Engineer

хук

⚓ скрипт, який git запускає на подію

Що це таке

Хук — це програма, яку git сам запускає у фіксованій точці власної роботи: перед тим як коміт буде записано, коли повідомлення коміту вже написано, перед тим як пуш залишить машину, або коли сервер його отримує. Кожен хук — це файл, названий за своєю миттю, наприклад `pre-commit`, `commit-msg` чи `pre-push`, без розширення. Усталено git шукає їх у `.git/hooks` усередині репозиторію, а параметр `core.hooksPath` вказує йому інше місце. Файл має бути виконуваним, бо файл без біта виконання git ігнорує, але написати його можна будь-якою мовою, яку машина вміє запускати.

Клієнтські хуки працюють у власному клоні розробника і спрацьовують на такі операції, як коміт і злиття. Серверні хуки працюють на машині, що приймає пуші, наприклад на форджі, і спрацьовують, коли надіслані коміти надходять.

Вартовим, а не просто сповіщенням, хук робить його код завершення. Хук `pre-commit` не отримує аргументів, і ненульовий код завершення перериває коміт. Хук `commit-msg` отримує шлях до файлу із запропонованим повідомленням і може відхилити повідомлення або змінити його. Хук `pre-push` отримує як аргументи назву й адресу віддаленого репозиторію, а на стандартний вхід — по одному рядку на кожне посилання, і ненульовий код завершення зупиняє пуш, перш ніж щось буде надіслано. На сервері `pre-receive` запускається один раз на пуш і читає для кожного посилання стару назву об’єкта, нову назву об’єкта та назву посилання, а якщо він завершується з ненульовим кодом, жодне з посилань не оновлюється. Хуки, що працюють після події, як-от `post-commit`, призначені для сповіщень і не можуть змінити її наслідок.

Коли `git init` створює репозиторій, він заповнює каталог хуків прикладами скриптів, назви яких закінчуються на `.sample` . git запускає такий скрипт лише після того, як його перейменують без цього суфікса.

Застосуйте поняття до рішення

Застосуйте «Що таке хук?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Перевірено, випущено і з можливістю відкату, тож зміни доходять до клієнтів у вівторок удень, а не пізно в п'ятницю.

Переглянути наші послуги Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. Автоматизувала розгортання GIS SaaS через CI/CD — розгортання, дані та звітність — усе без ручної роботи

Автоматизувала розгортання GIS SaaS через CI/CD Зв'язатися з нами →

Чому це важливо

Хук перетворює перевірку, про яку хтось має пам’ятати, на таку, що відбувається сама, і в найдешевшу мить. Форматер або сканер секретів, що працює під час створення коміту, ловить проблему, поки зміна ще свіжа в голові автора, а не на рев’ю через кілька днів чи в запуску CI, коли роботою вже поділилися з іншими.

Хуки — це також спосіб зробити правило властивістю репозиторію, а не того, хто випадково робить коміт. Домовленість у посібнику для учасників тримається лише доти, доки люди цей посібник читають; та сама домовленість у хуку `commit-msg` діє для кожного коміту.

Дві групи хуків можуть обіцяти різне. Книжка Pro Git нагадує, що клієнтські хуки не копіюються під час клонування репозиторію, і радить забезпечувати дотримання правил на боці сервера. Серверний хук, як-от `pre-receive` , діє на кожен пуш від усіх, хоч би що вони встановили в себе. Ціна — час: хук працює на кожному коміті чи кожному пуші, а перевірка, через яку коміт стає повільним, спокушає людей її оминути.

Як це виглядає на практиці

Репозиторій цього сайту встановлює свої хуки через фреймворк pre-commit: їх налаштовано у `.pre-commit-config.yaml` , а встановлює їх `task setup` , який виконує `pre-commit install --install-hooks` . Файл встановлює три хуки git. На етапі pre-commit він запускає Prettier, markdownlint, stylelint, yamllint, codespell, перевірки лінтування й форматування Ruff, пошук секретів

gitleaks, перевірку на емодзі та перевірку тону. На етапі commit-msg він запускає один скрипт, що тримає повідомлення в конвенційній формі без емодзі. На етапі pre-push він запускає всю перевірку якості, `task check`, яка триває хвилини, а не секунди; коментар біля хука pre-push вважає, що цю ціну варто платити, бо пуш буває рідко. Вступ файлу велить ніколи не оминати хуки за допомогою `--no-verify`, і коментар цього дотримується: пуш, який не може чекати, ще не готовий.

Хук повідомлень комітів також тримає чотири репозиторії на одному стандарті історії: конвенційний префікс і область, тема не довша за фіксовану довжину, без емодзі й без рядків атрибуції. Хук відхиляє все, що відступає від стандарту, у кожному репозиторії, де ведеться робота, тож стандарт належить репозиторію, а не тому, хто робить коміт.

Серверний хук для власної форджі компанії відрепетирували, перш ніж до нього дійшов справжній пуш. Такий хук виконується в bare-репозиторії без робочого дерева, у середовищі, яке задає форджа, і читає надіслані посилання зі стандартного входу, тож запуск вручну на ноутбуці мало що доводить. Репетиція створює bare-репозиторій, встановлює хук, клонує, робить коміт і пуш, а потім перевіряє і сценарій прийняття, і сценарій відхилення. Вона знайшла два дефекти, яких не було видно у файлі; обидва описано нижче.

В одному проєкті хук коміту і ворота якості, як з'ясувалося, виконували різні перевірки, хоча документ стверджував, що хук виконує всю перевірку якості. Тепер тест зводить обидва переліки до перевірок, які вони насправді викликають, і падає, щойно ці набори розходяться. П'ять перевірок, які працювали лише у воротах якості, були саме тими, що читали текст CV, тож код охоронявся на кожному коміті, а текст — лише тоді, коли хтось про це згадував.

Клієнтському хуку фреймворк узагалі не потрібен. Цей хук, збережений як `.git/hooks/commit-msg` і зроблений виконуваним, відхиляє коміт, тема якого не починається з конвенційного типу:

```
#!/bin/sh
# git передає шлях до запропонованого повідомлення коміту як $1
pattern='^(feat|fix|docs|chore)(\([a-z0-9-]+\))?: .+'
if head -n 1 "$1" | grep -Eq "$pattern"; then
    exit 0
```

```
fi
echo "commit-msg: почніть тему з типу, як у 'fix(nav): ...'" >&2
exit 1
```

Де виникають проблеми

- **Свіжий клон приходить без хуків.** Кожен клон має встановити їх сам, і тут цю роботу виконує `task setup`. Коментар у `Taskfile.yml` цього сайту описує тиху версію збою: без `pre-commit` на машині `task setup` нічого не встановлює, кожен хук мовчки не запускається, а результат виглядає точнісінько як репозиторій, що проходить перевірки.
- `--no-verify` **проходить просто повз них.** `git commit --no-verify` оминає хуки `pre-commit` і `commit-msg`, `git push --no-verify` оминає `pre-push`, а якщо встановити `core.hooksPath` у `/dev/null`, вимикаються всі хуки одразу. Правилу, яке мусить діяти, місце також у серверному хуку.
- **Середовище git іде за хуком в інші репозиторії.** git експортує змінні, як-от `GIT_DIR`, щоб команди git у хуку знаходили потрібний репозиторій, і githooks(5) радить очищати їх, коли хук працює з іншим репозиторієм. Репетиція на форджі натрапила саме на це: ворота якості створили робочу копію надісланої ревізії й запустили в ній тести, тести успадкували вказівник git на bare-репозиторій, і дев'ятнадцять із них стали червоними на правильному коді.
- **Стандартний вхід несе крайні випадки.** Другий дефект репетиції — запасний варіант, який, коли посилання для роботи не було, підставляв заповнювач, що його git не може розв'язати; оскільки значення надходить на стандартний вхід, збій проявлявся значно далі як не пов'язана з ним помилка. Сторінка також зазначає, що форджа бачитиме примусові оновлення й видалення гілок, яких репетиція не відтворює. За githooks(5), видалене посилання надходить до `pre-push` як `(delete)` з назвою об'єкта з самих нулів.
- **Повільний хук оминають.** Щоб перенести п'ять перевірок тексту в хук коміту, дві з них довелося пришвидшити настільки, щоб вони вкладалися в бюджет часу pre-commit, і та сторінка прямо каже, що час роботи зростає разом із вмістом і що з певного моменту повільний хук починають обходити.
- **Формат проходить, а зміст лишається неперевіреним.** Як каже сторінка про чотири репозиторії, формат перевіряється, а зміст — ні, тож тема у правильному форматі, яка нічого не описує, проходить так само успішно, як та, що пояснює зміну.

Що почитати далі

- githooks documentation — усі хуки, які підтримує git, коли кожен із них запускається, що отримує і що вирішує його код завершення.
- Pro Git: Git Hooks — клієнтські й серверні хуки, як установити хук і чому клон їх не переносить.
- Pro Git: An Example Git-Enforced Policy — серверний хук `update`, що забезпечує формат повідомлень і правила доступу, разом із клієнтськими хуками, які попереджають до пушу.
- git config — параметр `core.hooksPath`, щоб перенести хуки з `.git/hooks` або вимкнути їх усі.
- git commit — параметр `--no-verify` і два хуки, які він оминає.
- git push — хук `pre-push` і параметр `--no-verify`, що його пропускає.

Словник

Обирайте за фактичними умовами

Опишіть шлях розгортання, експлуатаційні ризики та чергову команду. Ми можемо визначити зміну з перевірками, шляхом відновлення та чіткою відповідальністю.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке хук?» стосується вашої організації. Конкретного прикладу з поточної системи достатньо для початку розмови.

Переглянути наші послуги

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

Дбайливо зроблено з любов'ю — D.

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

Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.

Надруковано з https://engineer.company/uk/glossary/git-hook/