

Run your linter after every Claude Code edit

A PostToolUse hook that runs ESLint, Prettier or Biome on every file Claude edits and hands the remaining errors back to Claude, in bash and PowerShell. Every file is in the ZIP and has been tested with all three tools.

FOR	<code>Claude Code on macOS, Linux or Windows</code>
TESTED WITH	<code>Claude Code 2.1.238 · ESLint 10.13 · Prettier 3.9 · Biome 2.6</code>
UPDATED	<code>26 October 2026</code>
DOCUMENTATION	<code>code.claude.com/docs/en/hooks · /hooks-guide</code>
ARTICLE	<code>ecomlabs.dev/blog/claude-code-posttooluse-hook-linter</code>
AUTHOR	<code>Luis Morales · @luismoralesdev</code>

Contents

Before
you start

What you will build

Step 01

Install the linter in the project

Step 02

The bash script

Step 03

The PowerShell script

Step 04

Register the hook in settings.json

Step 05

Test the script without opening Claude Code

Step 06

Try it inside Claude Code

Going
further

Performance, Prettier with ESLint, and rules

If
something
fails

Troubleshooting

When you
finish

Checklist

What you will build

A hook is a command Claude Code runs on its own at a fixed point in its lifecycle. The one in this guide runs on the `PostToolUse` event: right after Claude edits or creates a file. Claude Code pipes the script a JSON object with the path; the script lints that one file and, if errors remain, exits with code 2 so Claude reads the report and fixes them.

The difference from writing "run the linter when you're done" in `CLAUDE.md` is that an instruction depends on the model remembering; the hook runs on every edit.

How the hook answers

SITUATION	EXIT CODE	WHO HEARS ABOUT IT
The file is not <code>.ts</code> , <code>.tsx</code> or <code>.js</code>	0	Nobody: the hook does nothing
The linter finds no errors, or fixes them itself	0	Nobody
Errors remain in the file	2	Claude, with the first 30 lines of the report
The linter is not installed, or ESLint cannot load its configuration	1	You, as a notice; Claude gets nothing

A `PostToolUse` hook cannot undo the edit: by the time it runs, the file is written. What it does is tell Claude in time, while it still has the file in mind.

Requirements

- Claude Code installed and a Node project with a `package.json`.
- One of the three linters installed in the project (step 1).
- macOS or Linux: bash and `jq`. Windows: PowerShell 5.1 or 7, or Git Bash with `jq`.

STEP 01

Install the linter in the project

The hook uses the binary in `node_modules/.bin`, not a global one and not `npx`. That way nothing is downloaded on each edit and the version is the one in your `package.json`. Install the one you will use:

```
# ESLint (also needs your eslint.config.js)
npm i -D eslint

# Prettier
npm i -D prettier

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

Before going on, check that it works by hand on any file, for example `npx eslint src/index.ts`. If it fails here, it will fail in the hook too, and you will see it as a notice with exit 1.

Then create the hooks folder at the repository root:

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

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

STEP 02

The bash script

Save it as `.claude/hooks/lint-file.sh`. It is the same file that comes in the ZIP.

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

```
#!/usr/bin/env bash
# lint-file.sh: Claude Code PostToolUse hook.
# After every Edit or Write on a .ts, .tsx or .js file, runs the linter on that
# file only. Whatever the linter could not fix goes back to Claude on stderr.
# exit 0 = clean or not applicable · exit 2 = errors for Claude · exit 1 = notice for you.
# Requires jq: https://jqlang.org/download/

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

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

path=$(jq -r '.tool_input.file_path // empty') || { echo "lint-file: invalid input" >&2; exit 1; }
case "$path" in
  *.ts|*.tsx|*.js) ;;
  *) exit 0 ;;
esac

# Use the linter installed in the project, not a global one or one downloaded on the fly.
root=${CLAUDE_PROJECT_DIR:-$PWD}
bin="$root/node_modules/.bin/$linter"
[ -x "$bin" ] || { echo "lint-file: cannot find $bin" >&2; exit 1; }

case "$linter" in
  eslint) output=$("$bin" --fix "$path" 2>&1) ;;
  prettier) output=$("$bin" --write "$path" 2>&1) ;;
  biome) output=$("$bin" check --write "$path" 2>&1) ;;
  *) echo "lint-file: unknown linter: $linter" >&2; exit 1 ;;
esac
code=$?
[ $code -eq 0 ] && exit 0
```

```
# ESLint exits 2 when its configuration fails, not your code: that one is for you.
if [ "$linter" = eslint ] && [ $code -eq 2 ]; then
    echo "lint-file: ESLint could not run. $(echo "$output" | head -n 3)" >&2
    exit 1
fi

# The first 30 lines only: enough to fix the code without flooding Claude's context.
{
    echo "The linter ($linter) found problems in $path:"
    echo "$output" | head -n 30
} >&2
exit 2
```

The decisions that matter:

- **The first working line picks the linter.** Change `linter="eslint"` to `prettier` or `biome`. Each has its own command in the `case`.
- **It filters by extension first.** A `.md` or a `.json` exits 0 without starting Node.
- **It fixes what can be fixed automatically.** `--fix` and `--write` let the linter fix formatting and auto-fixable rules. Only what is left reaches Claude.
- **It keeps your problems apart from Claude's.** ESLint exits 2 when its configuration fails, not the code. That case exits 1, which is a notice for you.
- **It trims the report.** Thirty lines are enough to fix the code; hundreds fill Claude's context.

On macOS and Linux, make it executable:

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

STEP 03

The PowerShell script

If you work on Windows without Git Bash, or would rather not install `jq`, use this version. Save it as `.claude/hooks/lint-file.ps1`.

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

```
# lint-file.ps1: Claude Code PostToolUse hook for Windows (PowerShell 5.1 or 7).
# After every Edit or Write on a .ts, .tsx or .js file, runs the linter on that
# file only. Whatever the linter could not fix goes back to Claude on stderr.
# exit 0 = clean or not applicable · exit 2 = errors for Claude · exit 1 = notice for you.

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

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

try {
    $hookInput = [Console]::In.ReadToEnd() | ConvertFrom-Json
} catch {
```

```

[Console]::Error.WriteLine('lint-file: invalid input')
exit 1
}

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

# Use the linter installed in the project, not a global one or one downloaded on the fly.
$root = if ($env:CLAUDE_PROJECT_DIR) { $env:CLAUDE_PROJECT_DIR } else { (Get-Location).Path }
$bin = Join-Path $root "node_modules\.bin\$linter.cmd"
if (-not (Test-Path -LiteralPath $bin)) {
    [Console]::Error.WriteLine("lint-file: cannot find $bin")
    exit 1
}

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

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

# With 2>&1, PowerShell 5.1 wraps every stderr line in an ErrorRecord; the
# original text is in TargetObject (otherwise empty lines come out as
# "System.Management.Automation.RemoteException").
$lines = $output | ForEach-Object {
    if ($_ -is [System.Management.Automation.ErrorRecord]) { "$($_.TargetObject)" } else { "$_" }
}

# ESLint exits 2 when its configuration fails, not your code: that one is for you.
if ($linter -eq 'eslint' -and $code -eq 2) {
    [Console]::Error.WriteLine("lint-file: ESLint could not run. $((($lines | Select-Object -First 3) -join '
    '))")
    exit 1
}

# The first 30 lines only: enough to fix the code without flooding Claude's context.
[Console]::Error.WriteLine("The linter ($linter) found problems in ${path}:")
$lines | Select-Object -First 30 | ForEach-Object { [Console]::Error.WriteLine($_) }
exit 2

```

It does the same as the bash version, with two Windows-specific details:

- **It calls the `.cmd` shims.** On Windows, npm installs `eslint.cmd`, `prettier.cmd` and `biome.cmd` in `node_modules\.bin`.
- **It cleans up stderr.** With `2>&1`, PowerShell 5.1 wraps every error line from the linter in an `ErrorRecord`, and empty lines showed up as `System.Management.Automation.RemoteException`. The script takes the original text from `TargetObject`.

The messages are plain ASCII on purpose: PowerShell 5.1 reads a UTF-8 `.ps1` without a BOM using the system code page, and accented characters come out garbled.

STEP 04

Register the hook in settings.json

In `.claude/settings.json`, at the project root, for bash (macOS, Linux or Windows with Git Bash):

`.claude/settings.json`

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

The `Edit|Write` matcher covers the two tools that change files. `$CLAUDE_PROJECT_DIR` is the project root; it is quoted in case the path contains spaces.

For the PowerShell version, use this file instead. Exec form, with `args`, needs no quoting and gets past the execution policy:

`windows/settings.json`

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "powershell.exe",
            "args": [
              "-NoProfile",
              "-ExecutionPolicy",
              "Bypass",
              "-File",
              "${CLAUDE_PROJECT_DIR}/.claude/hooks/lint-file.ps1"
            ]
          }
        ]
      }
    ]
  }
}
```

```
    ]
  }
]
}
}
```

CAUTION

If you already have a `.claude/settings.json`, do not replace it: copy the `hooks.PostToolUse` key into yours. If you already have other `PostToolUse` hooks, add this one as another entry in the array.

STEP 05

Test the script without opening Claude Code

A hook is a program that reads JSON on stdin. You can test it by piping it the same JSON Claude Code would send:

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

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

The ZIP includes a script that checks all four cases at once: a `.md` that is skipped, a clean file, one with a syntax error and a project without the linter installed. It creates and deletes its own temporary files in `test/`.

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

Each case prints a line starting with `ok` or `FAIL`, and the last one says "All good." if all four passed. We ran it with ESLint, Prettier and Biome, in bash and in PowerShell: all twelve cases per shell returned the expected code.

STEP 06

Try it inside Claude Code

1. Open Claude Code in the project and type `/hooks`. The hook should be listed under `PostToolUse`. If it is not, check that the JSON is valid.
2. Ask for a change that leaves an error, for example: "Append exactly this line to `src/cart.ts`: `const unused = 1;`".
3. Claude gets the linter report next to the edit result. In our test with ESLint and `typescript-eslint` (a Spanish-language project), Claude replied that it had added the line exactly as asked and that ESLint reported the expected error because the variable is unused.

To see each hook's exit code and timing, start with `claude --debug-file claude.log` and search that file for `PostToolUse`. In our test these two lines showed up (the hook's message was in Spanish):

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

If you do not ask for an exact line, Claude usually fixes the error on its next step, and the hook runs again on the new version.

GOING FURTHER

Performance, Prettier with ESLint, and rules

Performance in large repos

The hook runs on every `Edit` and every `Write`. In the `ecomlabs.dev` repository (412 files, ESLint 8.57, warm system cache) ESLint took about 3 seconds on one file and about 9 on the whole project; the first cold run took 35. Always pass the file, never the folder. If your linter can hang, add `"timeout": 60` to the hook in `settings.json`: the default is 10 minutes.

ESLint's cache

With `--cache`, ESLint records which files it already checked in `.eslintcache`. Add it to `.gitignore`. The ESLint docs warn that the cache does not track dependencies between files, so rules that follow imports can report stale results.

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

Prettier and ESLint together

Do not register two hooks: per the reference, hooks for the same event run in parallel, and both would rewrite the file at once. Run them one after the other inside the same script, Prettier first:

```
"$root/node_modules/.bin/prettier" --write "$path" >/dev/null 2>&1
output=$(("$root/node_modules/.bin/eslint" --fix "$path" 2>&1)
```

Keep Claude from silencing the linter

When an error has no obvious fix, Claude may add `// eslint-disable-next-line`. In one of our tests it did that with a constant we asked to keep "unused for now". If you do not want that, say so in your `CLAUDE.md`:

```
- Do not use eslint-disable comments to quiet the linter hook.
  If a rule makes no sense here, tell me and I will decide.
```

Other file types

To also check `.jsx`, `.mjs` or `.css`, add them to the `case` line in bash and to the extension pattern in PowerShell, and confirm your linter understands them.

Troubleshooting

SYMPTOM	LIKELY CAUSE	FIX
<code>/hooks</code> does not list the hook	Invalid JSON or the file is in another folder	Remove trailing commas and comments; the path is <code>.claude/settings.json</code>
Notice "cannot find .../node_modules/.bin/eslint"	The linter is not installed in that project	<code>npm i -D eslint</code> (or the one you use)
Notice "ESLint could not run"	<code>eslint.config.js</code> is missing or has an error	Run <code>npx eslint</code> on a file by hand and fix the configuration
Notice "jq is missing"	<code>jq</code> is not installed	Install it, or use the PowerShell version
The hook does not run on macOS or Linux	The script is not executable	<code>chmod +x .claude/hooks/lint-file.sh</code>
Claude does not see the errors	The script exits 1 instead of 2	Compare your script with the one in the ZIP; only exit 2 reaches Claude
Every edit takes several seconds	The linter checks the folder, or it is the first cold run	Pass only the file; try <code>--cache</code> with ESLint
A change made by a shell command is not checked	The <code>Edit Write</code> matcher does not see Bash or PowerShell	Run the full linter before committing or in your CI

Checklist

- ☐ `.claude/hooks/lint-file.sh` or `.ps1` is in the project.
- ☐ The `linter` variable says `eslint`, `prettier` or `biome`, and that package is in `devDependencies`.
- ☐ `.claude/settings.json` registers the hook under `PostToolUse` with `"matcher": "Edit|Write"`.
- ☐ `test-lint` ends with "All good."
- ☐ `/hooks` lists the hook under `PostToolUse`.
- ☐ An edit with a deliberate error reaches Claude, and Claude fixes it.
- ☐ Your `CLAUDE.md` says whether Claude may use `eslint-disable`.
- ☐ `.claude/settings.json` is committed so the team gets the same hook.