

# DEVGROUND PARA VIBE CODERS

REVISA LO QUE CAMBIA LA IA  
ANTES DE SUBIRLO



# LO QUE PASA CUANDO NADIE REVISA

Armando el sitio donde está esta guía, Claude agregó una línea para apagar una regla de ESLint que en el proyecto ni siquiera existe. Yo no la vi. La vio el hook de pre-commit, que corrió el lint, dio error y no dejó guardar el commit. Lo arreglamos en un minuto.

Ese hook no lo configuré a mano, lo instaló devground. Es una herramienta mía, gratis y de código abierto.

Cuando haces vibe coding, el código lo escribe la IA y casi nunca lo revisa nadie antes de guardarlo. Le pides un botón y de paso toca otro archivo que no le pediste. O pega una clave para probar algo rápido. O se olvida de una regla que le diste ayer, porque se la dijiste en el chat y el chat no queda guardado en el proyecto.

Nada de eso falla el mismo día. Falla una semana después, y si todos tus commits dicen «arreglos», encontrar cuál lo rompió te toma más tiempo que arreglarlo.

## LO QUE HACE DEVGROUND

En un equipo esto se resuelve con herramientas que revisan cada cambio antes de guardarlo. Configurarlas toma una tarde, y si estás construyendo tu primera app es probable que no quieras gastarla en eso. devground las instala y las deja conectadas entre sí.

# LO QUE CORRE EN CADA COMMIT

Primero, un `AGENTS.md` donde escribes las reglas de tu proyecto una sola vez. devground lo enlaza como `CLAUDE.md`, `.cursorrules` y los de Copilot, Gemini y Codex, entonces da lo mismo con cuál abras el proyecto, porque todas leen las mismas reglas.

GITLEAKS

## CLAVES

Si encuentra algo que parece una clave o un token, bloquea el commit y te dice dónde está. Se instala aparte; sin él el commit pasa, pero sin esa revisión.

ESLINT · PRETTIER

## ERRORES Y FORMATO

Solo en los archivos que cambiaste. También avisa cuando un archivo crece demasiado o aparece `any`, dos atajos que la IA toma seguido.

COMMITLINT

## MENSAJES

Te pide decir qué tipo de cambio es: `feat:`, `fix:`, `docs:`. Al principio molesta un poco, pero es lo que después te deja buscar dónde se rompió algo.

VITEST · KNOWLEDGE/

## TESTS Y CRITERIO

Un trinquete de cobertura: el mínimo sube con `test:coverage` y, si la IA borra tests, el chequeo falla. Y decisiones de arquitectura escritas, para que Claude tenga un criterio y no elija la de siempre.

# UN COMANDO DESDE LA RAÍZ

Para proyectos JavaScript o TypeScript (Node, React, Next.js), con Node 20 o superior.

## 1

**Con asistente.** Detecta tu stack y te pregunta qué instalar.

```
npx devground-init
```

## 2

**Todo, sin preguntas.**

```
npx devground-init --yes
```

## 3

**Solo las reglas para la IA, sin hooks ni tests.**

```
npx devground-init --preset agents-only
```

## DOS COSAS QUE CONVIENE SABER

**No pisa lo que ya tienes.** Si ya existe un archivo de configuración de ESLint, TypeScript, commitlint o lint-staged, lo deja intacto.

**Instala gitleaks aparte.** Sin él, el commit no se bloquea, pero el escaneo de claves se salta. En Mac: `brew install gitleaks`

# PARA SABER SI QUEDÓ FUNCIONANDO

Haz cualquier cambio y trata de guardarlo con un mensaje vago.

```
git commit -am "arreglos"
```

**RECHAZADO**

Lo va a rechazar, porque el mensaje no dice qué tipo de cambio es. Prueba de nuevo así:

```
git commit -am "fix: corrige el botón de pago"
```

**GUARDADO**

PREFIJO

CUÁNDO SE USA

|                  |                                          |
|------------------|------------------------------------------|
| <b>feat:</b>     | Algo nuevo que el usuario puede usar     |
| <b>fix:</b>      | Arreglas algo que no funcionaba          |
| <b>docs:</b>     | Solo documentación                       |
| <b>refactor:</b> | Reordenas código sin cambiar lo que hace |
| <b>test:</b>     | Agregas o cambias tests                  |

# SI PREFIERES QUE LO INSTALE CLAUDE

Abre Claude Code en tu proyecto y pega este prompt.

```
Instala devground en este proyecto con `npx devground-init --yes`.
```

Cuando termine:

1. Corre los tests y confirma que siguen pasando. Si algo falla, explícame por qué antes de tocar nada.
2. Corre `test:coverage` una vez para fijar el mínimo de cobertura.
3. Dime qué archivos creó y para qué sirve cada uno, en una línea por archivo.
4. Completa el AGENTS.md con las reglas de este proyecto que encuentres en el código: stack, cómo se corren los tests y qué no se debe tocar. Muéstramelo antes de guardarlo.

No cambies código de la aplicación en este paso.

El paso 4 es el que más te va a servir. devground te deja un AGENTS.md con reglas generales de buenas prácticas, pero no sabe nada de tu proyecto en particular, entonces lo más rápido es que Claude saque esas reglas del código y tú revises lo que escribió.

# ANTES DE SEGUIR PROGRAMANDO

- ☐ Corriste `npx devground-init` en la raíz.
- ☐ Instalaste `gitleaks`.
- ☐ Un commit «arreglos» se rechaza.
- ☐ Corriste `test:coverage` una vez.
- ☐ Tu `AGENTS.md` dice qué no se toca.
- ☐ Tu IA responde con ese archivo si le preguntas por las reglas.

Si todo eso se cumple, puedes seguir programando como siempre. Los controles corren solos.

`devground` es gratis y de código abierto: [github.com/elkisdm/devground](https://github.com/elkisdm/devground) · Sígueme en `Elkis Dev` para más desarrollo de software, IA y tecnología, explicados simple.