

El linter corre solo tras cada edición de Claude Code

Un hook PostToolUse que pasa ESLint, Prettier o Biome por cada archivo que Claude edita y le devuelve los errores que quedan, en bash y en PowerShell. Todo el código está en el ZIP y está probado con las tres herramientas.

PARA	Claude Code en macOS, Linux o Windows
PROBADO CON	Claude Code 2.1.238 · ESLint 10.13 · Prettier 3.9 · Biome 2.6
ACTUALIZADA	26 de octubre de 2026
DOCUMENTACIÓN	code.claude.com/docs/en/hooks · /hooks-guide
ARTÍCULO	ecomlabs.dev/g/linter
AUTOR	Luis Morales · @luismoralesdev

Índice

Antes de empezar	Qué vas a construir
Paso 01	Instala el linter en el proyecto
Paso 02	El script en bash
Paso 03	El script en PowerShell
Paso 04	Registra el hook en settings.json
Paso 05	Prueba el script sin abrir Claude Code
Paso 06	Pruébalo dentro de Claude Code
Para ampliarlo	Rendimiento, Prettier con ESLint y reglas
Si algo falla	Solución de problemas
Al terminar	Checklist

Qué vas a construir

Un hook es un comando que Claude Code ejecuta solo, en un momento fijo de su ciclo de trabajo. El de esta guía corre en el evento `PostToolUse`: justo después de que Claude edita o crea un archivo. Claude Code le pasa al script un JSON con la ruta; el script pasa el linter solo por ese archivo y, si quedan errores, sale con código 2 para que Claude lea el reporte y los corrija.

La diferencia con escribir «corre el linter al terminar» en `CLAUDE.md` es que una instrucción depende de que el modelo se acuerde; el hook corre en cada edición.

Cómo responde el hook

SITUACIÓN	CÓDIGO DE SALIDA	QUIÉN SE ENTERA
El archivo no es <code>.ts</code> , <code>.tsx</code> ni <code>.js</code>	<code>0</code>	Nadie: el hook no hace nada
El linter no encuentra errores, o los corrige solo	<code>0</code>	Nadie
Quedan errores en el archivo	<code>2</code>	Claude, con las primeras 30 líneas del reporte
El linter no está instalado, o ESLint no puede leer su configuración	<code>1</code>	Tú, con un aviso; Claude no recibe nada

Un `PostToolUse` no puede deshacer la edición: cuando corre, el archivo ya está escrito. Lo que hace es avisarle a Claude a tiempo, mientras todavía tiene el archivo en mente.

Requisitos

- Claude Code instalado y un proyecto de Node con `package.json`.
- Uno de los tres linters instalado en el proyecto (paso 1).
- macOS o Linux: bash y `jq`. Windows: PowerShell 5.1 o 7, o Git Bash con `jq`.

Instala el linter en el proyecto

El hook usa el binario de `node_modules/.bin`, no uno global ni `npx`. Así no se descarga nada en cada edición y la versión es la de tu `package.json`. Instala el que vayas a usar:

```
# ESLint (necesita además tu eslint.config.js)
npm i -D eslint

# Prettier
npm i -D prettier

# Biome
npm i -D @biomejs/biome
```

Antes de seguir, confirma que funciona a mano sobre un archivo cualquiera, por ejemplo `npx eslint src/index.ts`. Si falla aquí, también fallará en el hook, y lo verás como un aviso con exit 1.

Después crea la carpeta de hooks en la raíz del repositorio:

```
# macOS, Linux o Git Bash
mkdir -p .claude/hooks

# PowerShell
New-Item -ItemType Directory -Force .claude\hooks
```

PASO 02

El script en bash

Guárdalo como `.claude/hooks/lint-archivo.sh`. Es el mismo archivo que viene en el ZIP.

`.claude/hooks/lint-archivo.sh`

```
#!/usr/bin/env bash

# lint-archivo.sh: hook PostToolUse de Claude Code.
# Después de cada Edit o Write sobre un .ts, .tsx o .js, pasa el linter solo por
# ese archivo. Lo que el linter no pudo corregir vuelve a Claude por el stderr.
# exit 0 = limpio o no aplica · exit 2 = errores para Claude · exit 1 = aviso para ti.
# Requiere jq: https://jqlang.org/download/

# eslint, prettier o biome.
linter="eslint"

command -v jq >/dev/null 2>&1 || { echo "lint-archivo: falta jq" >&2; exit 1; }

ruta=$(jq -r '.tool_input.file_path // empty') || { echo "lint-archivo: entrada no válida" >&2; exit 1; }
case "$ruta" in
    *.ts|*.tsx|*.js) ;;
    *) exit 0 ;;
esac

# Usa el linter instalado en el proyecto, no uno global ni uno descargado al vuelo.
raiz=${CLAUDE_PROJECT_DIR:-$PWD}
bin="$raiz/node_modules/.bin/$linter"
[ -x "$bin" ] || { echo "lint-archivo: no encuentro $bin" >&2; exit 1; }
```

```

case "$linter" in
  eslint)  salida=$(("$bin" --fix "$ruta" 2>&1) ;;
  prettier) salida=$(("$bin" --write "$ruta" 2>&1) ;;
  biome)   salida=$(("$bin" check --write "$ruta" 2>&1) ;;
  *) echo "lint-archivo: linter desconocido: $linter" >&2; exit 1 ;;
esac
codigo=$?

[ $codigo -eq 0 ] && exit 0

# ESLint sale con 2 cuando falla su configuración, no tu código: eso es para ti.
if [ "$linter" = eslint ] && [ $codigo -eq 2 ]; then
  echo "lint-archivo: ESLint no pudo correr. $(echo "$salida" | head -n 3)" >&2
  exit 1
fi

# Solo las primeras 30 líneas: suficiente para corregir, sin llenarle el contexto a Claude.
{
  echo "El linter ($linter) encontró problemas en $ruta:"
  echo "$salida" | head -n 30
} >&2
exit 2

```

Las decisiones que importan:

- **La primera línea útil elige el linter.** Cambia `linter="eslint"` por `prettier` o `biome`. Cada uno tiene su comando en el `case`.
- **Filtra por extensión antes de nada.** Un `.md` o un `.json` sale con 0 sin arrancar Node.
- **Corrige lo que se puede corregir solo.** `--fix` y `--write` dejan al linter arreglar el formato y las reglas con arreglo automático. Solo lo que queda llega a Claude.
- **Separa tus problemas de los de Claude.** ESLint sale con 2 cuando falla su configuración, no el código. Ese caso sale con 1, que es un aviso para ti.
- **Recorta el reporte.** Treinta líneas bastan para corregir; cientos llenan el contexto de Claude.

En macOS y Linux, dale permiso de ejecución:

```
chmod +x .claude/hooks/lint-archivo.sh
```

PASO 03

El script en PowerShell

Si trabajas en Windows sin Git Bash, o prefieres no instalar `jq`, usa esta versión. Guárdala como `.claude/hooks/lint-archivo.ps1`.

```
.claude/hooks/lint-archivo.ps1
```

```

# lint-archivo.ps1: hook PostToolUse de Claude Code para Windows (PowerShell 5.1 o 7).
# Después de cada Edit o Write sobre un .ts, .tsx o .js, pasa el linter solo por
# ese archivo. Lo que el linter no pudo corregir vuelve a Claude por el stderr.

```

```

# exit 0 = limpio o no aplica · exit 2 = errores para Claude · exit 1 = aviso para ti.
# Los mensajes van sin acentos: PowerShell 5.1 lee como ANSI un .ps1 UTF-8 sin BOM.

# eslint, prettier o biome.
$linter = 'eslint'

[Console]::OutputEncoding = [System.Text.Encoding]::UTF8

try {
    $entrada = [Console]::In.ReadToEnd() | ConvertFrom-Json
} catch {
    [Console]::Error.WriteLine('lint-archivo: entrada no valida')
    exit 1
}

$ruta = $entrada.tool_input.file_path
if (-not $ruta -or $ruta -notmatch '\.(ts|tsx|js)$') { exit 0 }

# Usa el linter instalado en el proyecto, no uno global ni uno descargado al vuelo.
$raiz = if ($env:CLAUDE_PROJECT_DIR) { $env:CLAUDE_PROJECT_DIR } else { (Get-Location).Path }
$bin = Join-Path $raiz "node_modules\.bin\$linter.cmd"
if (-not (Test-Path -LiteralPath $bin)) {
    [Console]::Error.WriteLine("lint-archivo: no encuentro $bin")
    exit 1
}

switch ($linter) {
    'eslint' { $salida = & $bin --fix $ruta 2>&1 }
    'prettier' { $salida = & $bin --write $ruta 2>&1 }
    'biome' { $salida = & $bin check --write $ruta 2>&1 }
    default { [Console]::Error.WriteLine("lint-archivo: linter desconocido: $linter"); exit 1 }
}

$codigo = $LASTEXITCODE
if ($codigo -eq 0) { exit 0 }

# Con 2>&1, PowerShell 5.1 envuelve cada linea de stderr en un ErrorRecord; el
# texto original esta en TargetObject (si no, las lineas vacias salen como
# "System.Management.Automation.RemoteException").
$lineas = $salida | ForEach-Object {
    if ($_ -is [System.Management.Automation.ErrorRecord]) { "$($_.TargetObject)" } else { "$_" }
}

# ESLint sale con 2 cuando falla su configuracion, no tu codigo: eso es para ti.
if ($linter -eq 'eslint' -and $codigo -eq 2) {
    [Console]::Error.WriteLine("lint-archivo: ESLint no pudo correr. $($lineas | Select-Object -First 3) -join
    ' ')")
    exit 1
}

# Solo las primeras 30 lineas: suficiente para corregir, sin llenarle el contexto a Claude.
[Console]::Error.WriteLine("El linter ($linter) encontro problemas en ${ruta}:")

```

```
$lineas | Select-Object -First 30 | ForEach-Object { [Console]::Error.WriteLine($_) }  
exit 2
```

Hace lo mismo que la versión en bash, con dos detalles propios de Windows:

- **Llama a los `.cmd`.** En Windows, npm instala `eslint.cmd`, `prettier.cmd` y `biome.cmd` en `node_modules\.bin`.
- **Limpia el `stderr`.** Con `2>&1`, PowerShell 5.1 envuelve cada línea de error del linter en un `ErrorRecord`, y las líneas vacías aparecían como `System.Management.Automation.RemoteException`. El script toma el texto original de `TargetObject`.

Los mensajes van sin acentos a propósito: PowerShell 5.1 lee un `.ps1` en UTF-8 sin BOM con la página de códigos del sistema, y los acentos salen rotos.

PASO 04

Registra el hook en settings.json

En `.claude/settings.json`, en la raíz del proyecto, para bash (macOS, Linux o Windows con Git Bash):

`.claude/settings.json`

```
{  
  "hooks": {  
    "PostToolUse": [  
      {  
        "matcher": "Edit|Write",  
        "hooks": [  
          {  
            "type": "command",  
            "command": "\"$CLAUDE_PROJECT_DIR\"/.claude/hooks/lint-archivo.sh"  
          }  
        ]  
      }  
    ]  
  }  
}
```

El `matcher` `Edit|Write` cubre las dos herramientas que cambian archivos. `$CLAUDE_PROJECT_DIR` es la raíz del proyecto; va entre comillas por si la ruta tiene espacios.

Para la versión de PowerShell, usa este archivo en su lugar. La forma `exec`, con `args`, no necesita comillas y salta la política de ejecución:

`windows/settings.json`

```
{  
  "hooks": {  
    "PostToolUse": [  
      {  
        "matcher": "Edit|Write",  
        "hooks": [  
          {  
            "type": "command",  
            "command": "exec powershell -c \"$CLAUDE_PROJECT_DIR/.claude/hooks/lint-archivo.ps1\" $args"  
          }  
        ]  
      }  
    ]  
  }  
}
```

```

{
  "type": "command",
  "command": "powershell.exe",
  "args": [
    "-NoProfile",
    "-ExecutionPolicy",
    "Bypass",
    "-File",
    "${CLAUDE_PROJECT_DIR}/.claude/hooks/lint-archivo.ps1"
  ]
}
]
}
]
}
}

```

CUIDADO

Si ya tienes un `.claude/settings.json`, no lo reemplaces: copia la clave `hooks.PostToolUse` dentro del tuyo. Si ya tienes otros hooks `PostToolUse`, agrega este como otra entrada del arreglo.

PASO 05

Prueba el script sin abrir Claude Code

Un hook es un programa que lee JSON por stdin. Puedes probarlo pasándole el mismo JSON que mandaría Claude Code:

```

# bash
echo '{"tool_name":"Edit","tool_input":{"file_path":"src/carrito.ts"}}' | bash .claude/hooks/lint-archivo.sh
echo $?

# PowerShell
'{"tool_name":"Edit","tool_input":{"file_path":"src\\carrito.ts"}}' |
powershell -NoProfile -ExecutionPolicy Bypass -File .claude\\hooks\\lint-archivo.ps1
$LASTEXITCODE

```

El ZIP trae un script que prueba los cuatro casos de una vez: un `.md` que se salta, un archivo limpio, uno con un error de sintaxis y un proyecto sin el linter instalado. Crea y borra sus propios archivos temporales en `probar/`.

```

bash probar/probar-lint.sh
powershell -NoProfile -ExecutionPolicy Bypass -File probar\\probar-lint.ps1

```

Cada caso imprime una línea que empieza con `ok` o `FALLA`, y la última dice «Todo en orden.» si pasaron los cuatro. Lo corrimos con ESLint, Prettier y Biome, en bash y en PowerShell: los doce casos de cada terminal dieron el código esperado.

Pruébalo dentro de Claude Code

1. Abre Claude Code en el proyecto y escribe `/hooks`. El hook debe aparecer bajo `PostToolUse`. Si no aparece, revisa que el JSON sea válido.
2. Pide un cambio que deje un error, por ejemplo: «Agrega al final de `src/carrito.ts` exactamente esta línea: `const sinUso = 1;`».
3. Claude recibe el reporte del linter junto al resultado de la edición. En nuestra prueba con ESLint y typescript-eslint, contestó así:

```
Listo. Agregué la línea `const sinUso = 1;` al final del archivo. ESLint reporta el error esperado porque la variable no se usa, pero la línea está exactamente como la pediste.
```

Para ver el código de salida y el tiempo de cada hook, arranca con `claude --debug-file claude.log` y busca `PostToolUse` en ese archivo. En nuestra prueba aparecieron estas dos líneas:

```
Hook PostToolUse:Edit (PostToolUse) error: El linter (eslint) encontro problemas en ...carrito.ts
Slow PostToolUse hooks: 2306ms for Edit
```

Si no le pides una línea exacta, Claude suele corregir el error en el paso siguiente y el hook vuelve a correr sobre la versión nueva.

Rendimiento, Prettier con ESLint y reglas

Rendimiento en repos grandes

El hook corre en cada `Edit` y cada `Write`. En el repositorio de `ecomlabs.dev` (412 archivos, ESLint 8.57, con la caché del sistema caliente) ESLint tardó unos 3 segundos con un archivo y unos 9 con todo el proyecto; la primera corrida en frío, 35. Pasa siempre el archivo, nunca la carpeta. Si tu linter puede colgarse, agrega `"timeout": 60` al hook en `settings.json`: el valor por defecto es de 10 minutos.

Caché de ESLint

Con `--cache`, ESLint guarda en `.eslintcache` qué archivos ya revisó. Agrégalo a `.gitignore`. La documentación de ESLint avisa que la caché no sigue dependencias entre archivos, así que las reglas que miran imports pueden dar resultados viejos.

```
eslint) salida=$(("$bin" --fix --cache "$ruta" 2>&1) ;;
```

Prettier y ESLint juntos

No registres dos hooks: según la referencia, los hooks de un mismo evento corren en paralelo y los dos reescribirían el archivo a la vez. Córrelos uno tras otro dentro del mismo script, Prettier primero:

```
"$raiz/node_modules/.bin/prettier" --write "$ruta" >/dev/null 2>&1
salida=$( "$raiz/node_modules/.bin/eslint" --fix "$ruta" 2>&1)
```

Que Claude no apague el linter

Cuando un error no tiene arreglo obvio, Claude puede agregar `// eslint-disable-next-line`. En una de nuestras pruebas lo hizo con una constante que le pedimos dejar «sin usar por ahora». Si no quieres que pase, escríbelo en tu `CLAUDE.md`:

```
- No uses comentarios eslint-disable para callar el hook del linter.
  Si una regla no tiene sentido aquí, dimelo y lo decido yo.
```

Otros tipos de archivo

Para revisar también `.jsx`, `.mjs` o `.css`, agrégalos a la línea del `case` en bash y a la expresión de extensiones en PowerShell, y confirma que tu linter los entiende.

— SI ALGO FALLA

Solución de problemas

SÍNTOMA	CAUSA PROBABLE	ARREGLO
<code>/hooks</code> no muestra el hook	JSON inválido o archivo en otra carpeta	Quita comas finales y comentarios; la ruta es <code>.claude/settings.json</code>
Aviso «no encuentro .../node_modules/.bin/eslint»	El linter no está instalado en ese proyecto	<code>npm i -D eslint</code> (o el que uses)
Aviso «ESLint no pudo correr»	Falta <code>eslint.config.js</code> o tiene un error	Corre <code>npx eslint</code> sobre un archivo a mano y arregla la configuración
Aviso «falta jq»	<code>jq</code> no está instalado	Instálalo, o usa la versión de PowerShell
El hook no corre en macOS o Linux	El script no es ejecutable	<code>chmod +x .claude/hooks/lint-archivo.sh</code>
Claude no ve los errores	El script sale con 1 en lugar de 2	Compara tu script con el del ZIP; solo exit 2 llega a Claude
Cada edición tarda varios segundos	El linter revisa la carpeta, o es la primera corrida en frío	Pasa solo el archivo; prueba <code>--cache</code> en ESLint
Un cambio hecho con un comando de terminal no se revisa	El matcher <code>Edit Write</code> no ve Bash ni PowerShell	Corre el linter completo antes del commit o en tu CI

Checklist

- ☐ `.claude/hooks/lint-archivo.sh` o `.ps1` está en el proyecto.
- ☐ La variable `linter` dice `eslint`, `prettier` o `biome`, y ese paquete está en `devDependencies`.
- ☐ `.claude/settings.json` registra el hook bajo `PostToolUse` con `"matcher": "Edit|Write"`.
- ☐ `probar-lint` termina con «Todo en orden.».
- ☐ `/hooks` muestra el hook bajo `PostToolUse`.
- ☐ Una edición con un error a propósito llega a Claude y Claude la corrige.
- ☐ Tu `CLAUDE.md` dice si Claude puede usar `eslint-disable`.
- ☐ `.claude/settings.json` está versionado para que el equipo tenga el mismo hook.