EcomLabs

GUÍA PASO A PASO

CLAUDE.md corto: procedimientos a skills y reglas a hooks

Una plantilla de CLAUDE.md corta, una skill de despliegue que solo corre cuando la invocas, un hook que hace cumplir una regla y un script en Node que marca qué líneas sacar del CLAUDE.md que ya tienes. Con la checklist.

PARA	Proyectos que usan Claude Code y tienen un CLAUDE.md que creció demasiado
PROBADO CON	Node 22.22.3 · hook y script probados con JSON y archivos de ejemplo · YAML validado con el paquete yaml
ACTUALIZADA	19 de noviembre de 2026
DOCUMENTACIÓN	code.claude.com/docs/en/memory · /skills · /hooks
ARTÍCULO	ecomlabs.dev/g/reglas
AUTOR	Luis Morales · @luismoralesdev

Índice

Paso 01	La plantilla de CLAUDE.md
Paso 02	Un procedimiento como skill
Paso 03	Una regla obligatoria como hook
Paso 04	Podar el CLAUDE.md que ya tienes

PASO 01

La plantilla de CLAUDE.md

Guárdala como `CLAUDE.md` en la raíz del repo y cambia todo lo que va entre `< >`. Se carga completa en cada sesión, así que cada línea debe servirle a todas. Meta: menos de 200 líneas, contando los imports con `@`, que se cargan junto con el archivo. Fuera: el árbol de carpetas, las dependencias y todo lo que Claude lee del código.

```
# <Nombre del proyecto>

<Una frase: qué es y con qué se construye.> Gestor de paquetes: <pnpm>.

## Comandos
- Desarrollo: <pnpm dev>
- Pruebas: <pnpm test> · Tipos: <pnpm typecheck>
- Antes de dar algo por terminado: <pnpm test y pnpm typecheck>

## Convenciones
- <Una decisión que no se ve en un solo archivo, escrita para poder comprobarla.>
- <Ejemplo: los precios se guardan en centavos enteros y se formatean solo en la UI.>

## Trampas conocidas
- <Algo que rompe y no se ve en el código. Ejemplo: el seed borra la tabla de pedidos.>

## Dónde está cada cosa
- Procedimientos: .claude/skills/ (migraciones, release).
- Despliegue: lo hago yo con /deploy; tú no despliegues.
- Reglas por carpeta: .claude/rules/.

<!-- Para personas (Claude no lo ve): las reglas obligatorias son hooks
     en .claude/settings.json; lo personal va en CLAUDE.local.md. -->
```

PASO 02

Un procedimiento como skill

Guárdalo como `.claude/skills/deploy/SKILL.md`. Con `disable-model-invocation: true`, Claude no puede invocarla por su cuenta y su descripción no ocupa contexto: solo corre cuando escribes `/deploy`. Para procedimientos sin efectos, como crear una migración, quita esa línea. Cambia los comandos por los tuyos:

```
---
name: deploy
description: Despliega a producción el commit actual después de pasar las comprobaciones.
disable-model-invocation: true
---

# Desplegar
```

```
1. Corre git status. Si hay cambios sin commit, detente y avísame.
2. Corre pnpm test y pnpm build. Si algo falla, detente y muestra el error.
3. Corre bash scripts/deploy.sh y espera a que termine.
4. Comprueba que la URL de producción responde con 200.
5. Responde con el commit desplegado y el resultado de cada paso.
```

En una skill que Claude sí puede invocar, la `description` es lo único que ve en cada sesión: pon primero qué hace y luego cuándo usarla. Tras compactar, Claude Code vuelve a adjuntar los primeros 5,000 tokens del cuerpo, así que lo importante va arriba.

PASO 03

Una regla obligatoria como hook

"Nunca uses npm" en CLAUDE.md es una petición. Este hook la hace cumplir. Regístralo en `.claude/settings.json`:

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": "node",
            "args": ["${CLAUDE_PROJECT_DIR}/.claude/hooks/no-npm.mjs"]
          }
        ]
      }
    ]
  }
}
```

La forma `command` + `args` no usa shell: Claude Code sustituye `${CLAUDE_PROJECT_DIR}` como texto plano, así que funciona igual con Git Bash o PowerShell. El script, en `.claude/hooks/no-npm.mjs`:

```
// Hook PreToolUse: este repo usa pnpm, así que bloquea cualquier comando de Bash que corra npm o npx.
import { readFileSync } from "node:fs";

let input;
try {
  input = JSON.parse(readFileSync(0, "utf8")); // Claude Code manda el evento como JSON por stdin
} catch {
  process.exit(0); // entrada ilegible: sin decisión, sigue el flujo normal de permisos
}

const command = input.tool_input?.command ?? "";
// npm o npx al inicio del comando o después de ; & | ( o un salto de línea
```

```
if (/(^|[;&|(\n]\s*)(npm|npx)(\s|$)/.test(command)) {
  console.error("Este repo usa pnpm. Usa pnpm en lugar de npm, y pnpm dlx en lugar de npx.");
  process.exit(2); // salida 2 en PreToolUse: la llamada se bloquea y Claude lee stderr
}
process.exit(0);
```

Pruébalo antes de confiar en él, pasándole JSON a mano:

```
echo '{"tool_name":"Bash","tool_input":{"command":"cd app && npm test"}}' | node .claude/hooks/no-npm.mjs;
echo "exit $?"
# esperado: el mensaje de pnpm y exit 2
echo '{"tool_name":"Bash","tool_input":{"command":"pnpm test"}}' | node .claude/hooks/no-npm.mjs; echo "exit
$?"
# esperado: exit 0
```

Con salida 2 la llamada se bloquea y Claude lee stderr; cualquier otro código distinto de 0 no bloquea. No detecta `npm` dentro de otro script ni `sudo npm`: es una guarda, no un sandbox.

PASO 04

Podar el CLAUDE.md que ya tienes

Lee tu CLAUDE.md línea por línea (o córrele el script `claude-md-check.mjs` del artículo, que marca candidatos) y decide el destino de cada una:

SI LA LÍNEA ES…	VA A
Un comando, una convención o un dato que toda sesión necesita	CLAUDE.md
Un procedimiento o material de referencia	`.claude/skills/`
Un procedimiento con efectos	Skill con `disable-model-invocation: true`
Una regla de una carpeta o tipo de archivo	`.claude/rules/` con `paths`
Una regla que debe cumplirse siempre	Hook o regla `deny` en permisos
Algo que Claude deduce del código	Se borra
Una preferencia tuya	`CLAUDE.local.md`

Luego corre `/doctor`, que propone recortes, y revisa `/context all` en una sesión nueva.