GUÍA PASO A PASO

Cómo reportarle un bug a la IA

Una plantilla para pedirle a una IA que arregle un bug sin que tenga que adivinar: el error exacto, cómo reproducirlo, dónde buscar y la causa antes del cambio. Con tres ejemplos llenos: frontend, API y tienda Shopify.

PARA	Claude Code, Cursor o cualquier agente que lea tu código
PROBADO CON	Node 22.22 · ejemplos de carrito, React y API corridos en local
ACTUALIZADA	30 de octubre de 2026
DOCUMENTACIÓN	platform.claude.com · code.claude.com/docs
ARTÍCULO	ecomlabs.dev/g/bug
AUTOR	Luis Morales · @luismoralesdev

Índice

Paso 01	Copia la plantilla
Paso 02	Llena el error exacto
Paso 03	Escribe cómo reproducirlo
Paso 04	Di dónde buscar
Paso 05	Pide la causa, luego el arreglo
Ejemplo 1	Frontend: una lista de React
Ejemplo 2	API: un JSON que no es JSON
Ejemplo 3	Tienda Shopify: se agrega otra talla
Al terminar	Checklist

PASO 01

Copia la plantilla

Le da a la IA lo que no puede adivinar y le pide la causa antes de tocar código. Llénala en orden; si una sección no aplica, escribe por qué («no hay mensaje de error»): eso también es información.

```
## Error
<mensaje completo, sin resumir>
<stack trace o salida completa>

## Cómo reproducirlo
Comando o pasos: <npm test, una URL, los clics en orden>
Qué hice: <...>
Qué esperaba: <resultado correcto, con números si hay>
Qué pasó: <lo que viste>
¿Siempre o a veces?: <siempre / a veces / solo en producción>

## Dónde buscar
Archivo o función: <...>
Cambio reciente: <commit, PR o qué se agregó>
Lo que ya descarté: <...>

## Cómo quiero que trabajes
1. Antes de cambiar código, explícame la causa. No edites todavía.
2. Cuando te lo confirme, arregla la causa, no el síntoma.
3. Agrega un test que hoy falle y pase con el arreglo.
4. Corre <comando de pruebas> y pégame la salida.
```

PASO 02

Llena el error exacto

Copia el texto, no lo resumas. El mensaje exacto se puede buscar en el código; «sale algo de undefined», no. Dónde encontrarlo:

DÓNDE PASA	QUÉ COPIAR
Navegador	El texto de la consola de DevTools, con sus líneas «at …»
Una petición	De la pestaña Network: URL, código de estado, `content-type` y cuerpo de la respuesta
Terminal o pruebas	La salida completa del comando; con Claude Code puedes pasarla directo: `cat error.log | claude -p "..."`
Servidor	Las líneas del log alrededor del error, con su hora

Del stack trace, lo más útil es la primera línea que menciona un archivo de tu proyecto.

> **CUIDADO**
>
> Antes de pegar logs o respuestas, quita tokens, contraseñas, llaves de API y datos de clientes. Un error no necesita el valor real de un secreto para entenderse: cámbialo por `<TOKEN>`.

PASO 03

Escribe cómo reproducirlo

- **Qué hiciste**: los pasos exactos, en orden. Si hay datos, los datos (el SKU, el código postal, el usuario de prueba).
- **Qué esperabas**: el resultado correcto. Es lo que le dice a la IA cuándo terminó.
- **Qué pasó**: lo que viste, sin interpretarlo todavía.
- **¿Siempre o a veces?** Un error intermitente suele apuntar a tiempos, caché o datos, no a una sola línea.

Si puedes, conviértelo en una prueba que falle. Es la reproducción que la IA puede correr sola, antes y después del arreglo.

PASO 04

Di dónde buscar

Lo que sabes y el código no dice: qué cambió, desde cuándo falla, qué parte es nueva. En Claude Code, escribe `@` y la ruta para incluir el archivo. Si usas git, `git log --oneline -5` te da los últimos commits para citar el sospechoso. Si no tienes idea, dilo y pide que investigue sin editar.

PASO 05

Pide la causa, luego el arreglo

Cuando la IA explique la causa, revísala: ¿explica el error y el resultado que viste? Si sí, pide el arreglo:

```
La causa tiene sentido. Arregla la causa, no el síntoma.
Agrega un test que hoy falle y corre npm test al terminar.
Pégame el comando que corriste y su salida.
```

Si la explicación no cuadra con lo que viste, dilo con el dato que no cuadra, antes de que exista código nuevo.

EJEMPLO 1

Frontend: una lista de React

El componente carga productos en un `useEffect`. El mensaje de error lo comprobamos en local, renderizando el componente con `react-dom/server`; la línea «at» y los pasos son ilustrativos.

```
const [productos, setProductos] = useState();
// ...
return <ul>{productos.map((p) => <li key={p.id}>{p.nombre}</li>)}</ul>;
```

```
## Error
TypeError: Cannot read properties of undefined (reading 'map')
    at ListaProductos (ListaProductos.jsx:10:24)

## Cómo reproducirlo
Pasos: abro /productos.
Esperaba: la lista de productos.
Pasó: pantalla en blanco y el error en la consola. Siempre, en la primera carga.

## Dónde buscar
src/components/ListaProductos.jsx. Empezó cuando movimos la carga a useEffect.
```

La causa: en el primer render, `useEffect` todavía no corrió y `productos` vale `undefined`. El arreglo va en el estado inicial: lo comprobamos, y con `useState([])` el componente renderiza sin error. Si además quieres un «Cargando…», pídelo aparte: es otra decisión, no parte del bug.

EJEMPLO 2

API: un JSON que no es JSON

Este lo corrimos en local con Node 22: un servidor que solo tiene `/api/envios` y un cliente que llama a `/api/envio`.

```
const res = await fetch(`/api/envio?cp=${cp}`);
const datos = await res.json();
```

```
## Error
SyntaxError: Unexpected token '<', "<!DOCTYPE "... is not valid JSON
En Network: GET /api/envio?cp=06700, estado 404, content-type text/html

## Cómo reproducirlo
Pasos: en el checkout escribo el código postal 06700.
Esperaba: el costo de envío ({"costo": 9900}).
Pasó: no aparece el costo y sale el error en la consola. Siempre.

## Dónde buscar
La llamada está en src/checkout/envio.js. Empezó después de reordenar
las rutas de la API.
```

El dato clave es la línea de Network: el error de `JSON.parse` dice que la respuesta empieza con `<`, es decir, HTML. El 404 dice por qué: la URL no existe. Sin esa línea, la IA podría ponerse a «arreglar» el parseo. En nuestra prueba, con la URL correcta la respuesta fue `200 { costo: 9900 }`. Vale la pena pedir también que el cliente revise `res.ok` antes de llamar a `res.json()`.

EJEMPLO 3

Tienda Shopify: se agrega otra talla

Ejemplo ilustrativo: no lo corrimos en una tienda real. El formulario de producto del tema guarda la variante en un input oculto; validamos esta línea de Liquid con Theme Check:

```
<input type="hidden" name="id" value="{{ product.selected_or_first_available_variant.id }}">
```

```
## Error
No hay mensaje. En Network, la petición POST a /cart/add.js manda el id
44100000000001, que es la variante S. La M es 44100000000002 (ids ilustrativos).

## Cómo reproducirlo
Pasos: abro /products/playera-basica, elijo talla M y doy clic en
"Agregar al carrito".
Esperaba: "Playera básica - M" en el carrito.
Pasó: se agrega la S. Siempre, con cualquier talla que no sea la primera.

## Dónde buscar
El input oculto name="id" del formulario de producto y el JavaScript del
selector de variantes. Empezó al cambiar el selector por botones.
```

Aquí no hay error que copiar, así que la evidencia es la petición: qué id se mandó y cuál debía ir. El input arranca con la primera variante disponible; al elegir otra talla, el JavaScript del tema tiene que actualizar ese valor. Si el selector nuevo no lo hace, el formulario sigue mandando el id inicial.

> **CUIDADO**
>
> Prueba el arreglo en un tema sin publicar o en una tienda de desarrollo, nunca en el tema publicado.

AL TERMINAR

Checklist

- [ ] El mensaje de error va completo, copiado como texto.
- [ ] Va el stack trace, o la petición con su estado y su respuesta.
- [ ] Escribiste qué hiciste, qué esperabas y qué pasó.
- [ ] Dijiste si pasa siempre o a veces.
- [ ] Señalaste archivo, función o cambio reciente.
- [ ] Quitaste tokens, contraseñas y datos de clientes.
- [ ] Pediste la causa antes del cambio.
- [ ] Pediste un test que hoy falle y la salida de las pruebas.