EcomLabs

GUÍA PASO A PASO

Proteger tu .env de los agentes de IA

Las reglas deny de settings.json que impiden que Claude Code lea tus archivos de secretos, qué cubren y qué no, dónde debe vivir cada llave y la checklist para rotar una llave que se filtró.

PARA	Proyectos donde trabajas con Claude Code y guardas llaves de API en un .env
PROBADO CON	Node 22.22.3 · Ajv 6 contra el esquema de SchemaStore · git 2.54 · ayuda local de Google Cloud SDK 581.0.0
ACTUALIZADA	19 de noviembre de 2026
DOCUMENTACIÓN	code.claude.com/docs/en/permissions
ARTÍCULO	ecomlabs.dev/g/llaves
AUTOR	Luis Morales · @luismoralesdev

Índice

Antes de empezar	Qué bloquea una regla deny de Read
Paso 01	El settings.json del proyecto
Paso 02	Una red para todos tus proyectos
Paso 03	Comprobar que funciona
Si se filtró	Checklist de rotación
Al terminar	Checklist

ANTES DE EMPEZAR

Qué bloquea una regla deny de Read

Claude Code lee archivos de tu directorio de trabajo sin pedir aprobación, y comandos como `cat` o `grep` corren sin preguntar. Una regla `deny` de `Read` cambia eso para las rutas que nombres. Según la documentación de permisos (revisada el 19 de noviembre de 2026):

- **Cubre** las herramientas de archivos de Claude (Read, y como mejor esfuerzo Grep y Glob), las menciones con `@`, el contexto del IDE, los comandos que Claude Code reconoce en Bash (`cat`, `head`, `tail`, `sed`, `tee`) y las redirecciones como `< .env`. También bloquea Edit y Write sobre la misma ruta.
- **No cubre** un comando que lee sin nombrar el archivo, como `grep -r pattern .`, ni un script de Python o Node que lo abre por su cuenta, ni las variables que ya están en el entorno de tu terminal: no abras el agente dentro de `railway run` o `railway shell` vinculado a producción, y considera `CLAUDE_CODE_SUBPROCESS_ENV_SCRUB=1`.
- **Dónde vive cada llave:** nombres sin valores en `.env.example`; llaves de prueba en `.env.local`, fuera de git; llaves reales solo en el gestor de secretos (variables selladas de Railway, Secret Manager de Google Cloud).
- La documentación no menciona `Get-Content` de PowerShell. En Windows, súmale el hook PreToolUse del artículo `ecomlabs.dev/g/hook`.
- Un archivo `.claudeignore` no tiene efecto, y `.gitignore` solo cambia el selector de `@`.

PASO 01

El settings.json del proyecto

Guarda esto en `.claude/settings.json`, en la raíz del repo, y súbelo a git para que todo el equipo tenga las mismas reglas. Validado con Ajv contra el esquema de SchemaStore:

```
{
  "$schema": "https://json.schemastore.org/claude-code-settings.json",
  "permissions": {
    "deny": [
      "Read(.env)",
      "Read(.env.*)",
      "Read(!.env.example)",
      "Read(*.pem)"
    ]
  }
}
```

- Un nombre suelto coincide a cualquier profundidad: `Read(.env)` equivale a `Read(**/.env)`.
- El orden importa: `Read(!.env.example)` va después de `Read(.env.*)`. Un `!` listado primero no recorta nada, y solo recorta reglas del mismo archivo.
- Si ya tienes un `settings.json`, agrega las reglas a su lista `deny`; no lo reemplaces.
- El archivo es JSON estricto: un comentario `//` o una coma final lo rompen.

PASO 02

Una red para todos tus proyectos

En `~/.claude/settings.json` una ruta con una sola barra empieza en `~/.claude`, no en tu proyecto. Para cubrir cualquier carpeta, usa `//`, que es la raíz del sistema (en Windows, todas las unidades):

```
{
  "permissions": {
    "deny": [
      "Read(//**/.env)",
      "Read(//**/.env.*)"
    ]
  }
}
```

> **CUIDADO**
> Aquí la excepción para `.env.example` no funciona: un `!` no alcanza reglas ancladas con `//`, y un archivo de proyecto no puede reabrir lo que bloquea tu archivo de usuario. Con esta red, el agente tampoco ve la plantilla; si la necesita, deja solo la regla del Paso 01.

PASO 03

Comprobar que funciona

1. Abre Claude Code en el proyecto y escribe `/permissions`. El diálogo lista cada regla y el archivo del que sale.
2. Pídele que lea el `.env` y que corra `cat .env`. Las dos peticiones deben terminar bloqueadas. Usa un `.env` con valores falsos para la prueba.
3. Pídele que lea `.env.example`: debe poder.
4. En la terminal, revisa que git no tenga ningún `.env` real:

```
git ls-files | grep -i "\.env"
git log --all --oneline -- .env
```

La primera línea solo debe mostrar `.env.example`. La segunda debe salir vacía; si muestra commits, la llave está en el historial aunque hoy esté en `.gitignore`: pasa a la checklist de rotación.

SI SE FILTRÓ

Checklist de rotación

Una llave que el agente leyó está en la transcripción local (`~/.claude/projects/`, 30 días por defecto) y viajó al modelo. Una que llegó a un commit está en cada clon. Borrar no basta: se rota.

1. **Si mueve dinero o datos de clientes, revócala ya** en el proveedor, aunque falle la app unos minutos.
2. **Crea la llave nueva.** En Stripe, rotar revoca la vieja y crea la de reemplazo; con vencimiento futuro las dos sirven hasta 7 días, así que ante una filtración elige vencimiento inmediato.

3. **Guárdala en el gestor de secretos.** En Railway, edita la variable y despliega el cambio pendiente. En Secret Manager con Cloud Run (versiones ilustrativas):

```
gcloud secrets versions add PAYMENTS_KEY --data-file=-
gcloud run services update my-service \
  --update-secrets=PAYMENTS_SECRET_KEY=PAYMENTS_KEY:2
gcloud secrets versions disable 1 --secret=PAYMENTS_KEY
```

4. **Comprueba que la app funciona** con la llave nueva.
5. **Revoca la llave vieja en el proveedor.** Deshabilitar la versión en Secret Manager no la invalida.
6. **Revisa los registros del proveedor** desde el día en que la llave pudo salir.
7. **Quita la llave de donde quedó**: tu `.env`, archivos o commits que escribió el agente. Si estaba en el historial, sigue la guía de GitHub para quitar datos sensibles; después de rotar, no antes.
8. **Cierra la puerta**: reglas deny del Paso 01 y solo llaves de prueba en tu máquina.

> **CUIDADO**
> Los tres comandos de gcloud están revisados contra la ayuda local de Google Cloud SDK 581.0.0 y la documentación de Google, no ejecutados contra un proyecto. Cambia los nombres por los tuyos y fija siempre una versión: las variables de entorno de Cloud Run se resuelven al arrancar la instancia.

AL TERMINAR

Checklist

- [ ] `.claude/settings.json` bloquea `.env`, `.env.*` y `*.pem`, con la excepción de `.env.example` al final.
- [ ] `/permissions` muestra las reglas y el agente no puede leer ni hacer `cat` del `.env`.
- [ ] En Windows, el hook PreToolUse cubre PowerShell.
- [ ] Tu `.env` local solo tiene llaves de prueba o de solo lectura.
- [ ] Tu carpeta está vinculada a un entorno de desarrollo.
- [ ] Las llaves reales viven en el gestor de secretos, selladas o con versión fija.
- [ ] `git log --all -- .env` sale vacío.
- [ ] Para cada llave sabes dónde se rota y dónde ver sus registros.