

Un subagente que revisa tu código en Claude Code

Un subagente de solo lectura que revisa los archivos que cambiaste con una checklist fija y devuelve hallazgos con archivo, línea y severidad. Incluye un archivo con errores para probarlo y un script que muestra qué herramientas usó de verdad.

PARA	Claude Code en macOS, Linux o Windows
PROBADO CON	Claude Code 2.1.238 · Windows 11 · sesión Haiku, revisor Sonnet
ACTUALIZADA	26 de octubre de 2026
DOCUMENTACIÓN	code.claude.com/docs/en/sub-agents
ARTÍCULO	ecomlabs.dev/g/revisor
AUTOR	Luis Morales · @luismoralesdev

Índice

Antes de empezar	Qué vas a construir
---------------------	---------------------

Paso 01	Crea la carpeta de subagentes
---------	-------------------------------

Paso 02	El archivo del revisor
---------	------------------------

Paso 03	Adapta la checklist a tu proyecto
---------	-----------------------------------

Paso 04	Pruébalo con el archivo de ejemplo
---------	------------------------------------

Paso 05	Comprueba qué hizo de verdad
---------	------------------------------

Paso 06	Comprueba que no edita
---------	------------------------

En el día a día	Cómo pedir una revisión
--------------------	-------------------------

Si algo falla	Solución de problemas
------------------	-----------------------

Al terminar	Checklist
----------------	-----------

Qué vas a construir

Un subagente es un asistente que la sesión principal de Claude Code llama para una tarea concreta. Arranca con un contexto nuevo: no ve tu conversación ni los archivos que Claude ya leyó. Hace su trabajo con sus propias herramientas y a tu sesión solo vuelve el resultado.

El de esta guía es un revisor de código de solo lectura. Le pasas los archivos que cambiaste, los recorre con una checklist de cinco puntos y devuelve un informe con un formato fijo. No puede editar nada porque no tiene las herramientas para hacerlo.

Lo que trae el ZIP

ARCHIVO	PARA QUÉ SIRVE
<code>.claude/agents/revisor-codigo.md</code>	El subagente: frontmatter, checklist y formato del informe.
<code>ejemplo/cupones.ts</code>	Veinte líneas con errores a propósito para probar el revisor. No lo uses en tu tienda.
<code>probar/ver-herramientas.sh</code>	Muestra, desde la transcripción, qué herramientas usó el subagente (bash).
<code>probar/ver-herramientas.ps1</code>	Lo mismo en PowerShell.

Requisitos

- Claude Code instalado y un proyecto donde quieras revisar código.
- Para el script de comprobación: bash (macOS, Linux o Git Bash) o PowerShell 5.1 o 7.

Crea la carpeta de subagentes

Los subagentes de un proyecto viven en `.claude/agents/`, en la raíz del repositorio. Los de `~/ .claude/agents/` valen para todos tus proyectos; si dos tienen el mismo nombre, gana el del proyecto.

```
# macOS, Linux o Git Bash
mkdir -p .claude/agents

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

CUIDADO

Claude Code vigila las carpetas de agentes que existían al abrir la sesión. Si acabas de crear `.claude/agents/` con Claude Code abierto, ciérralo y vuelve a abrirlo una vez. Después, cualquier cambio en el archivo se detecta en unos segundos.

El archivo del revisor

Guárdalo como `.claude/agents/revisor-codigo.md`. Es el mismo archivo que viene en el ZIP.

`.claude/agents/revisor-codigo.md`

```
---
name: revisor-codigo
description: Revisor de código de solo lectura. Úsalo de forma proactiva al terminar un cambio, antes de un
commit o un PR, o cuando el usuario pida una revisión. Pásale la lista de archivos que cambiaron.
tools: Read, Grep, Glob
model: sonnet
---
```

Eres un revisor de código. Solo lees: no tienes herramientas para editar y no reescribes archivos completos en tu respuesta.

Revisa solo los archivos que te indiquen en la tarea. Si no te indican ninguno, responde «Necesito la lista de archivos que cambiaron» y termina. Puedes abrir otros archivos para entender el contexto (lo que importan, sus tests), pero no los revises.

Recorre esta lista en cada archivo:

1. Seguridad: secretos en el código, entradas sin validar, URLs, consultas o comandos armados con texto que llega de fuera, datos personales en logs.
2. Manejo de errores: catch vacíos, respuestas HTTP que no se revisan, errores que se pierden, código que sigue como si nada hubiera fallado.
3. Tests: lógica nueva o cambiada sin un test que la cubra. Antes de decir que no hay tests, búscalos con Glob y escribe en el hallazgo el patrón que usaste.
4. Nombres: funciones o variables cuyo nombre no dice lo que hacen.
5. Rendimiento: llamadas de red o consultas dentro de un bucle, trabajo que se repite sin necesidad.

Responde con este formato y nada más:

Resumen

Una o dos frases: qué revisaste y si lo dejarías pasar a un commit.

Hallazgos

- [bloqueante|importante|menor] archivo:linea – qué pasa. Por qué importa. Cómo arreglarlo.

Ordénalos de bloqueante a menor. Si no hay ninguno, escribe «Sin hallazgos».

Lista

- Seguridad: ok | N hallazgos
- Manejo de errores: ok | N hallazgos
- Tests: ok | N hallazgos
- Nombres: ok | N hallazgos
- Rendimiento: ok | N hallazgos

No inventes problemas para llenar la lista. Si un hallazgo depende de código que no viste, dilo en vez de suponer.

El frontmatter, campo por campo

CAMPO	VALOR	POR QUÉ
<code>name</code>	<code>revisor-codigo</code>	Es el nombre con el que lo llamas. Único, sin <code>:</code> .
<code>description</code>	Cuándo usarlo y qué necesita	Claude la lee para decidir si delega solo. Una descripción vaga no se usa nunca.
<code>tools</code>	<code>Read</code> , <code>Grep</code> , <code>Glob</code>	Lo que lo hace de solo lectura. Si omites el campo, hereda todas las herramientas, Edit y Bash incluidas.
<code>model</code>	<code>sonnet</code>	El modelo del revisor, independiente del de tu sesión. También acepta <code>opus</code> , <code>haiku</code> , <code>fable</code> , un ID completo o <code>inherit</code> .

El prompt

Debajo del frontmatter va el prompt del subagente. Tiene tres partes: qué revisar (solo los archivos que le indiquen), cómo revisar (la lista de cinco puntos) y cómo responder (un formato fijo). La última línea le pide que no invente problemas para llenar la lista, y que diga cuándo un hallazgo depende de código que no vio.

PASO 03

Adapta la checklist a tu proyecto

Los cinco puntos sirven para casi cualquier proyecto de JavaScript o TypeScript. Puedes cambiarlos, pero conserva dos reglas: que cada punto diga qué buscar, no solo un tema, y que el formato del informe no cambie.

Ejemplos de puntos para una tienda

- **Checkout:** «Cualquier cambio en `src/checkout/` o en precios, descuentos o impuestos es al menos importante. Señala cálculos con números de coma flotante en dinero.»
- **Temas de Shopify:** «En archivos `.liquid`, señala bucles sobre `collection.products` sin `limit` y textos visibles que no pasan por traducciones.»
- **Accesibilidad:** «Imágenes sin `alt`, botones sin texto accesible, campos de formulario sin `label`.»
- **Analítica:** «Cambios que quitan o renombran eventos del `dataLayer`.»

Si un punto crece mucho, sepáralo en otro subagente (por ejemplo, `revisor-accesibilidad`) con su propia `description`. Escribe descripciones que no se pisen, para que Claude sepa a cuál delegar cada tarea.

La regla de las evidencias

En nuestras pruebas, la primera versión del revisor afirmó haber buscado tests con Glob sin hacerlo: su transcripción solo tenía una lectura del archivo. Desde que el punto 3 le pide escribir el patrón que usó, la búsqueda aparece en la transcripción. Si añades puntos que dependen de buscar algo, pide la misma evidencia.

Pruébalo con el archivo de ejemplo

1. Copia `ejemplo/cupones.ts` a `src/cupones.ts`.
2. Abre Claude Code en el proyecto y escribe: «Usa el subagente revisor-codigo para revisar `src/cupones.ts` y muéstrame su informe tal cual».
3. Espera el informe. Pide «tal cual» porque, si no, la sesión principal suele resumirlo con sus propias palabras.

El archivo tiene cinco problemas puestos a propósito. Un informe correcto los menciona todos:

PROBLEMA	LÍNEA	PUNTO DE LA CHECKLIST
Clave escrita en el código y enviada en la URL	4 y 15	Seguridad
<code>codigo</code> metido en la URL sin codificar	15	Seguridad
<code>catch (e) {}</code> vacío	18	Manejo de errores
Un <code>fetch</code> por línea dentro del bucle	10 a 12	Rendimiento
Ningún test para <code>calcular</code>	—	Tests

En nuestras corridas, el revisor encontró los cinco y además señaló que ninguna respuesta de `fetch` revisa `r.ok` y que el total puede quedar negativo. El punto de nombres varió: una vez marcó `calcular` como demasiado genérico y las otras lo dejó en «ok». Cada revisión de este archivo costó unos 0,15 USD con la sesión en Haiku y el revisor en Sonnet.

Comprueba qué hizo de verdad

Claude Code guarda la transcripción de cada subagente en

`~/.claude/projects/<proyecto>/<sesión>/subagents/`, junto a un `.meta.json` con su nombre. Estos scripts leen la más reciente y listan cada herramienta que usó.

`probar/ver-herramientas.sh`

```
#!/usr/bin/env bash
# ver-herramientas.sh: muestra qué herramientas usó de verdad un subagente,
# leyendo la transcripción que Claude Code guarda en ~/.claude/projects.
# Uso:  bash probar/ver-herramientas.sh                (el subagente más reciente)
#       bash probar/ver-herramientas.sh <agent-xxxx.jsonl>

archivo=${1:-$(ls -t "$HOME"/.claude/projects/*/subagents/agent-*.jsonl 2>/dev/null | head -n 1)}
[ -f "$archivo" ] || { echo "No encuentro transcripciones de subagentes en ~/.claude/projects" >&2; exit 1; }

echo "Transcripción: $archivo"
meta="${archivo%.jsonl}.meta.json"
[ -f "$meta" ] && grep -o '"agentType": "[^"]*" ' "$meta" | sed 's/"agentType": "/Subagente: /; s/"$//'
```

```
# Una línea por llamada: la herramienta y lo que recibió.
grep -o '{"type":"tool_use"[^}]*}' "$Archivo" | sed -E 's/.*"name":"([^\"]+)", "input":/\1 /'
```

probar/ver-herramientas.ps1

```
# ver-herramientas.ps1: muestra qué herramientas usó de verdad un subagente,
# leyendo la transcripción que Claude Code guarda en ~/.claude/projects.
# Uso: powershell -NoProfile -ExecutionPolicy Bypass -File probar\ver-herramientas.ps1
#      (añade -Archivo <agent-xxxx.jsonl> para revisar uno concreto)
# Los mensajes van sin acentos: PowerShell 5.1 lee este archivo con la página
# de códigos del sistema.

param([string]$Archivo)

if (-not $Archivo) {
    $ultimo = Get-ChildItem -Path "$HOME\.claude\projects\*\*\subagents\agent-*.jsonl" -ErrorAction
    SilentlyContinue |
        Sort-Object LastWriteTime -Descending | Select-Object -First 1
    if (-not $ultimo) {
        [Console]::Error.WriteLine('No encuentro transcripciones de subagentes en ~/.claude/projects')
        exit 1
    }
    $Archivo = $ultimo.FullName
}

if (-not (Test-Path -LiteralPath $Archivo)) {
    [Console]::Error.WriteLine("No existe: $Archivo")
    exit 1
}

"Transcripcion: $Archivo"

$meta = $Archivo -replace '\.jsonl$', '.meta.json'
if (Test-Path -LiteralPath $meta) {
    'Subagente: ' + (Get-Content -LiteralPath $meta -Raw | ConvertFrom-Json).agentType
}

# Una línea por llamada: la herramienta y lo que recibio.
foreach ($linea in Get-Content -LiteralPath $Archivo -Encoding UTF8) {
    $m = $linea | ConvertFrom-Json
    foreach ($c in @($m.message.content)) {
        if ($c.type -eq 'tool_use') { '{0} {1}' -f $c.name, ($c.input | ConvertTo-Json -Compress) }
    }
}
```

```
bash probar/ver-herramientas.sh
powershell -NoProfile -ExecutionPolicy Bypass -File probar\ver-herramientas.ps1
```

Con la segunda versión del revisor, la salida de nuestra prueba fue esta (ruta recortada):

```
Subagente: revisor-codigo
Glob {"pattern":"**/cupones.ts"}
Read {"file_path":"...\src\cupones.ts"}
Glob {"pattern":"**/*cupones*"}
Glob {"pattern":"**/*.test.ts"}
Glob {"pattern":"**/*.spec.ts"}
Glob {"pattern":"**/_tests_/**"}
```

Si el informe dice que buscó tests y aquí no hay ninguna línea `Glob`, el informe no es fiable en ese punto. Nunca debería aparecer `Edit`, `Write` ni `Bash`.

PASO 06

Comprueba que no edita

Arranca una sesión con el revisor como agente principal y pídele un cambio, con las ediciones autorizadas de antemano:

```
claude --agent revisor-codigo --permission-mode acceptEdits
```

Escribe: «Corrige el catch vacío de src/cupones.ts». En nuestra prueba respondió que no puede editar archivos porque es un revisor de solo lectura con Read, Grep y Glob, y entregó el arreglo sugerido para aplicarlo a mano. Después, `git status` no debe mostrar cambios en `src/cupones.ts`.

Cuando termines, borra `src/cupones.ts`.

EN EL DÍA A DÍA

Cómo pedir una revisión

CUÁNDO	QUÉ ESCRIBIR
Antes de un commit	«Usa el subagente revisor-codigo para revisar los archivos que cambiaron según git diff --name-only»
Una carpeta concreta	<code>@revisor-codigo (agent)"</code> revisa src/checkout/
Una sesión solo de revisión	<code>claude --agent revisor-codigo</code>
Después de la revisión	«Aplica los hallazgos bloqueantes del informe; los demás déjalos para después»

La mención con `@` se elige en la lista que aparece al escribir la arroba. Asegura que la tarea va a ese subagente y no a otro. El revisor no corrige: las correcciones las hace la sesión principal, con el informe delante, y tú decides cuáles.

Solución de problemas

SÍNTOMA	CAUSA PROBABLE	ARREGLO
Claude no encuentra el subagente	La carpeta se creó con la sesión abierta	Reinicia Claude Code una vez
Claude revisa él mismo en lugar de delegar	No nombraste el subagente	Usa su nombre o la mención @
El revisor edita archivos	Falta <code>tools</code> en el frontmatter	Agrega <code>tools: Read, Grep, Glob</code>
Responde «Necesito la lista de archivos»	La tarea no nombró archivos	Pásale rutas o pide que use <code>git diff --name-only</code>
El informe llega resumido o con otro formato	La sesión principal lo reescribió	Pide «muéstrame su informe tal cual»
Dice que buscó tests pero no hay <code>Glob</code>	El modelo afirmó algo que no hizo	Conserva la regla de las evidencias del paso 3
<code>ver-herramientas</code> no encuentra nada	Ningún subagente ha corrido aún	Haz una revisión primero

Checklist

- ☐ `.claude/agents/revisor-codigo.md` está en el proyecto y versionado.
- ☐ El frontmatter tiene `name`, `description` y `tools: Read, Grep, Glob`.
- ☐ La `description` dice cuándo usarlo y que necesita la lista de archivos.
- ☐ La revisión de `ejemplo/cupones.ts` encontró los cinco problemas.
- ☐ `ver-herramientas` muestra solo `Read`, `Grep` y `Glob`.
- ☐ Con `--agent`, se negó a editar y `git status` quedó limpio.
- ☐ Borraste `src/cupones.ts`.