

Claude Code en tu CI con GitHub Actions

Tres workflows para que Claude Code revise cada pull request, explique por qué fallan los tests y documente cuando alguien escribe @claude, con permisos mínimos, límites de tiempo y un script para probar el paso de explicación en local antes de subir nada.

PARA	Repositorios en github.com con acceso de administrador
PROBADO CON	Claude Code 2.1.238 en local · workflows validados, no ejecutados en GitHub
ACTUALIZADA	26 de octubre de 2026
DOCUMENTACIÓN	code.claude.com/docs/en/github-actions · /headless
ARTÍCULO	ecomlabs.dev/g/bot
AUTOR	Luis Morales · @luismoralesdev

Índice

Antes de
empezar

Qué vas a montar

Paso 01

Prueba el paso de explicación en local

Paso 02

Instala la app y el secreto

Paso 03

Revisión de cada pull request

Paso 04

Explicación cuando fallan los tests

Paso 05

@claude para documentar

Paso 06

Comprueba que funciona

Para
ampliarlo

Costos y variantes

Si algo
falla

Solución de problemas

Al
terminar

Checklist

Qué vas a montar

La action oficial `anthropics/claude-code-action@v1` corre Claude Code dentro de un job de GitHub Actions. Esta guía monta tres usos con ella:

WORKFLOW	CUÁNDO CORRE	QUÉ HACE
<code>claude-revision.yml</code>	Al abrir o actualizar un PR	Revisa el PR y comenta en sus líneas
<code>ci.yml</code>	En cada push y PR, solo si fallan los tests	Explica la causa en el log del run
<code>claude.yml</code>	Cuando alguien escribe <code>@claude</code>	Responde y puede subir commits, por ejemplo de documentación

CUIDADO

Los workflows siguen los ejemplos de la documentación oficial y su YAML está validado, pero no los corrimos en GitHub para esta guía. Lo que sí está probado es el paso de explicación, en local, con los scripts del paso 1.

Requisitos

- Un repositorio en github.com en el que tengas acceso de administrador.
- Una API key de la consola de Claude, o una suscripción de Claude para usar `claude setup-token`.
- Claude Code en tu equipo, para probar en local y, si quieres, para el instalador rápido.
- Para el instalador rápido, la CLI de GitHub (`gh`) con sesión iniciada.

Prueba el paso de explicación en local

Antes de gastar minutos de CI, comprueba que el prompt da una explicación útil con tus tests. Estos scripts hacen lo mismo que el paso «Explicar el fallo» de `ci.yml`, pero con `claude -p` en tu terminal:

`probar/explicar-fallo.sh`

```
#!/usr/bin/env bash
# explicar-fallo.sh: corre los tests y, si fallan, le pide a Claude Code que
# explique la causa, igual que el paso «Explicar el fallo» de ci.yml.
# Uso, desde la raíz del proyecto: bash probar/explicar-fallo.sh [opciones de claude]
# por ejemplo: bash probar/explicar-fallo.sh --model haiku
# Necesita Claude Code con sesión iniciada, Node y npm.

salida=$(npm test 2>&1)
codigo=$?
if [ "$codigo" -eq 0 ]; then
    echo "Los tests pasan: no hay nada que explicar."
```

```

    exit 0
fi

prompt="Los tests de este proyecto fallaron; su salida llega por la entrada estándar. \
Explica en español, en menos de 15 líneas: qué test falla, la causa más probable y qué \
archivo revisar. No cambies ningún archivo."

json=$(printf '%s\n' "$salida" | tail -n 200 | claude -p "$prompt" --max-turns 3 --output-format json "$@" ) ||
{ echo "explicar-fallo: claude -p no terminó bien" >&2; exit 1; }

# Del JSON de claude -p interesan tres campos: result, num_turns y total_cost_usd.
printf '%s' "$json" | node -e '
    let s = "";
    process.stdin.on("data", (d) => (s += d)).on("end", () => {
        const r = JSON.parse(s);
        console.log(r.result);
        console.log(`\n${r.num_turns} turnos, ${r.total_cost_usd.toFixed(4)} USD`);
    });'
exit "$codigo"

```

probar/explicar-fallo.ps1

```

# explicar-fallo.ps1: corre los tests y, si fallan, le pide a Claude Code que
# explique la causa, igual que el paso "Explicar el fallo" de ci.yml.
# Uso, desde la raíz del proyecto:
#   powershell -NoProfile -ExecutionPolicy Bypass -File probar\explicar-fallo.ps1 [opciones de claude]
#   por ejemplo: ... -File probar\explicar-fallo.ps1 --model haiku
# Necesita Claude Code con sesión iniciada, Node y npm.
# El texto va sin acentos: PowerShell 5.1 lee este archivo con la página de
# códigos del sistema.

# PowerShell 5.1 manda texto a un programa en ASCII y lo lee en la página de
# códigos de la consola; con UTF-8 en los dos sentidos no se pierden acentos.
$OutputEncoding = [System.Text.UTF8Encoding]::new($false)
[Console]::OutputEncoding = [System.Text.UTF8Encoding]::new($false)

# Con 2>&1, PowerShell envuelve cada línea de stderr en un ErrorRecord;
# TargetObject guarda el texto original.
$salida = & npm test 2>&1 | ForEach-Object {
    if ($_ -is [System.Management.Automation.ErrorRecord]) { "$($_.TargetObject)" } else { "$_" }
}

$codigo = $LASTEXITCODE
if ($codigo -eq 0) {
    'Los tests pasan: no hay nada que explicar.'
    exit 0
}

$prompt = 'Los tests de este proyecto fallaron; su salida llega por la entrada estándar. ' +
    'Explica en español, en menos de 15 líneas: qué test falla, la causa más probable y qué ' +
    'archivo revisar. No cambies ningún archivo.'

```

```
$json = $salida | Select-Object -Last 200 | & claude -p $prompt --max-turns 3 --output-format json @args
if ($LASTEXITCODE -ne 0) {
    [Console]::Error.WriteLine('explicar-fallo: claude -p no termino bien')
    exit 1
}

# Del JSON de claude -p interesan tres campos: result, num_turns y total_cost_usd.
$r = ($json -join "`n") | ConvertFrom-Json
$r.result
''
'({0} turnos, {1:N4} USD)' -f $r.num_turns, $r.total_cost_usd
exit $codigo
```

Desde la raíz del proyecto:

```
bash probar/explicar-fallo.sh --model haiku
powershell -NoProfile -ExecutionPolicy Bypass -File probar\explicar-fallo.ps1 --model haiku
```

Si los tests pasan, el script lo dice y termina con 0. Para ver una explicación, rompe algo a propósito. Nosotros subimos el envío de la tienda de prueba de 490 a 590 céntimos: la explicación encontró los dos tests que fallaban, la diferencia de 100 céntimos y el archivo que había que revisar. Cada ejecución con Haiku costó entre 0,048 y 0,059 USD.

La versión de PowerShell fija UTF-8 en la entrada y la salida: sin eso, PowerShell 5.1 manda a `claude` los acentos de la salida de los tests convertidos en signos de interrogación.

PASO 02

Instala la app y el secreto

Camino rápido

Abre `claude` en el repositorio y ejecuta `/install-github-app`. Instala la app de Claude, guarda el secreto y abre un PR con el workflow que elijas. Solo funciona con repositorios de github.com.

Camino manual

1. Instala la app desde github.com/apps/claude en el repositorio.
2. En *Settings* → *Secrets and variables* → *Actions*, crea uno de estos secretos:
 - `ANTHROPIC_API_KEY`: una clave de platform.claude.com. Se factura por uso.
 - `CLAUDE_CODE_OAUTH_TOKEN`: ejecuta `claude setup-token` en tu equipo y pega el resultado. Usa tu suscripción.
3. Copia los workflows a `.github/workflows/`.

Los workflows del ZIP usan `ANTHROPIC_API_KEY`. Si elegiste el token, cambia en cada uno la línea `anthropic_api_key` por:

```
claude_code_oauth_token: ${ secrets.CLAUDE_CODE_OAUTH_TOKEN }
```

Para una organización, la documentación recomienda una API key como secreto de la organización: el token queda atado a la suscripción de quien lo generó.

PASO 03

Revisión de cada pull request

`.github/workflows/claude-revision.yml`

```
# Revisión de cada pull request con el plugin code-review de Claude Code.
# Basado en el ejemplo «Run a skill» de code.claude.com/docs/en/github-actions.
# Claude deja comentarios en las líneas del PR, o uno solo si no encuentra nada.
name: Revisión de Claude

on:
  pull_request:
    types: [opened, synchronize, ready_for_review, reopened]

# Un push nuevo al mismo PR cancela la revisión anterior: no pagas dos.
concurrency:
  group: claude-revision-${{ github.event.pull_request.number }}
  cancel-in-progress: true

jobs:
  review:
    runs-on: ubuntu-latest
    timeout-minutes: 15
    permissions:
      contents: read
      pull-requests: read
      issues: read
      id-token: write
    steps:
      - uses: actions/checkout@v6
        with:
          fetch-depth: 1
      - uses: anthropics/claude-code-action@v1
        with:
          anthropic_api_key: "${{ secrets.ANTHROPIC_API_KEY }}"
          plugin_marketplaces: "https://github.com/anthropics/claude-code.git"
          plugins: "code-review@claude-code-plugins"
          prompt: "/code-review:code-review --comment ${{ github.repository }}/pull/${{
github.event.pull_request.number }}"
          claude_args: '--allowedTools "mcp__github_inline_comment__create_inline_comment"'
```

- `--comment` hace que la revisión aparezca en el PR: un comentario por hallazgo en su línea, o uno solo si no encuentra nada. Sin él, solo queda en el log.
- `--allowedTools "mcp__github_inline_comment__create_inline_comment"` es necesario aunque el plugin ya nombre esa herramienta: la action solo arranca el servidor de comentarios si aparece aquí.
- `concurrency` cancela la revisión anterior cuando llega un push nuevo al mismo PR.

Según la documentación, Claude se salta los PR en borrador, los cerrados y los que ya tienen un comentario suyo. En repositorios públicos no corre en PR de forks, porque GitHub no les pasa los secretos.

PASO 04

Explicación cuando fallan los tests

.github/workflows/ci.yml

```
# CI con explicación: si los tests fallan, Claude lee su salida y explica la
# causa en el log del run. No cambia ningún archivo.
name: CI

on:
  pull_request:
  push:
    branches: [main]

concurrency:
  group: ci-${{ github.ref }}
  cancel-in-progress: true

jobs:
  test:
    runs-on: ubuntu-latest
    timeout-minutes: 15
    permissions:
      contents: read
      id-token: write
    steps:
      - uses: actions/checkout@v6
      - uses: actions/setup-node@v7
        with:
          node-version: 22
      - run: npm ci
      - name: Tests
        id: tests
        # shell: bash activa pipefail: si npm test falla, el paso falla aunque tee no.
        shell: bash
        run: npm test 2>&1 | tee test-output.txt
      - name: Explicar el fallo
        if: failure() && steps.tests.outcome == 'failure'
        uses: anthropics/claude-code-action@v1
        with:
          anthropic_api_key: ${ secrets.ANTHROPIC_API_KEY }
          prompt: |
            Los tests de este repositorio fallaron. Su salida está en test-output.txt.
            Explica en español, en menos de 15 líneas: qué test falla, la causa más
            probable y qué archivo revisar. No cambies ningún archivo.
          claude_args: '--max-turns 5 --allowedTools "Read,Grep,Glob"
```

- `shell: bash` activa `pipefail`. Sin él, `npm test | tee` termina bien aunque los tests fallen y Claude nunca corre.
- `steps.tests.outcome == 'failure'` evita que Claude corra cuando lo que falló fue `npm ci`.
- `--allowedTools "Read,Grep,Glob"` le deja leer la salida y el código, sin herramientas para editar.
- `--max-turns 5` y `timeout-minutes: 15` ponen techo al run.

Si ya tienes un workflow de CI, no copies este entero: añade el `id` y el `tee` a tu paso de tests, el paso «Explicar el fallo» y `id-token: write` a los permisos del job.

PASO 05

@claude para documentar

`.github/workflows/claude.yml`

```
# Claude responde cuando alguien con acceso de escritura escribe @claude en un
# issue o un PR: por ejemplo, «@claude documenta la función resumen en docs/».
# Basado en el ejemplo «Respond to @claude mentions» de la documentación.
name: Claude Code

on:
  issue_comment:
    types: [created]
  pull_request_review_comment:
    types: [created]

jobs:
  claude:
    if: contains(github.event.comment.body, '@claude')
    runs-on: ubuntu-latest
    timeout-minutes: 30
    permissions:
      contents: write
      pull-requests: write
      issues: write
      id-token: write
      actions: read
    steps:
      - uses: actions/checkout@v6
        with:
          fetch-depth: 1
      - uses: anthropics/claude-code-action@v1
        with:
          anthropic_api_key: ${ secrets.ANTHROPIC_API_KEY }
          claude_args: "--max-turns 20"
```

Este workflow espera a que alguien con acceso de escritura mencione `@claude`. Algunos pedidos para documentación:

```
@claude documenta en docs/checkout.md cómo funciona el envío gratis y añade un ejemplo
@claude actualiza el README con el nuevo script npm run guides:build
@claude explica en un comentario qué cambia este PR para quien no lo escribió
```

Claude responde en un comentario que va actualizando y sube los cambios como commits. Por eso pide `contents: write`, y por eso esos commits se revisan antes de fusionar.

CUIDADO

No pases `github_token: ${ secrets.GITHUB_TOKEN }` a la action. GitHub no dispara workflows con commits hechos con ese token, así que tu CI no correría sobre los cambios de Claude.

PASO 06

Comprueba que funciona

1. Sube los workflows en una rama y abre un PR. `claude-revision.yml` debería correr y comentar en el PR.
2. En esa rama, rompe un test a propósito y haz push. En la pestaña *Actions*, el job de CI debe fallar y el paso «Explicar el fallo» debe mostrar la explicación en su log.
3. Escribe en el PR: «@claude resume en un comentario qué cambia este PR». Claude debe responder en un comentario.
4. Arregla el test y fusiona solo cuando todo esté en verde y hayas revisado lo que Claude propuso.

Mira en el log de cada run cuántos turnos usó y cuánto tardó. Con eso ajustas `--max-turns` y `timeout-minutes` a tu repositorio.

PARA AMPLIARLO

Costos y variantes

Lo que cuesta cada run

Dos cosas: minutos de GitHub Actions y tokens de Claude, o tu suscripción con el token. Como referencia de nuestras pruebas en local: explicar un fallo con Haiku, unos 0,05 USD; revisar un archivo de 20 líneas con Sonnet, unos 0,15 USD. Crece con el tamaño del diff, del `CLAUDE.md` y de lo que lea. La documentación recomienda mantener `CLAUDE.md` corto porque se lee en cada run.

Tu propio revisor

Si tienes un subagente revisor en `.claude/agents/`, el `checkout` lo deja en el runner. Puedes cambiar el `prompt` de la revisión para que lo use, con la herramienta de comentarios en `--allowedTools` si quieres que publique en el PR.

Planificar en un issue

Con `--permission-mode plan` en `claude_args`, Claude propone un plan sin editar. Sirve para que deje un plan en un issue antes de que alguien lo apruebe.

Solución de problemas

SÍNTOMA	CAUSA PROBABLE	ARREGLO
Claude no responde a <code>@claude</code>	Falta la app o el secreto, o el autor no tiene acceso de escritura	Revisa la app, el secreto y los permisos de quien comenta
Escribiste <code>/claude</code> o <code>@claude-bot</code>	La action busca la palabra <code>@claude</code> completa	Escribe <code>@claude</code>
El CI no corre sobre los commits de Claude	Pasaste <code>github_token</code> con el token por defecto	Quita esa línea
Error de autenticación	Clave o token inválidos	Pruébalos en local con <code>claude</code> antes de depurar el workflow
Los tests fallan pero Claude no corre	Falta <code>shell: bash</code> o el <code>id</code> del paso	Compara con <code>ci.yml</code> del ZIP
La revisión no aparece en el PR	Falta <code>--comment</code> o la herramienta en <code>--allowedTools</code>	Copia las dos líneas del ejemplo
Un bot dispara a Claude y falla	La action rechaza bots por defecto	Añádalo a <code>allowed_bots</code> solo si es necesario

Checklist

- ☐ `explicar-fallo` dio una explicación útil con tus tests en local.
- ☐ La app de Claude está instalada y el secreto guardado; la clave no aparece en ningún YAML.
- ☐ Cada workflow tiene solo sus permisos, más `id-token: write`.
- ☐ Los jobs tienen `timeout-minutes` y los pasos de Claude, `--max-turns`.
- ☐ Un PR de prueba recibió la revisión, la explicación del fallo y la respuesta a `@claude`.
- ☐ `main` exige revisión humana antes de fusionar.