Engineer

Hook de Git

⚓ script que Git ejecuta cuando ocurre un suceso

Qué es

Un hook de Git es un programa que el propio Git inicia en un punto fijo de su trabajo: antes de registrar un commit, una vez escrito su mensaje, antes de que un push salga de la máquina o cuando un servidor lo recibe. Cada hook es un archivo sin extensión cuyo nombre corresponde a ese momento, como `pre-commit`, `commit-msg` o `pre-push`. De forma predeterminada, Git los busca en `.git/hooks`, dentro del repositorio, y el ajuste `core.hooksPath` permite indicar otro lugar. El archivo debe ser ejecutable, ya que Git lo ignora si no tiene el bit de ejecución, pero puede estar escrito en cualquier lenguaje que ejecute la máquina.

Los hooks del cliente se ejecutan en el clon de cada desarrollador y se activan en operaciones como crear commits y fusionar cambios. Los del servidor se ejecutan en la máquina que recibe los pushes, como una forja, y se activan cuando llegan los commits enviados.

Lo que convierte un hook de notificación en una barrera es su estado de salida. El hook `pre-commit` no recibe argumentos y una salida distinta de cero aborta el commit. `commit-msg` recibe la ruta de un archivo con el mensaje propuesto y puede rechazarlo o editarlo. `pre-push` recibe como argumentos el nombre y la dirección del remoto y, por la entrada estándar, una línea por referencia; una salida distinta de cero detiene el push antes de enviar nada. En el servidor, `pre-receive` se ejecuta una vez por push, lee el nombre antiguo del objeto, el nuevo y el nombre de la referencia para cada referencia y, si termina con estado distinto de cero, no se actualiza ninguna. Los hooks posteriores al suceso, como `post-commit`, sirven para notificar y no pueden cambiar su resultado.

Cuando `git init` crea un repositorio, llena el directorio de hooks con scripts de ejemplo cuyos nombres terminan en `.sample` . Git solo ejecuta uno después de renombrarlo sin ese sufijo.

Utilice el concepto en una decisión

Aplique «¿Qué es un hook de Git?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Cambios probados, publicados y reversibles, para que lleguen a sus clientes un martes por la tarde en vez de un viernes por la noche.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. Despliegue GIS SaaS con GitHub Actions — despliegue, datos e informes automatizados

Despliegue GIS SaaS con GitHub Actions Póngase en contacto →

Por qué importa

Un hook convierte una comprobación que alguien debe recordar en otra que ocurre por sí sola, en el momento de menor coste. Un formateador o un detector de secretos que se ejecuta al crear el commit encuentra un problema mientras quien hizo el cambio aún lo tiene presente, en lugar de días después durante la revisión o en una ejecución de CI una vez compartido el trabajo.

Los hooks también convierten una regla en una propiedad del repositorio, en vez de depender de quien esté creando el commit. Una convención en una guía de contribución solo se mantiene mientras la gente lea la guía; esa misma convención en un hook `commit-msg` se aplica a cada commit.

Los dos grupos difieren en lo que pueden garantizar. El libro Pro Git señala que clonar un repositorio no copia sus hooks del cliente y recomienda imponer las políticas en el servidor. Un hook del servidor como `pre-receive` se aplica a cada push de cualquier persona, con independencia de lo que tenga instalado localmente. El coste es tiempo: un hook se ejecuta en cada commit o cada push, y una comprobación que ralentiza los commits invita a saltársela.

Cómo se aplica en la práctica

El repositorio de este sitio instala sus hooks mediante el framework pre-commit, configurado en `.pre-commit-config.yaml` e instalado por `task setup` , que ejecuta `pre-commit install --install-hooks` . El archivo instala tres hooks de Git. En la fase pre-commit ejecuta Prettier, markdownlint, stylelint, yamllint, codespell, las comprobaciones de lint y formato de Ruff, un análisis de secretos con gitleaks,

una comprobación de emojis y otra del tono. En la fase commit-msg ejecuta un script que exige mensajes con formato convencional y sin emojis. En la fase pre-push ejecuta todo el control de calidad, `task check`, que tarda minutos en vez de segundos: un precio que el comentario del hook acepta porque los pushes son poco frecuentes. La cabecera del archivo dice que nunca se deben omitir los hooks con `--no-verify`, y el comentario mantiene esa postura: un push que no puede esperar no está listo.

Un hook para mensajes de commit también mantiene cuatro repositorios bajo una misma norma de historial: un prefijo y ámbito convencionales, un asunto por debajo de una longitud fija, sin emojis ni líneas finales de atribución. El hook rechaza todo lo que incumpla esa norma en cada repositorio que contiene trabajo, de modo que el estándar pertenece al repositorio y no a quien crea el commit.

Se ensayó un hook del servidor para la forja propia de la empresa antes de que recibiera un push real. Ese hook se ejecuta en un repositorio bare sin árbol de trabajo, con el entorno que establece la forja, y lee las referencias enviadas por la entrada estándar; ejecutarlo a mano en un portátil demuestra poco. El ensayo crea un repositorio bare, instala el hook, clona, crea commits, hace pushes y comprueba tanto la ruta de aceptación como la de rechazo. Encontró dos defectos que no se veían en el archivo, ambos descritos más abajo.

En un proyecto, se comprobó que el hook de commit y el control de calidad ejecutaban comprobaciones distintas, aunque un documento decía que el hook ejecutaba el control. Una prueba determina ahora qué comprobaciones invoca realmente cada uno y falla si los conjuntos difieren. Las cinco que solo se ejecutaban en el control eran las que leían los textos del currículum: el código quedaba protegido en cada commit y la redacción solo cuando alguien se acordaba.

Un hook del cliente no necesita ningún framework. Este, guardado como `.git/hooks/commit-msg` y marcado como ejecutable, rechaza un commit cuyo asunto no empieza por un tipo convencional:

```
#!/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
```

Dónde surgen los problemas

- **Un clon nuevo llega sin hooks.** Cada clon debe instalarlos, que es lo que hace aquí `task setup`. Un comentario en el `Taskfile.yml` de este sitio explica la versión silenciosa del fallo: sin `pre-commit` en la máquina, `task setup` no instala nada, ningún hook se ejecuta y el resultado parece exactamente el de un repositorio que supera las comprobaciones.
- `--no-verify` **pasa de largo.** `git commit --no-verify` omite los hooks `pre-commit` y `commit-msg`, `git push --no-verify` omite `pre-push`, y configurar `core.hooksPath` como `/dev/null` desactiva todos a la vez. Una regla que debe cumplirse también debe estar en un hook del servidor.
- **El entorno de Git sigue al hook a otros repositorios.** Git exporta variables como `GIT_DIR` para que las órdenes de Git en un hook encuentren el repositorio correcto, y githooks(5) aconseja eliminarlas cuando el hook trabaja en otro. El ensayo de la forja encontró exactamente ese caso: el control extraía la revisión enviada y ejecutaba las pruebas dentro de ella; estas heredaban el puntero de Git al repositorio bare y diecinueve fallaban con código correcto.
- **La entrada estándar transporta los casos límite.** El otro defecto del ensayo era una alternativa que, al no tener una referencia con la que trabajar, usaba en su lugar un marcador que Git no podía resolver. Como el valor llega por la entrada estándar, el fallo afloraba mucho más adelante como un error aparentemente ajeno. La página también señala que la forja verá actualizaciones forzadas y eliminaciones de ramas, que el ensayo no genera. En githooks(5), una referencia eliminada llega a `pre-push` como `(delete)` con un nombre de objeto formado solo por ceros.
- **Un hook lento se omite.** Incorporar las cinco comprobaciones de textos al hook de commit exigió acelerar dos para ajustarlas al tiempo asignado a un hook pre-commit. La página deja claro que el tiempo de ejecución crece con el contenido y que hay un punto a partir del cual se evita un hook lento.
- **El formato pasa y el significado queda sin revisar.** Como dice la página de los cuatro repositorios, se comprueba el formato y no el contenido. Un asunto que se ajusta a la norma sin describir nada pasa igual que otro que explica el cambio.

Lecturas adicionales

- Documentación de githooks — Todos los hooks que admite Git, cuándo se ejecutan, qué reciben y qué decide su estado de salida.

- Pro Git: hooks de Git — Hooks del cliente y del servidor, cómo instalar uno y por qué un clon no se los lleva.
- Pro Git: ejemplo de una política impuesta por Git — Un hook `update` del servidor que impone un formato de mensajes y reglas de acceso, junto con hooks del cliente que avisan antes de un push.
- git config — El ajuste `core.hooksPath`, que permite sacar los hooks de `.git/hooks` o desactivarlos todos.
- git commit — La opción `--no-verify` y los dos hooks que omite.
- git push — El hook `pre-push` y la opción `--no-verify` que lo omite.

Glosario

Elija según sus restricciones reales

Describa la ruta de despliegue, los riesgos operativos y quiénes se encargan de las guardias. Podemos definir el alcance de un cambio con comprobaciones, una vía de recuperación y responsabilidades claras.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es un hook de Git?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

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

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/git-hook/