässt oder wenn ein Server einen empfängt. Jeder Hook ist eine Datei ohne Endung, die nach diesem Zeitpunkt benannt ist, etwa `pre-commit`, `commit-msg` oder `pre-push`. Standardmäßig sucht Git im Verzeichnis `.git/hooks` des Repositorys danach; mit `core.hooksPath` lässt sich ein anderer Ort angeben. Die Datei muss ausführbar sein, denn Git ignoriert sie ohne gesetztes Ausführungsbit. Sie kann aber in jeder Sprache geschrieben sein, die der Rechner ausführen kann.

Clientseitige Hooks laufen im eigenen Klon eines Entwicklers und werden bei Vorgängen wie Commits und Merges ausgelöst. Serverseitige Hooks laufen auf dem Rechner, der Pushes empfängt, etwa einer Forge, und werden ausgelöst, wenn gepushte Commits eintreffen.

Sein Rückgabestatus macht aus einem Hook zur Benachrichtigung einen Hook zum Durchsetzen von Regeln. Der Hook `pre-commit` erhält keine Argumente. Ein Rückgabestatus ungleich null bricht den Commit ab. `commit-msg` erhält den Pfad einer Datei mit der vorgeschlagenen Nachricht und kann sie ablehnen oder bearbeiten. `pre-push` erhält den Namen und die Adresse des Remotes als Argumente sowie über die Standardeingabe eine Zeile je Referenz. Ein Status ungleich null stoppt den Push, bevor etwas gesendet wird. Auf dem Server läuft `pre-receive` einmal je Push. Er liest für jede Referenz den alten Objektnamen, den neuen Objektnamen und den Referenznamen. Endet er mit einem Status ungleich null, wird keine der Referenzen aktualisiert. Nach dem Ereignis laufende Hooks wie `post-commit` dienen der Benachrichtigung und können das Ergebnis nicht mehr ändern.

Wenn `git init` ein Repository anlegt, füllt es das Hook-Verzeichnis mit Beispielskripten, deren Namen auf `.sample` enden. Git führt ein solches Skript erst aus, nachdem diese Endung beim Umbenennen entfernt wurde.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist ein Git-Hook?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Geprüft, veröffentlicht und rückgängig zu machen: Änderungen erreichen Ihre Kunden am Dienstagnachmittag statt spät am Freitagabend.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. GIS-SaaS-Deployments mit GitHub automatisiert — Bereitstellung, Daten und Berichte ohne Handarbeit

GIS-SaaS-Deployments mit GitHub automatisiert Kontakt aufnehmen →

Warum es wichtig ist

Ein Hook macht aus einer Prüfung, an die jemand denken muss, eine Prüfung, die von selbst und zum günstigsten Zeitpunkt stattfindet. Ein Formatierer oder ein Secret-Scanner, der beim Erstellen des Commits läuft, findet ein Problem, solange der Autor die Änderung noch im Kopf hat. Es fällt dann nicht erst Tage später im Review oder nach dem Teilen der Arbeit in einem CI-Lauf auf.

Hooks machen eine Regel außerdem zu einer Eigenschaft des Repositorys und lösen sie von der Person, die gerade committet. Eine Konvention in einem Leitfaden für Mitwirkende gilt nur, solange die Leute ihn lesen. Dieselbe Konvention in einem Hook vom Typ `commit-msg` gilt für jeden Commit.

Die beiden Gruppen unterscheiden sich darin, wie viel sie versprechen können. Das Buch Pro Git weist darauf hin, dass beim Klonen eines Repositorys dessen clientseitige Hooks nicht kopiert werden, und empfiehlt, Regeln auf der Serverseite durchzusetzen. Ein serverseitiger Hook wie `pre-receive` gilt für jeden Push von jedem Mitwirkenden, unabhängig von dessen lokaler Installation. Der Preis ist Zeit: Ein Hook läuft bei jedem Commit oder jedem Push. Eine Prüfung, die Commits verlangsamt, lädt zum Umgehen ein.

Wie es in der Praxis aussieht

Das Repository dieser Website installiert seine Hooks über das Framework pre-commit. Die Konfiguration steht in `.pre-commit-config.yaml`; die Hooks installiert `task setup`, das `pre-commit install --install-hooks` ausführt. Die Datei installiert drei Git-Hooks. In der pre-commit-Phase laufen Prettier, markdownlint, stylelint, yamllint, codespell, Ruffs Lint- und Formatprüfungen, ein gitleaks-Scan nach Geheimnissen, eine Emoji-Prüfung und eine Prüfung des sprachlichen Tons. In der commit-msg-Phase erzwingt ein Skript eine konventionelle Form der Nachrichten ohne Emojis. In der pre-push-Phase läuft die gesamte verbindliche Qualitätsprüfung, `task check`. Sie dauert Minuten statt Sekunden. Der Kommentar zu diesem Hook akzeptiert den Preis, weil ein Push selten ist. Der Dateikopf verbietet das Umgehen der Hooks mit `--no-verify`, und der Kommentar bleibt dabei: Ein Push, der nicht warten kann, ist nicht bereit.

Ein Hook für Commit-Nachrichten setzt außerdem für vier Repositorys einen gemeinsamen Standard für die Historie durch: ein konventionelles Präfix mit Geltungsbereich, eine Betreffzeile unter einer festgelegten Länge, keine Emojis und keine Attribution-Trailer. Der Hook lehnt alles ab, was davon abweicht, und zwar in jedem Repository, das Arbeit enthält. Der Standard gehört damit zum Repository und nicht zur jeweils committenden Person.

Ein serverseitiger Hook für die unternehmenseigene Forge wurde erprobt, bevor ihn ein echter Push erreichte. Ein solcher Hook läuft in einem Bare-Repository ohne Arbeitsbaum, in einer von der Forge gesetzten Umgebung, und liest gepushte Referenzen über die Standardeingabe. Ihn von Hand auf einem Laptop auszuführen, beweist deshalb wenig. Die Erprobung legt ein Bare-Repository an, installiert den Hook, klont, committet und pusht und prüft sowohl den Annahme- als auch den Ablehnungspfad. Dabei wurden zwei Fehler gefunden, die beim Lesen der Datei nicht erkennbar waren. Beide werden unten beschrieben.

In einem Projekt stellte sich heraus, dass Commit-Hook und Qualitätsprüfung unterschiedliche Prüfungen ausführten, obwohl ein Dokument behauptete, der Hook führe die Qualitätsprüfung aus. Ein Test ermittelt nun für beide die tatsächlich aufgerufenen Prüfungen und schlägt fehl, wenn die Mengen auseinanderfallen. Die fünf Prüfungen, die nur in der Qualitätsprüfung gelaufen waren, lasen die Lebenslauftexte. Der Code war somit bei jedem Commit geschützt, die Texte nur, wenn jemand daran dachte.

Ein clientseitiger Hook braucht überhaupt kein Framework. Dieser hier, gespeichert als `.git/hooks/commit-msg` und ausführbar gemacht, lehnt einen Commit ab, dessen Betreff nicht mit einem konventionellen Typ beginnt:

```
#!/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
```

Wo es schiefgeht

- **Ein frischer Klon kommt ohne Hooks an.** Jeder Klon muss sie installieren. Hier erledigt das `task setup`. Ein Kommentar in der `Taskfile.yml` dieser Website beschreibt die unauffällige Form des Fehlers: Ohne `pre-commit` auf dem Rechner installiert `task setup` nichts, sämtliche Hooks bleiben stillschweigend aus, und das Ergebnis sieht genauso aus wie ein Repository, das alle Prüfungen besteht.
- `--no-verify` **geht geradewegs an ihnen vorbei.** `git commit --no-verify` umgeht `pre-commit` und `commit-msg`, `git push --no-verify` umgeht `pre-push`, und `core.hooksPath` auf `/dev/null` zu setzen, schaltet alle Hooks zugleich ab. Eine Regel, die gelten muss, gehört auch in einen serverseitigen Hook.
- **Gits Umgebung folgt dem Hook in andere Repositorys.** Git exportiert Variablen wie `GIT_DIR`, damit Git-Befehle in einem Hook das richtige Repository finden. githooks(5) empfiehlt, sie zu entfernen, wenn ein Hook in einem anderen arbeitet. Genau das trat bei der Forge-Erprobung auf: Die Qualitätsprüfung checkte die gepushte Revision aus und führte darin die Tests aus. Diese erbten Gits Verweis auf das Bare-Repository, und neunzehn von ihnen schlugen bei korrektem Code fehl.
- **Die Standardeingabe transportiert die Randfälle.** Der andere Fehler der Erprobung lag in einer Ausweichlösung, die mangels einer verwendbaren Referenz einen von Git nicht auflösbaren Platzhalter einsetzte. Weil der Wert über die Standardeingabe ankommt, erschien der Fehler erst viel später als scheinbar unzusammenhängende Fehlermeldung. Die Seite hält außerdem fest, dass die Forge erzwungene Aktualisierungen und gelöschte Branches sehen wird, die die Erprobung nicht nachstellt. Laut githooks(5) erreicht eine gelöschte Referenz `pre-push` als `(delete)` mit einem ausschließlich aus Nullen bestehenden Objektnamen.

- **Ein langsamer Hook wird umgangen.** Die fünf Textprüfungen in den Commit-Hook aufzunehmen, erforderte, zwei davon für das Zeitbudget eines pre-commit-Hooks schnell genug zu machen. Die Seite sagt deutlich, dass die Laufzeit mit dem Inhalt wächst und irgendwann der Punkt erreicht ist, an dem ein langsamer Hook umgangen wird.
- **Das Format passt, die Bedeutung bleibt ungeprüft.** Wie die Seite über die vier Repositorys sagt: Das Format wird geprüft, der Inhalt nicht. Eine formal passende Betreffzeile, die nichts beschreibt, besteht daher genauso wie eine, die die Änderung erklärt.

Weiterführende Lektüre

- githooks-Dokumentation — sämtliche von Git unterstützten Hooks, ihre Ausführungszeitpunkte, ihre Eingaben und die Entscheidungen, die ihr Rückgabestatus trifft.
- Pro Git: Git-Hooks — clientseitige und serverseitige Hooks, ihre Installation und warum sie beim Klonen zurückbleiben.
- Pro Git: Beispiel einer durch Git erzwungenen Richtlinie — ein serverseitiger Hook vom Typ `update`, der ein Nachrichtenformat und Zugriffsregeln durchsetzt, mit clientseitigen Hooks, die vor dem Push warnen.
- git config — die Einstellung `core.hooksPath`, um Hooks aus `.git/hooks` zu verlagern oder alle abzuschalten.
- git commit — die Option `--no-verify` und die zwei Hooks, die sie umgeht.
- git push — der Hook `pre-push` und die Option `--no-verify`, die ihn überspringt.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Bereitstellungsweg, die Betriebsrisiken und die Bereitschaftsverantwortlichen. Wir können eine Änderung mit Prüfungen, einem Wiederherstellungsweg und klaren Zuständigkeiten abgrenzen.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist ein Git-Hook?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

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

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/git-hook/